
jMetalPy Documentation

Release

Antonio J. Nebro

Sep 25, 2018

Contents:

1	Contributing	3
2	jMetal	13
3	About	687
4	Installation steps	689

Warning: Documentation is WIP!! Some information may be missing.

Contributions to the jMetalPy project are welcome. Please, take into account the following guidelines (all developers should follow these guidelines):

1.1 Git WorkFlow

We have a set of branches on the remote Git server. Some branches are temporary, and others are constant throughout the life of the repository.

- **Branches always present in the repository:**

- ***master***: You have the latest released to production, receive merges from the develop branch, or merge from a *hotfix*

- * Do I have to put a TAG when doing a merge from develop to master? yes

- * Do I have to put a TAG when doing a merge from a hotfix branch to master? yes

- * After merge from a hotfix to master, do I have to merge from master to develop? yes

- ***develop***: It is considered the “Next Release”, receives merges from branches of each developer, either corrections (*fix*) or new features (*feature*).

- **Temporary branches:**

- ***feature/<task-id>-<description>***: When we are doing a development, we create a local branch with the prefix “*feature*”

- * Where does this branch emerge? This branch always emerge from the develop branch

- * When I finish the development in my feature branch, which branch to merge into?: You always merge feature branch into develop branch

- ***fix/<task-id>-<description>***: When we are making a correction, we create a local branch with the prefix “*fix*”, then

- * Where does this branch emerge? This branch always emerge from the develop branch

- * When I finish the correction in my fix branch, which branch to merge into?: You always merge feature branch into develop branch

– *hotfix/<task-id>-<description>*: When we are correcting an emergency incidence in production, we create a local

- * Where does this branch emerge?: This branch always emerge from the master branch
- * When I finish the correction in my hotfix branch, which branch to merge into?: This branch always emerge from the master and develop branch

- **Steps to follow when you are creating or going to work on a branch of any kind (feature / fix / hotfix):**

1. After you create your branch (feature / fix / hotfix) locally, upload it to the remote Git server. The integration system will verify your code from the outset.
2. Each time you commit, as much as possible, you send a push to the server. Each push will trigger the automated launch of the tests, etc.
3. Once the development is finished, having done a push to the remote Git server, and that the test phase has passed without problem, you create an [pull request](#).

Note: Do not forget to remove your branch (feature / fix / hotfix) once the merge has been made.

Some useful Git commands:

- `git fetch --prune`: Cleaning branches removed and bringing new branches

1.2 PEP8!

It is really important to follow some standards when a team develops an application. If all team members format the code in the same format, then it is much easier to read the code. PEP8 is Python's style guide. It's a set of rules for how to format your Python code.

Some style rules:

- Package and module names: Modules should have short, **all-lowercase** names. Underscores can be used in the module name if it improves readability. Python packages should also have short, **all-lowercase** names, although the use of underscores is discouraged. In Python, a module is a file with the suffix `.py`.
- Class names: Class names should normally use the **CapWords** convention.
- Method names and instance variables: **Lowercase with words separated by underscores** as necessary to improve readability.

There are many more style standards in PEP8 so, please, refer to [PEP8 documentation](#) . The most appropriate is to use an IDE that has support for PEP8. For example, [PyCharm](#).

1.3 Object-oriented programming

Object-oriented programming should be the single programming paradigm used. Avoiding as far as possible, imperative and functional programming.



```
# Object-oriented programming

class Imprint(object):
    def world(self) -> str:
        return "World"

    def hello(self) -> str:
        return "Hello"

    def hello_world(self) -> None:
        print(self.hello(), self.world())

imprint = Imprint()
imprint.hello_world()
```




```
# Functional programming

def world() -> str:
    return "World"

def hello() -> str:
    return "Hello"


def print_hello_world() -> None:
    print(hello(), world())

print_hello_world()
```



```
# Imperative programming

world = "World"
hello = "Hello"
hello_world = hello + ' ' + world
print(hello_world)
```



In classes, we directly access the attributes, which are usually defined as public.

```
class Circle(object):  
  
    def __init__(self, radius: int):  
        self.radius = radius
```



Only when we want to **implement additional logic in the accesses to the attributes** we define getter/setter methods, but **always by using the `*property*` annotation or the `*property*` function**:

```
class Circle(object):  
  
    def __init__(self):  
        self.__radius = None  
  
    @property  
    def radius(self) -> int:  
        print("Accessing the radius attribute by get")  
        return self.__radius  
  
    @radius.setter  
    def radius(self, radius: int) -> None:  
        print("Accessing the radius attribute by set")  
        # Logic to validate  
        if radius < 0:  
            raise ValueError("The radius value must be a positive integer")  
        self.__radius = radius
```



```
class Circle(object):  
  
    def __init__(self):  
        self.__radius = None  
  
    def __get_radius(self) -> int:  
        print("Accessing the radius attribute by get")  
        return self.__radius  
  
    def __set_radius(self, radius: int) -> None:  
        print("Accessing the radius attribute by set")  
        # Logic to validate  
        if radius < 0:  
            raise ValueError("The radius value must be a positive integer")  
        self.__radius = radius  
  
    radius = property(fget=__get_radius, fset=__set_radius)
```



By using `*property*`, we continue to access the attributes directly:

```
circle = Circle()
circle.radius = 3
print(circle.radius)
```



Do not use getter/setter methods without the *property* annotation or the *property* function:

```
class Circle(object):

    def __init__(self):
        self.__radius = None

    def get_radius(self) -> int:
        return self.__radius

    def set_radius(self, radius: int) -> None:
        # Logic to validate
        if radius < 0:
            raise ValueError("The radius value must be a positive integer")
        self.__radius = radius
```



Since this way of accessing the attribute is not commonly used in Python:

```
circle = Circle()
circle.set_radius(3)
print(circle.get_radius())
```



1.4 Structure


Python is not Java. In Java you cannot, by design, have more than one class in a file. In Python, you can do it.

In Python, it is appropriate to group several classes into a single .py file. For that reason, the .py files are called **modules**.

1.5 Python 3.6


We **always** define types in the parameters of the arguments and the return value:

```
class Car(object):  
  
    def __init__(self):  
        self.fuel = 0  
        self.battery = 0  
  
    def refuel(self, new_fuel: int):  
        self.fuel += new_fuel  
  
    def recharge(self, new_energy: int):  
        self.battery += new_energy  
  
    def status(self) -> Tuple[int, int]:  
        return self.fuel, self.battery
```



We can define abstract classes (ABCs) in Python:

```
class AbstractClass(metaclass=ABCMeta):  
  
    @abstractmethod  
    def abstract_method(self) -> float:  
        pass  
  
class ImplementingClass(AbstractClass):  
  
    def abstract_method(self) -> float:  
        # implementation ...
```



In the case that we want to define an **interface** class, it is done in the same way. We just have to define all the methods of the class as abstract.

Example of use of generic types:

```

T = TypeVar('T') # <- Can be anything
S = TypeVar('S', int, float) # <- Must be int or float

class Car(object):

    def __init__(self, fuel: S, battery: S, model: T):
        self.fuel = fuel
        self.battery = battery
        self.model = model

    def refuel(self, new_fuel: S) -> None:
        self.fuel += new_fuel

    def recharge(self, new_energy: S) -> None:
        self.battery += new_energy

    def status(self) -> Tuple[S, S]:
        return self.fuel, self.battery

```

In the code below, the IDE displays a **warning**, since although the 2nd parameter is a float type, which is a type allowed in the definition of the generic type X, it is not of the same type as the first, since the first 2 parameters must be of the same generic type (S):

```
car1 = Car(3, 3.44, "FORD-F-150")
```

In the code below, the IDE displays a **warning**, since the 2nd parameter is a type not allowed in the definition of the generic type (*TypeVar('S', int, float)*):

```
car2 = Car(3, "hello", "FORD-F-150")
```

When the class inherits from *Generic[...]*, the **class is defined as generic**. In this way we can indicate the types that will have the values of the generic types, when using the class as type. Look at the *add_car()* method of the *Parking* class.

Note: The generic classes inherit from *abc.ABCMeta*, so they are abstract classes and **abstract methods can be used**.

```
T = TypeVar('T') # <- Can be anything
S = TypeVar('S', int, float) # <- Must be int or float
```


```
class CarGeneric(Generic[S, T]):

    def __init__(self, fuel: S, battery: S, model: T):
        self.fuel = fuel
        self.battery = battery
        self.model = model

    def refuel(self, new_fuel: S) -> None:
        self.fuel += new_fuel

    def recharge(self, new_energy: S) -> None:
        self.battery += new_energy


    def status(self) -> Tuple[S, S]:
        return self.fuel, self.battery
```



```
class Parking(object):

    def __init__(self):
        self.car_list = list()

    def add_car(self, new_car: CarGeneric[int, str]) -> None:
        self.car_list.append(new_car)
```




In the code below, the IDE displays a **warning** in the call to the `add_car()` method when adding the car, since the 3rd parameter of the init must be a *str* type, as defined in the `add_car()` method of the *Parking* class.

```
car3 = CarGeneric(3, 4, 777)
parking = Parking()
parking.add_car(car3)
```

When inheriting from generic classes, some type variables could be fixed:

```
T = TypeVar('T')


class MyClass(CarGeneric[str, T]):
    pass
```



Example of inheritance from non-generic class to generic class:

```
class A(object):
    pass


class B(A, Generic[S, T]):
    pass
```



Example of inheritance from generic class to another generic class:

```
class A(Generic[S, T]):
    pass

class B(A[S, T], Generic[S, T]):
    pass
```



1.6 Create automatic documentation files with Sphinx

First, you need to know how to correctly document your code. It is **important** to follow these simple rules in order to automatically create good documentation for the project.

When you create a new module file (testDoc.py in this example), you should mention it using this format:

```
"""
.. module:: testDoc
   :platform: Unix, Windows
   :synopsis: A useful module indeed.

.. moduleauthor:: Andrew Carter <andrew@invalid.com>
"""

class testDoc(object):
    """We use this as a public class example class.

    This class is ruled by the very trendy important method :func:`public_fn_with_
    ↪ sphinxy_docstring`.
```

```

.. note::
    An example of intersphinx is this: you cannot use :mod:`pickle` on this_
↪class.
    """

    def __init__(self):
        pass

```

This code snippet generates the following documentation:

jmetal.algorithm.singleobjective.testDoc module

`class jmetal.algorithm.singleobjective.testDoc.testDoc(foo: str, bar: str)`

Bases: `object`

We use this as a public class example class.

This class is ruled by the very trendy important method `public_fn_with_sphixy_docstring()`.

Note: An example of intersphinx is this: you **cannot** use `pickle` on this class.

Now, you can document your methods using the following syntax:

```

def public_fn_with_sphixy_docstring(self, name: str, state: bool = False) -> int:
    """This function does something.

    :param name: The name to use.
    :type name: str.
    :param state: Current state to be in.
    :type state: bool.
    :returns: int -- the return code.
    :raises: AttributeError, KeyError
    """
    return 0

def public_fn_without_docstring(self):
    return True

```

And the produced output doc will be:

public_fn_with_sphixy_docstring(name: str, state: bool = False) → int

This function does something.

Parameters:	<ul style="list-style-type: none"> • name (str.) – The name to use. • state (bool.) – Current state to be in.
Returns:	int – the return code.
Raises:	AttributeError, KeyError

public_fn_without_docstring()

As you may notice, if you don't use any docstring, the method documentation will be empty.

2.1 org.uma.jmetal.algorithm

2.1.1 Algorithm

public interface **Algorithm**<Result> extends *Runnable*, *Serializable*, *DescribedEntity*

Interface representing an algorithm

Author Antonio J. Nebro

Parameters

- <**Result**> – Result

Methods

getResult

Result **getResult** ()

run

void **run** ()

2.1.2 InteractiveAlgorithm

public interface **InteractiveAlgorithm**<S, R> extends *Algorithm*<R>

Methods

updatePointOfInterest

public void **updatePointOfInterest** ([List<Double>](#) *newReferencePoints*)

2.2 org.uma.jmetal.algorithm.impl

2.2.1 AbstractCoralReefsOptimization

public abstract class **AbstractCoralReefsOptimization**<S, R> implements [Algorithm](#)<R>

Abstract class representing a Coral Reefs Optimization Algorithm Reference: S. Salcedo-Sanz, J. Del Ser, S. Gil-López, I. Landa-Torres and J. A. Portilla-Figueras, “The coral reefs optimization algorithm: an efficient meta-heuristic for solving hard optimization problems,” 15th Applied Stochastic Models and Data Analysis International Conference, Mataró, Spain, June, 2013.

Author Inacio Medeiros

Fields

comparator

protected [Comparator](#)<S> **comparator**

coordinates

protected [List](#)<[Coordinate](#)> **coordinates**

crossoverOperator

protected [CrossoverOperator](#)<S> **crossoverOperator**

mutationOperator

protected [MutationOperator](#)<S> **mutationOperator**

population

protected [List](#)<S> **population**

selectionOperator

protected [SelectionOperator](#)<[List](#)<S>, S> **selectionOperator**

Constructors

AbstractCoralReefsOptimization

```
public AbstractCoralReefsOptimization (Comparator<S> comparator, SelectionOperator<List<S>, S> selectionOperator, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, int n, int m, double rho, double fbs, double fa, double pd, int attemptsToSettle)
```

Constructor

Parameters

- **comparator** – Object for comparing two solutions
- **selectionOperator** – Selection Operator
- **crossoverOperator** – Crossover Operator
- **mutationOperator** – Mutation Operator
- **n** – width of Coral Reef Grid
- **m** – height of Coral Reef Grid
- **rho** – Percentage of occupied reef
- **fbs** – Percentage of broadcast spawners
- **fa** – Percentage of budders
- **pd** – Probability of depredation
- **attemptsToSettle** – number of attempts a larvae has to try to settle reef

Methods

asexualReproduction

```
protected abstract List<S> asexualReproduction (List<S> brooders)
```

createInitialPopulation

```
protected abstract List<S> createInitialPopulation ()
```

depredation

```
protected abstract List<S> depredation (List<S> population, List<Coordinate> coordinates)
```

evaluatePopulation

```
protected abstract List<S> evaluatePopulation (List<S> population)
```

generateCoordinates

protected abstract `List<Coordinate> generateCoordinates ()`

getAttemptsToSettle

public int `getAttemptsToSettle ()`

getCoordinates

public `List<Coordinate> getCoordinates ()`

getFa

public double `getFa ()`

getFbr

public double `getFbr ()`

getFbs

public double `getFbs ()`

getFd

public double `getFd ()`

getM

public int `getM ()`

getN

public int `getN ()`

getPd

public double `getPd ()`

getPopulation

public `List<S> getPopulation ()`

getPopulationSize

```
public int getPopulationSize ()
```

getResult

```
public abstract R getResult ()
```

getRho

```
public double getRho ()
```

initProgress

```
protected abstract void initProgress ()
```

isStoppingConditionReached

```
protected abstract boolean isStoppingConditionReached ()
```

larvaeSettlementPhase

```
protected abstract List<S> larvaeSettlementPhase (List<S> larvae, List<S> population,
                                                    List<Coordinate> coordinates)
```

run

```
public void run ()
```

selectBroadcastSpawners

```
protected abstract List<S> selectBroadcastSpawners (List<S> population)
```

setCoordinates

```
public void setCoordinates (List<Coordinate> coordinates)
```

setPopulation

```
public void setPopulation (List<S> population)
```

sexualReproduction

```
protected abstract List<S> sexualReproduction (List<S> broadcastSpawners)
```

updateProgress

protected abstract void **updateProgress** ()

2.2.2 AbstractCoralReefsOptimization.Coordinate

public static class **Coordinate** implements [Comparable<Coordinate>](#)
Represents a Coordinate in Coral Reef Grid

Author inacio-medeiros

Constructors

Coordinate

public **Coordinate** (int *x*, int *y*)
Constructor

Parameters

- **x** – Coordinate's x-position
- **y** – Coordinate's y-position

Methods

compareTo

public int **compareTo** (*Coordinate arg0*)

equals

public boolean **equals** (*Object obj*)

getX

public int **getX** ()
Retrieves Coordinate's x-position
Returns Coordinate's x-position

getY

public int **getY** ()
Retrieves Coordinate's y-position
Returns Coordinate's y-position

setX

public void **setX** (int x)
 Sets Coordinate's x-position to a new value

Parameters

- **x** – new value for Coordinate's x-position

setY

public void **setY** (int y)
 Sets Coordinate's y-position to a new value

Parameters

- **x** – new value for Coordinate's y-position

2.2.3 AbstractDifferentialEvolution

public abstract class **AbstractDifferentialEvolution**<Result> extends *AbstractEvolutionaryAlgorithm*<*DoubleSolution*, Result>
 Abstract class representing differential evolution (DE) algorithms

Author Antonio J. Nebro

Fields**crossoverOperator**

protected *DifferentialEvolutionCrossover* **crossoverOperator**

selectionOperator

protected *DifferentialEvolutionSelection* **selectionOperator**

2.2.4 AbstractEvolutionStrategy

public abstract class **AbstractEvolutionStrategy**<S, Result> extends *AbstractEvolutionaryAlgorithm*<S, Result>
 Abstract class representing an evolution strategy algorithm

Author Antonio J. Nebro

Fields**mutationOperator**

protected *MutationOperator*<S> **mutationOperator**

Constructors

AbstractEvolutionStrategy

public **AbstractEvolutionStrategy** (*Problem*<S> *problem*)
Constructor

Parameters

- **problem** – The problem to solve

Methods

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

2.2.5 AbstractEvolutionaryAlgorithm

public abstract class **AbstractEvolutionaryAlgorithm**<S, R> implements *Algorithm*<R>
Abstract class representing an evolutionary algorithm

Author Antonio J. Nebro

Parameters

- <S> – Solution
- <R> – Result

Fields

population

protected *List*<S> **population**

problem

protected *Problem*<S> **problem**

Methods

createInitialPopulation

protected abstract *List*<S> **createInitialPopulation** ()

evaluatePopulation

protected abstract *List*<S> **evaluatePopulation** (*List*<S> *population*)

getPopulation

```
public List<S> getPopulation ()
```

getProblem

```
public Problem<S> getProblem ()
```

getResult

```
public abstract R getResult ()
```

initProgress

```
protected abstract void initProgress ()
```

isStoppingConditionReached

```
protected abstract boolean isStoppingConditionReached ()
```

replacement

```
protected abstract List<S> replacement (List<S> population, List<S> offspringPopulation)
```

reproduction

```
protected abstract List<S> reproduction (List<S> population)
```

run

```
public void run ()
```

selection

```
protected abstract List<S> selection (List<S> population)
```

setPopulation

```
public void setPopulation (List<S> population)
```

setProblem

```
public void setProblem (Problem<S> problem)
```

updateProgress

protected abstract void **updateProgress** ()

2.2.6 AbstractGeneticAlgorithm

public abstract class **AbstractGeneticAlgorithm**<S, Result> extends *AbstractEvolutionaryAlgorithm*<S, Result>
Abstract class representing a genetic algorithm

Author Antonio J. Nebro

Fields

crossoverOperator

protected *CrossoverOperator*<S> **crossoverOperator**

maxPopulationSize

protected int **maxPopulationSize**

mutationOperator

protected *MutationOperator*<S> **mutationOperator**

selectionOperator

protected *SelectionOperator*<List<S>, S> **selectionOperator**

Constructors

AbstractGeneticAlgorithm

public **AbstractGeneticAlgorithm** (*Problem*<S> *problem*)

Constructor

Parameters

- **problem** – The problem to solve

Methods

checkNumberOfParents

protected void **checkNumberOfParents** (*List*<S> *population*, int *numberOfParentsForCrossover*)

A crossover operator is applied to a number of parents, and it assumed that the population contains a valid number of solutions. This method checks that.

Parameters

- `population` –
- `numberOfParentsForCrossover` –

`createInitialPopulation`

protected `List<S>` `createInitialPopulation()`

This method implements a default scheme create the initial population of genetic algorithm

`getCrossoverOperator`

public `CrossoverOperator<S>` `getCrossoverOperator()`

`getMaxPopulationSize`

public int `getMaxPopulationSize()`

`getMutationOperator`

public `MutationOperator<S>` `getMutationOperator()`

`getSelectionOperator`

public `SelectionOperator<List<S>, S>` `getSelectionOperator()`

`reproduction`

protected `List<S>` `reproduction(List<S> population)`

This methods iteratively applies a `CrossoverOperator` a `MutationOperator` to the population to create the offspring population. The population size must be divisible by the number of parents required by the `CrossoverOperator`; this way, the needed parents are taken sequentially from the population. No limits are imposed to the number of solutions returned by the `CrossoverOperator`.

Parameters

- `population` –

Returns The new created offspring population

`selection`

protected `List<S>` `selection(List<S> population)`

This method iteratively applies a `SelectionOperator` to the population to fill the mating pool population.

Parameters

- `population` –

Returns The mating pool population

setMaxPopulationSize

public void **setMaxPopulationSize** (int *maxPopulationSize*)

2.2.7 AbstractParticleSwarmOptimization

public abstract class **AbstractParticleSwarmOptimization**<S, Result> implements *Algorithm*<Result>
Abstract class representing a PSO algorithm

Author Antonio J. Nebro

Methods

createInitialSwarm

protected abstract *List*<S> **createInitialSwarm** ()

evaluateSwarm

protected abstract *List*<S> **evaluateSwarm** (*List*<S> *swarm*)

getResult

public abstract Result **getResult** ()

getSwarm

public *List*<S> **getSwarm** ()

initProgress

protected abstract void **initProgress** ()

initializeLeader

protected abstract void **initializeLeader** (*List*<S> *swarm*)

initializeParticlesMemory

protected abstract void **initializeParticlesMemory** (*List*<S> *swarm*)

initializeVelocity

protected abstract void **initializeVelocity** (*List*<S> *swarm*)

isStoppingConditionReached

protected abstract boolean **isStoppingConditionReached**()

perturbation

protected abstract void **perturbation**(List<S> swarm)

run

public void **run**()

setSwarm

public void **setSwarm**(List<S> swarm)

updateLeaders

protected abstract void **updateLeaders**(List<S> swarm)

updateParticlesMemory

protected abstract void **updateParticlesMemory**(List<S> swarm)

updatePosition

protected abstract void **updatePosition**(List<S> swarm)

updateProgress

protected abstract void **updateProgress**()

updateVelocity

protected abstract void **updateVelocity**(List<S> swarm)

2.2.8 AbstractScatterSearch

public abstract class **AbstractScatterSearch**<S, R> implements *Algorithm*<R>
Abstract class representing a scatter search algorithm

Author Antonio J. Nebro

Parameters

- <S> – Solution

- <R> – Result

Methods

diversificationGeneration

public abstract S **diversificationGeneration** ()

getPopulation

public List<S> **getPopulation** ()

getPopulationSize

public int **getPopulationSize** ()

getResult

public abstract R **getResult** ()

improvement

public abstract S **improvement** (S *solution*)

initializationPhase

public void **initializationPhase** ()

Initialization phase of the scatter search: the population is filled with diverse solutions that have been improved.

Returns The population

isStoppingConditionReached

public abstract boolean **isStoppingConditionReached** ()

referenceSetUpdate

public abstract void **referenceSetUpdate** ()

referenceSetUpdate

public abstract void **referenceSetUpdate** (S *solution*)

restart

```
public abstract void restart ()
```

restartConditionIsFulfilled

```
public abstract boolean restartConditionIsFulfilled (List<S> solutionList)
```

run

```
public void run ()
```

setPopulation

```
public void setPopulation (List<S> population)
```

setPopulationSize

```
public void setPopulationSize (int populationSize)
```

solutionCombination

```
public abstract List<S> solutionCombination (List<List<S>> population)
```

subsetGeneration

```
public abstract List<List<S>> subsetGeneration ()
```

2.3 org.uma.jmetal.algorithm.multiobjective.abys

2.3.1 ABYSS

```
public class ABYSS extends AbstractScatterSearch<DoubleSolution, List<DoubleSolution>>
```

This class implements the AbYSS algorithm, a multiobjective scatter search metaheuristics, which is described in: A.J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, A. Beham “AbYSS: Adapting Scatter Search to Multiobjective Optimization.” IEEE Transactions on Evolutionary Computation. Vol. 12, No. 4 (August 2008), pp. 439-457

Author Antonio J. Nebro , Cristobal Barba

Fields**archive**

```
protected Archive<DoubleSolution> archive
```

archiveSize

protected final int **archiveSize**

crossover

protected *CrossoverOperator*<*DoubleSolution*> **crossover**

crowdingDistanceComparator

protected *Comparator*<*DoubleSolution*> **crowdingDistanceComparator**

distanceToSolutionListAttribute

protected *DistanceToSolutionListAttribute* **distanceToSolutionListAttribute**

dominanceComparator

protected *Comparator*<*DoubleSolution*> **dominanceComparator**

equalComparator

protected *Comparator*<*DoubleSolution*> **equalComparator**

evaluations

protected int **evaluations**

fitnessComparator

protected *Comparator*<*DoubleSolution*> **fitnessComparator**

frequency

protected int[][] **frequency**

localSearch

protected *LocalSearchOperator*<*DoubleSolution*> **localSearch**

marked

protected *MarkAttribute* **marked**

maxEvaluations

protected final int **maxEvaluations**

numberOfSubRanges

protected int **numberOfSubRanges**

These variables are used in the diversification method.

problem

protected final *Problem*<*DoubleSolution*> **problem**

randomGenerator

protected *JMetalRandom* **randomGenerator**

referenceSet1

protected *List*<*DoubleSolution*> **referenceSet1**

referenceSet1Size

protected final int **referenceSet1Size**

referenceSet2

protected *List*<*DoubleSolution*> **referenceSet2**

referenceSet2Size

protected final int **referenceSet2Size**

reverseFrequency

protected int[][] **reverseFrequency**

strengthRawFitness

protected *StrengthRawFitness*<*DoubleSolution*> **strengthRawFitness**

sumOfFrequencyValues

protected int[] **sumOfFrequencyValues**

sumOfReverseFrequencyValues

protected int[] **sumOfReverseFrequencyValues**

Constructors

ABYSS

public **ABYSS** (*DoubleProblem* problem, int maxEvaluations, int populationSize, int referenceSet1Size, int referenceSet2Size, int archiveSize, *Archive*<*DoubleSolution*> archive, *LocalSearchOperator*<*DoubleSolution*> localSearch, *CrossoverOperator*<*DoubleSolution*> crossoverOperator, int numberOfSubRanges)

Methods

buildNewReferenceSet1

public void **buildNewReferenceSet1** ()

Build the referenceSet1 by moving the best referenceSet1Size individuals, according to a fitness comparator, from the population to the referenceSet1

buildNewReferenceSet2

public void **buildNewReferenceSet2** ()

Build the referenceSet2 by moving to it the most diverse referenceSet2Size individuals from the population in respect to the referenceSet1. The size of the referenceSet2 can be lower than referenceSet2Size depending on the current size of the population

diversificationGeneration

public *DoubleSolution* **diversificationGeneration** ()

generatePairsFromSolutionList

public List<List<*DoubleSolution*>> **generatePairsFromSolutionList** (List<*DoubleSolution*> solutionList)

Generate all pair combinations of the referenceSet1

getDescription

public String **getDescription** ()

getName

public String **getName** ()

getResult

```
public List<DoubleSolution> getResult ()
```

improvement

```
public DoubleSolution improvement (DoubleSolution solution)
```

isStoppingConditionReached

```
public boolean isStoppingConditionReached ()
```

refSet1Test

```
public boolean refSet1Test (DoubleSolution solution)
```

Tries to update the reference set one with a solution

Parameters

- **solution** – The Solution

Returns true if the Solution has been inserted, false otherwise.

refSet2Test

```
public boolean refSet2Test (DoubleSolution solution)
```

Try to update the reference set 2 with a Solution

Parameters

- **solution** – The Solution

Throws

- **JMException** –

Returns true if the Solution has been inserted, false otherwise.

referenceSetUpdate

```
public void referenceSetUpdate ()
```

Build the reference set after the initialization phase

referenceSetUpdate

```
public void referenceSetUpdate (DoubleSolution solution)
```

Update the reference set with a new solution

Parameters

- **solution** –

restart

```
public void restart ()
```

restartConditionIsFulfilled

```
public boolean restartConditionIsFulfilled (List<DoubleSolution> combinedSolutions)
```

solutionCombination

```
public List<DoubleSolution> solutionCombination (List<List<DoubleSolution>> solutionList)
```

subsetGeneration

```
public List<List<DoubleSolution>> subsetGeneration ()  
    Subset generation method
```

2.3.2 ABYSSBuilder

```
public class ABYSSBuilder implements AlgorithmBuilder<ABYSS>
```

Author Cristobal Barba

Fields

improvementOperator

```
protected LocalSearchOperator<DoubleSolution> improvementOperator
```

Constructors

ABYSSBuilder

```
public ABYSSBuilder (DoubleProblem problem, Archive<DoubleSolution> archive)
```

Methods

build

```
public ABYSS build ()
```

getArchiveSize

```
public int getArchiveSize ()
```

getCrossoverOperator

```
public CrossoverOperator<DoubleSolution> getCrossoverOperator ()
```

getImprovementOperator

```
public LocalSearchOperator<DoubleSolution> getImprovementOperator ()
```

getMaxEvaluations

```
public int getMaxEvaluations ()
```

getMutationOperator

```
public MutationOperator<DoubleSolution> getMutationOperator ()
```

getNumberOfSubranges

```
public int getNumberOfSubranges ()
```

getPopulationSize

```
public int getPopulationSize ()
```

getRefSet1Size

```
public int getRefSet1Size ()
```

getRefSet2Size

```
public int getRefSet2Size ()
```

setArchiveSize

```
public ABYSSBuilder setArchiveSize (int archiveSize)
```

setCrossoverOperator

```
public ABYSSBuilder setCrossoverOperator (CrossoverOperator<DoubleSolution> crossoverOperator)
```

setImprovementOperator

```
public ABYSSBuilder setImprovementOperator (ArchiveMutationLocalSearch<DoubleSolution>  
                                           improvementOperator)
```

setMaxEvaluations

```
public ABYSSBuilder setMaxEvaluations (int maxEvaluations)
```

setMutationOperator

```
public ABYSSBuilder setMutationOperator (MutationOperator<DoubleSolution> mutationOperator)
```

setNumberOfSubranges

```
public ABYSSBuilder setNumberOfSubranges (int numberOfSubranges)
```

setPopulationSize

```
public ABYSSBuilder setPopulationSize (int populationSize)
```

setRefSet1Size

```
public ABYSSBuilder setRefSet1Size (int refSet1Size)
```

setRefSet2Size

```
public ABYSSBuilder setRefSet2Size (int refSet2Size)
```

2.3.3 ABYSSIT

```
public class ABYSSIT  
    Created by ajnebro on 11/6/15.
```

Fields

algorithm

```
Algorithm<List<DoubleSolution>> algorithm
```

archive

```
Archive<DoubleSolution> archive
```

crossover

CrossoverOperator<DoubleSolution> **crossover**

localSearchOperator

LocalSearchOperator<DoubleSolution> **localSearchOperator**

mutation

MutationOperator<DoubleSolution> **mutation**

problem

DoubleProblem **problem**

Methods**setup**

public void **setup** ()

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

public void **shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem** ()

shouldTheHypervolumeHaveAMinimumValue

public void **shouldTheHypervolumeHaveAMinimumValue** ()

2.3.4 ABYSSTest

public class **ABYSSTest**

Created by ajnebro on 11/6/15.

Fields**archive**

Archive<DoubleSolution> **archive**

localSearch

LocalSearchOperator<DoubleSolution> **localSearch**

mutation

MutationOperator<*DoubleSolution*> **mutation**

problem

DoubleProblem **problem**

Methods

setup

public void **setup** ()

shouldInitializationPhaseLeadToAPopulationFilledWithEvaluatedSolutions

public void **shouldInitializationPhaseLeadToAPopulationFilledWithEvaluatedSolutions** ()

shouldIsStoppingConditionReachedReturnFalseIfTheConditionDoesNotFulfill

public void **shouldIsStoppingConditionReachedReturnFalseIfTheConditionDoesNotFulfill** ()

shouldIsStoppingConditionReachedReturnTrueIfTheConditionFulfills

public void **shouldIsStoppingConditionReachedReturnTrueIfTheConditionFulfills** ()

shouldReferenceSetUpdateCreateAReducedSizeReferenceSet2IfThePopulationIsNotBigEnough

public void **shouldReferenceSetUpdateCreateAReducedSizeReferenceSet2IfThePopulationIsNotBigEnough**

shouldReferenceSetUpdateCreateTheTwoRefSetsAfterBeingInvokedTheFirstTime

public void **shouldReferenceSetUpdateCreateTheTwoRefSetsAfterBeingInvokedTheFirstTime** ()

shouldRestartCreateANewPopulationWithTheRefSet1Solutions

public void **shouldRestartCreateANewPopulationWithTheRefSet1Solutions** ()

shouldSolutionCombinationProduceTheRightNumberOfSolutions

public void **shouldSolutionCombinationProduceTheRightNumberOfSolutions** ()

shouldSubsetGenerationProduceAnEmptyListIfAllTheSolutionsAreMarked

```
public void shouldSubsetGenerationProduceAnEmptyListIfAllTheSolutionsAreMarked()
```

2.4 org.uma.jmetal.algorithm.multiobjective.abyss.util**2.4.1 MarkAttribute**

```
public class MarkAttribute extends GenericSolutionAttribute<Solution<?>, Boolean>  
    Created by cbarba on 24/3/15.
```

2.5 org.uma.jmetal.algorithm.multiobjective.artificialdecisionmaker**2.5.1 ArtificialDecisionMakerIT**

```
public class ArtificialDecisionMakerIT
```

Fields**algorithm**

```
Algorithm<List<DoubleSolution>> algorithm
```

Methods**shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem()
```

2.6 org.uma.jmetal.algorithm.multiobjective.cdg**2.6.1 AbstractCDG**

```
public abstract class AbstractCDG<S extends Solution<?>> implements Algorithm<List<S>>  
    Abstract class for implementing versions of the CDG algorithm.
```

Author Feng Zhang

Fields**badPopulation**

```
protected List<S> badPopulation
```

badSolution

protected int[] **badSolution**

badSolutionNum

protected int **badSolutionNum**

border

protected List<List<S>> **border**

borderLength

protected int **borderLength**

childGridNum

protected int **childGridNum_**

childGrid

protected int **childGrid_**

crossoverOperator

protected *CrossoverOperator*<S> **crossoverOperator**

d

protected double[] **d_**

evaluations

protected int **evaluations**

gridDetalSum

protected double[][] **gridDetalSum_**

gridDetal

protected int[] **gridDetal_**

idealPoint

protected double[] **idealPoint**
Z vector in Zhang & Li paper

k

protected int **k_**

maxEvaluations

protected int **maxEvaluations**

nadirPoint

protected double[] **nadirPoint**

neighborhood

protected int[][] **neighborhood**

neighborhoodNum

protected int[] **neighborhoodNum**

neighborhoodSelectionProbability

protected double **neighborhoodSelectionProbability**
Delta in Zhang & Li paper

population

protected List<S> **population**

populationSize

protected int **populationSize**

problem

protected Problem<S> **problem**

randomGenerator

protected *JMetalRandom* **randomGenerator**

resultPopulationSize

protected int **resultPopulationSize**

sigma

protected double **sigma_**

slimDetal

protected int **slimDetal_**

spPopulationOrder

protected *List<Integer>* **spPopulationOrder**

specialPopulation

protected *List<S>* **specialPopulation**

subP

protected int[][] **subP**

subPNum

protected int[] **subPNum**

subproblem

protected *List<List<S>>* **subproblem**

subproblemNum

protected int **subproblemNum_**

t

protected int **t_**

team

protected `List<List<Integer>>` **team**

tempBorder

protected `List<List<S>>` **tempBorder**

Constructors**AbstractCDG**

public **AbstractCDG** (*Problem*<S> *problem*, int *populationSize*, int *resultPopulationSize*, int *maxEvaluations*, *CrossoverOperator*<S> *crossoverOperator*, double *neighborhoodSelectionProbability*, double *sigma_*, int *k_*, int *t_*, int *subproblemNum_*, int *childGrid_*, int *childGridNum_*)

Methods**allocateSolution**

protected void **allocateSolution** ()

chooseNeighborType

protected *NeighborType* **chooseNeighborType** (int *i*)

chooseSolution

protected void **chooseSolution** ()

chooseSpecialPopulation

protected void **chooseSpecialPopulation** ()

excludeBadSolution

protected void **excludeBadSolution** ()

excludeBadSolution3

protected void **excludeBadSolution3** ()

getBorder

protected void **getBorder** ()

getG

protected int **getG** (*S individual*, int *index*)

getGridPos

protected int **getGridPos** (int *j*, double *funValue*)

getOrder

protected int **getOrder** (*S individual*)

getPos

protected int **getPos** (int *i*, int *j*, int *k*)

getRank

protected int **getRank** (*S individual*, int *index*)

getResult

public List<S> **getResult** ()

gridSystemSetup

protected void **gridSystemSetup** ()

gridSystemSetup3

protected void **gridSystemSetup3** ()

group2

protected void **group2** ()

group3

protected void **group3** ()

individualObjRankSort

protected void **individualObjRankSort** ()

initialCDGAttributes

protected void **initialCDGAttributes** (*S individual*)

initialGridDetal

protected void **initialGridDetal** ()

initializeIdealPoint

protected void **initializeIdealPoint** ()

initializeNadirPoint

protected void **initializeNadirPoint** ()

initializeNeighborhood

protected void **initializeNeighborhood** ()
Initialize cdg neighborhoods

initializeNeighborhoodGrid

protected void **initializeNeighborhoodGrid** ()

initializeSubP2

protected void **initializeSubP2** ()

initializeSubP3

protected void **initializeSubP3** ()

isInner

protected boolean **isInner** (*S individual*)

lexicographicSort

protected void **lexicographicSort** ()

mateSelection

protected List<Integer> **mateSelection** (int *subproblemId*, int *numberOfSolutionsToSelect*, NeighborType *neighbourType*)

Parameters

- **subproblemId** – the id of current subproblem
- **neighbourType** – neighbour type

parentSelection

protected List<S> **parentSelection** (int *subProblemId*, NeighborType *neighborType*)

paretoDom

protected boolean **paretoDom** (S *individual*, int *i*)

paretoFilter

protected void **paretoFilter** ()

rankBasedSelection

protected void **rankBasedSelection** ()

setG

protected void **setG** (S *individual*, int *index*, int *value*)

setIndividualObjRank

protected void **setIndividualObjRank** ()

setOrder

protected void **setOrder** (S *individual*, int *value*)

setRank

protected void **setRank** (S *individual*, int *index*, int *value*)

setSpIndividualRank

protected void **setSpIndividualRank** ()

subproblemSort1

protected void **subproblemSort1** ()

supplyBadSolution

protected void **supplyBadSolution** ()

updateBorder

protected void **updateBorder** ()

updateIdealPoint

protected void **updateIdealPoint** (*S individual*)

updateNadirPoint

void **updateNadirPoint** ()

updateNeighborhood

protected void **updateNeighborhood** ()

2.6.2 AbstractCDG.NeighborType

protected enum **NeighborType**

Enum Constants**NEIGHBOR**

public static final *AbstractCDG.NeighborType* **NEIGHBOR**

POPULATION

public static final *AbstractCDG.NeighborType* **POPULATION**

2.6.3 CDG

public class **CDG** extends *AbstractCDG<DoubleSolution>*

Xinye Cai, Zhiwei Mei, Zhun Fan, Qingfu Zhang, A Constrained Decomposition Approach with Grids for Evolutionary Multiobjective Optimization, IEEE Transaction on Evolutionary Computation, press online, 2018, DOI: 10.1109/TEVC.2017.2744674 The paper and Matlab code can be download at <http://xinyecai.github.io/>

Author Feng Zhang

Constructors

CDG

```
public CDG (Problem<DoubleSolution> problem, int populationSize, int resultPopulationSize, int maxEvaluations, CrossoverOperator<DoubleSolution> crossover, double neighborhoodSelectionProbability, double sigma_, int k_, int t_, int subproblemNum_, int childGrid_, int childGridNum_)
```

Methods

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

initializePopulation

```
protected void initializePopulation ()
```

run

```
public void run ()
```

2.6.4 CDGBuilder

```
public class CDGBuilder implements AlgorithmBuilder<AbstractCDG<DoubleSolution>>  
    Builder class for algorithm CDG
```

Author Feng Zhang

Fields

childGridNum

```
protected int childGridNum_
```

childGrid

```
protected int childGrid_
```

crossover

```
protected CrossoverOperator<DoubleSolution> crossover
```

k

protected int **k_**

maxEvaluations

protected int **maxEvaluations**

neighborhoodSelectionProbability

protected double **neighborhoodSelectionProbability**
Delta in Zhang & Li paper

numberOfThreads

protected int **numberOfThreads**

populationSize

protected int **populationSize**

problem

protected *Problem*<*DoubleSolution*> **problem**

resultPopulationSize

protected int **resultPopulationSize**

sigma

protected double **sigma_**

subproblemNum

protected int **subproblemNum_**

t

protected int **t_**

Constructors

CDGBuilder

```
public CDGBuilder (Problem<DoubleSolution> problem)  
    Constructor
```

Methods

build

```
public AbstractCDG<DoubleSolution> build ()
```

getChildGrid

```
public int getChildGrid ()
```

getChildGridNum

```
public int getChildGridNum ()
```

getCrossover

```
public CrossoverOperator<DoubleSolution> getCrossover ()
```

getK

```
public int getK ()
```

getMaxEvaluations

```
public int getMaxEvaluations ()
```

getNeighborhoodSelectionProbability

```
public double getNeighborhoodSelectionProbability ()
```

getNumberOfThreads

```
public int getNumberOfThreads ()
```

getPopulationSize

```
public int getPopulationSize ()
```

getResultPopulationSize

```
public int getResultPopulationSize ()
```

getT

```
public double getT ()
```

setChildGrid

```
public CDGBuilder setChildGrid (int childGrid)
```

setChildGridNum

```
public CDGBuilder setChildGridNum (int childGridNum)
```

setCrossover

```
public CDGBuilder setCrossover (CrossoverOperator<DoubleSolution> crossover)
```

setK

```
public CDGBuilder setK (int k)
```

setMaxEvaluations

```
public CDGBuilder setMaxEvaluations (int maxEvaluations)
```

setNeighborhoodSelectionProbability

```
public CDGBuilder setNeighborhoodSelectionProbability (double neighborhoodSelectionProbability)
```

setNumberOfThreads

```
public CDGBuilder setNumberOfThreads (int numberOfThreads)
```

setPopulationSize

```
public CDGBuilder setPopulationSize (int populationSize)
```

setResultPopulationSize

```
public CDGBuilder setResultPopulationSize (int resultPopulationSize)
```

setT

public *CDGBuilder* **setT** (int *t*)

2.7 org.uma.jmetal.algorithm.multiobjective.cellde

2.7.1 CellDE

public class **CellDE**

Author Antonio J. Nebro

2.7.2 CellDE45

public class **CellDE45** implements *Algorithm*<*List*<*DoubleSolution*>>

Author Antonio J. Nebro

Fields

evaluations

protected int **evaluations**

maxEvaluations

protected int **maxEvaluations**

Constructors

CellDE45

public **CellDE45** (*Problem*<*DoubleSolution*> *problem*, int *maxEvaluations*, int *populationSize*, *BoundedArchive*<*DoubleSolution*> *archive*, *Neighborhood*<*DoubleSolution*> *neighborhood*, *SelectionOperator*<*List*<*DoubleSolution*>, *DoubleSolution*> *selection*, *DifferentialEvolutionCrossover* *crossover*, double *feedback*, *SolutionListEvaluator*<*DoubleSolution*> *evaluator*)

Methods

computeRanking

protected *Ranking*<*DoubleSolution*> **computeRanking** (*List*<*DoubleSolution*> *solutionList*)

createInitialPopulation

protected *List*<*DoubleSolution*> **createInitialPopulation** ()

evaluatePopulation

protected `List<DoubleSolution> evaluatePopulation (List<DoubleSolution> population)`

getDescription

public `String getDescription ()`

getName

public `String getName ()`

getResult

public `List<DoubleSolution> getResult ()`

initProgress

protected void `initProgress ()`

isStoppingConditionReached

protected boolean `isStoppingConditionReached ()`

run

public void `run ()`

updateProgress

protected void `updateProgress ()`

2.8 org.uma.jmetal.algorithm.multiobjective.dmopso

2.8.1 DMOPSO

public class **DMOPSO** implements `Algorithm<List<DoubleSolution>>`

Fields**dataDirectory**

`String dataDirectory`

functionType

FunctionType **functionType**

indArray

DoubleSolution[] **indArray**

iterations

protected int **iterations**

lambda

double[][] **lambda**

maxAge

protected int **maxAge**

maxIterations

protected int **maxIterations**

swarmSize

protected int **swarmSize**

z

double[] **z**

Constructors

DMOPSO

```
public DMOPSO (DoubleProblem problem, int swarmSize, int maxIterations, double r1Min, double r1Max, double r2Min, double r2Max, double c1Min, double c1Max, double c2Min, double c2Max, double weightMin, double weightMax, double changeVelocity1, double changeVelocity2, FunctionType functionType, String dataDirectory, int maxAge)
```


DMOPSO

public **DMOPSO** (*DoubleProblem* problem, int swarmSize, int maxIterations, double r1Min, double r1Max, double r2Min, double r2Max, double c1Min, double c1Max, double c2Min, double c2Max, double weightMin, double weightMax, double changeVelocity1, double changeVelocity2, *FunctionType* functionType, *String* dataDirectory, int maxAge, *String* name)

Methods

createInitialSwarm

protected *List*<*DoubleSolution*> **createInitialSwarm** ()

evaluateSwarm

protected *List*<*DoubleSolution*> **evaluateSwarm** (*List*<*DoubleSolution*> swarm)

getDescription

public *String* **getDescription** ()

getName

public *String* **getName** ()

getResult

public *List*<*DoubleSolution*> **getResult** ()

getSwarm

public *List*<*DoubleSolution*> **getSwarm** ()

initProgress

protected void **initProgress** ()

initializeLeaders

protected void **initializeLeaders** (*List*<*DoubleSolution*> swarm)

initializeParticlesMemory

protected void **initializeParticlesMemory** (*List*<*DoubleSolution*> swarm)

initializeVelocity

protected void **initializeVelocity** (*List<DoubleSolution> swarm*)

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

run

public void **run** ()

updateProgress

protected void **updateProgress** ()

updateVelocity

protected void **updateVelocity** (int *i*)

2.8.2 DMOPSO.FunctionType

public enum **FunctionType**

Enum Constants

AGG

public static final *DMOPSO.FunctionType* **AGG**

PBI

public static final *DMOPSO.FunctionType* **PBI**

TCHE

public static final *DMOPSO.FunctionType* **TCHE**

2.8.3 DMOPSOBuilder

public class **DMOPSOBuilder** implements *AlgorithmBuilder<DMOPSO>*

Author Jorge Rodriguez

Constructors

DMOPSOBuilder

public **DMOPSOBuilder** (*DoubleProblem* problem)

Methods

build

public *DMOPSO* **build** ()

getC1Max

public double **getC1Max** ()

getC1Min

public double **getC1Min** ()

getC2Max

public double **getC2Max** ()

getC2Min

public double **getC2Min** ()

getChangeVelocity1

public double **getChangeVelocity1** ()

getChangeVelocity2

public double **getChangeVelocity2** ()

getDataDirectory

public *String* **getDataDirectory** ()

getEvaluator

public *SolutionListEvaluator*<*DoubleSolution*> **getEvaluator** ()

getFunctionType

```
public DMOPSO.FunctionType getFunctionType ()
```

getMaxAge

```
public int getMaxAge ()
```

getMaxIterations

```
public int getMaxIterations ()
```

getName

```
public String getName ()
```

getProblem

```
public DoubleProblem getProblem ()
```

getR1Max

```
public double getR1Max ()
```

getR1Min

```
public double getR1Min ()
```

getR2Max

```
public double getR2Max ()
```

getR2Min

```
public double getR2Min ()
```

getSwarmSize

```
public int getSwarmSize ()
```

getWeightMax

```
public double getWeightMax ()
```

getWeightMin

public double **getWeightMin** ()

setC1Max

public *DMOPSOBuilder* **setC1Max** (double *c1Max*)

setC1Min

public *DMOPSOBuilder* **setC1Min** (double *c1Min*)

setC2Max

public *DMOPSOBuilder* **setC2Max** (double *c2Max*)

setC2Min

public *DMOPSOBuilder* **setC2Min** (double *c2Min*)

setChangeVelocity1

public *DMOPSOBuilder* **setChangeVelocity1** (double *changeVelocity1*)

setChangeVelocity2

public *DMOPSOBuilder* **setChangeVelocity2** (double *changeVelocity2*)

setDataDirectory

public *DMOPSOBuilder* **setDataDirectory** (*String* *dataDirectory*)

setFunctionType

public *DMOPSOBuilder* **setFunctionType** (*DMOPSO.FunctionType* *functionType*)

setMaxAge

public *DMOPSOBuilder* **setMaxAge** (int *maxAge*)

setMaxIterations

public *DMOPSOBuilder* **setMaxIterations** (int *maxIterations*)

setName

public *DMOPSOBuilder* **setName** (*String name*)

setR1Max

public *DMOPSOBuilder* **setR1Max** (*double r1Max*)

setR1Min

public *DMOPSOBuilder* **setR1Min** (*double r1Min*)

setR2Max

public *DMOPSOBuilder* **setR2Max** (*double r2Max*)

setR2Min

public *DMOPSOBuilder* **setR2Min** (*double r2Min*)

setRandomGenerator

public *DMOPSOBuilder* **setRandomGenerator** (*PseudoRandomGenerator randomGenerator*)

setSolutionListEvaluator

public *DMOPSOBuilder* **setSolutionListEvaluator** (*SolutionListEvaluator<DoubleSolution> evaluator*)

setSwarmSize

public *DMOPSOBuilder* **setSwarmSize** (*int swarmSize*)

setVariant

public *DMOPSOBuilder* **setVariant** (*DMOPSOVariant variant*)

setWeightMax

public *DMOPSOBuilder* **setWeightMax** (*double weightMax*)

setWeightMin

public *DMOPSOBuilder* **setWeightMin** (*double weightMin*)

2.8.4 DMOPSOBuilder.DMOPSOVariant

public enum **DMOPSOVariant**

Enum Constants

DMOPSO

public static final *DMOPSOBuilder.DMOPSOVariant* **DMOPSO**

Measures

public static final *DMOPSOBuilder.DMOPSOVariant* **Measures**

2.8.5 DMOPSOIT

public class **DMOPSOIT**

Integration tests for algorithm DMOPSO

Author Antonio J. Nebro

Fields

algorithm

Algorithm<List<DoubleSolution>> **algorithm**

Methods

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

public void **shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**()

shouldTheHypervolumeHaveAMinimumValue

public void **shouldTheHypervolumeHaveAMinimumValue**()

2.8.6 DMOPSOMeasures

public class **DMOPSOMeasures** extends *DMOPSO* implements *Measurable*

Fields

durationMeasure

protected *DurationMeasure* **durationMeasure**

epsilonValue

protected *BasicMeasure*<*Double*> **epsilonValue**

hypervolumeValue

protected *BasicMeasure*<*Double*> **hypervolumeValue**

iterations

protected *CountingMeasure* **iterations**

measureManager

protected *SimpleMeasureManager* **measureManager**

referenceFront

protected *Front* **referenceFront**

solutionListMeasure

protected *BasicMeasure*<*List*<*DoubleSolution*>> **solutionListMeasure**

Constructors

DMOPSOmeasures

```
public DMOPSOmeasures (DoubleProblem problem, int swarmSize, int maxIterations, double r1Min, double r1Max, double r2Min, double r2Max, double c1Min, double c1Max, double c2Min, double c2Max, double weightMin, double weightMax, double changeVelocity1, double changeVelocity2, FunctionType functionType, String dataDirectory, int maxAge)
```

DMOPSOmeasures

```
public DMOPSOmeasures (DoubleProblem problem, int swarmSize, int maxIterations, double r1Min, double r1Max, double r2Min, double r2Max, double c1Min, double c1Max, double c2Min, double c2Max, double weightMin, double weightMax, double changeVelocity1, double changeVelocity2, FunctionType functionType, String dataDirectory, int maxAge, String name)
```


Methods

getDescription

```
public String getDescription ()
```

getMeasureManager

```
public MeasureManager getMeasureManager ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

run

```
public void run ()
```

setReferenceFront

```
public void setReferenceFront (Front referenceFront)
```

updateProgress

```
protected void updateProgress ()
```

2.9 org.uma.jmetal.algorithm.multiobjective.espea

2.9.1 ESPEA

```
public class ESPEA<S extends Solution<?>> extends AbstractGeneticAlgorithm<S, List<S>>
```

Implementation of the Electrostatic Potential Energy Evolutionary Algorithm (ESPEA) from the paper “Obtaining Optimal Pareto Front Approximations using Scalarized Preference Information” by M. Braun et al.

The algorithm generates preference-biased Pareto front approximations that cover the entire front but focus more solutions in those regions that are interesting to the decision maker. Preferences are presented to the algorithm in the form of a scalarization function (value function) that maps the vector of objective to a real value. Smaller values are deemed to indicate higher desirability to comply with minimization.

If no scalarized preference is specified, uniform preferences are assumed and ESPEA generates a uniform approximation of the Pareto front.

Author Marlon Braun

Fields

archive

protected final *EnergyArchive*<S> **archive**

An archive of nondominated solutions that approximates the energy minimum state based on the chosen scalarization function.

evaluations

protected int **evaluations**

The number of function evaluations that have been executed so far.

evaluator

protected final *SolutionListEvaluator*<S> **evaluator**

Evaluates the solutions

fullArchiveCrossoverOperator

protected *CrossoverOperator*<S> **fullArchiveCrossoverOperator**

ESPEA uses two different crossover operators depending on the current archive size. If the archive is not full, it uses the crossover operator provided by *getCrossoverOperator()*. If the archive is full, *fullArchiveCrossoverOperator* is used.

maxEvaluations

protected int **maxEvaluations**

Maximum number of functions evaluations that are executed.

Constructors

ESPEA

```
public ESPEA (Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S> crossoverOperator, CrossoverOperator<S> fullArchiveCrossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, ScalarizationWrapper scalarizationWrapper, SolutionListEvaluator<S> evaluator, boolean normalizeObjectives, ReplacementStrategy replacementStrategy)
```

Constructor for setting all parameters of ESPEA.

Methods

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

reproduction

```
protected List<S> reproduction (List<S> population)
```

selection

```
protected List<S> selection (List<S> population)
```

updateProgress

```
protected void updateProgress ()
```

2.9.2 ESPEABuilder

```
public class ESPEABuilder<S extends Solution<?>> implements AlgorithmBuilder<ESPEA<S>>
```

Constructors

ESPEABuilder

```
public ESPEABuilder (Problem<S> problem, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator)
```

Methods

build

```
public ESPEA<S> build ()
```

getCrossoverOperator

```
public CrossoverOperator<S> getCrossoverOperator ()  
Returns the crossoverOperator
```

getEvaluator

```
public SolutionListEvaluator<S> getEvaluator ()  
Returns the evaluator
```

getFullArchiveCrossoverOperator

```
public CrossoverOperator<S> getFullArchiveCrossoverOperator ()  
Returns the fullArchiveCrossoverOperator
```

getMaxEvaluations

```
public int getMaxEvaluations ()  
Returns the maxEvaluations
```

getMutationOperator

```
public MutationOperator<S> getMutationOperator ()  
Returns the mutationOperator
```

getOperationType

```
public ReplacementStrategy getOperationType ()  
Returns the replacement strategy
```

getPopulationSize

public int **getPopulationSize** ()

Returns the populationSize

getScalarization

public *ScalarizationWrapper* **getScalarization** ()

Returns the scalarization

getSelectionOperator

public *SelectionOperator*<*List*<S>, S> **getSelectionOperator** ()

Returns the selectionOperator

isNormalizeObjectives

public boolean **isNormalizeObjectives** ()

Returns the normalizeObjectives

setCrossoverOperator

public void **setCrossoverOperator** (*CrossoverOperator*<S> crossoverOperator)

Parameters

- **crossoverOperator** – the crossoverOperator to set

setEvaluator

public void **setEvaluator** (*SolutionListEvaluator*<S> evaluator)

Parameters

- **evaluator** – the evaluator to set

setFullArchiveCrossoverOperator

public void **setFullArchiveCrossoverOperator** (*CrossoverOperator*<S> fullArchiveCrossoverOperator)

Parameters

- **fullArchiveCrossoverOperator** – the fullArchiveCrossoverOperator to set

setMaxEvaluations

public void **setMaxEvaluations** (int *maxEvaluations*)

Parameters

- **maxEvaluations** – the maxEvaluations to set

setMutationOperator

public void **setMutationOperator** (*MutationOperator*<S> *mutationOperator*)

Parameters

- **mutationOperator** – the mutationOperator to set

setNormalizeObjectives

public void **setNormalizeObjectives** (boolean *normalizeObjectives*)

Parameters

- **normalizeObjectives** – the normalizeObjectives to set

setPopulationSize

public void **setPopulationSize** (int *populationSize*)

Parameters

- **populationSize** – the populationSize to set

setReplacementStrategy

public void **setReplacementStrategy** (*ReplacementStrategy* *replacementStrategy*)

Parameters

- **replacementStrategy** – the replacement strategy to set

setScalarization

public void **setScalarization** (*ScalarizationWrapper* *scalarization*)

Parameters

- **scalarization** – the scalarization to set

setSelectionOperator

public void **setSelectionOperator** (*SelectionOperator*<List<S>, S> *selectionOperator*)

Parameters

- **selectionOperator** – the selectionOperator to set

2.10 org.uma.jmetal.algorithm.multiobjective.espea.util

2.10.1 AchievementScalarizationComparator

public class **AchievementScalarizationComparator**<S extends Solution<?>> implements **Comparator**<S>
 Compares solutions based on their achievement scalarization value (ASV). The ASV is always defined for a specific objective k. A solution x dominates solution y w.r.t. to their ASV, if the maximum of all objectives without k is smaller for x compared to y. If both maxima are the same, solutions are compared w.r.t. to objective k. Achievement scalarization values can be used for identifying extreme points.

Author marlon.braun

Parameters

- **<S>** – The solution type.

Constructors

AchievementScalarizationComparator

public **AchievementScalarizationComparator** (int *objective*)

The achievement scalarization comparator requires an objective for which it is defined.

Parameters

- **objective** – The objective for which achievement scalarization values are computed.

Methods

compare

public int **compare** (S *s1*, S *s2*)

2.10.2 EnergyArchive

public class **EnergyArchive**<S extends Solution<?>> extends **AbstractBoundedArchive**<S>

The archive that is used within the *ESPEA* algorithm. The archive is of variable size and bounded by the population size. A new solution can only replace an existing archive member if it leads to a reduction of the total energy of the archive.

Author marlon.braun

Constructors

EnergyArchive

public **EnergyArchive** (int *maxSize*)

Standard constructor that uses uniform preferences - all Pareto optimal solutions are equally desirable.

Parameters

- **maxSize** – Size of the final distribution of points generated by the archive.

EnergyArchive

public **EnergyArchive** (int *maxSize*, *ScalarizationWrapper* *scalWrapper*)

Constructor that requires archive size and scalarization method

Parameters

- **maxSize** – Size of the final distribution of points generated by the archive.
- **scalWrapper** – The scalarization method that is used for computing energy contributions.

EnergyArchive

public **EnergyArchive** (int *maxSize*, *ScalarizationWrapper* *scalWrapper*, boolean *normalizeObjectives*)

Constructor that requires archive size, scalarization method and whether objectives are normliazed.

Parameters

- **maxSize** – Size of the final distribution of points generated by the archive.
- **scalWrapper** – The scalarization method that is used for computing energy contributions.
- **normalizeObjectives** – Whether or not objective values are normlalized between distance computation.

EnergyArchive

public **EnergyArchive** (int *maxSize*, *ScalarizationWrapper* *scalWrapper*, boolean *normalizeObjectives*, *ReplacementStrategy* *replacementStrategy*)

Constructor that requires archive size, scalarization method, whether objectives are normalized and the replacement strategy.

Parameters

- **maxSize** – Size of the final distribution of points generated by the archive.
- **scalWrapper** – The scalarization method that is used for computing energy contributions.
- **normalizeObjectives** – Whether or not objective values are normlalized between distance computation.
- **replacementStrategy** – Replacement strategy for archive update.

Methods

computeDensityEstimator

public void **computeDensityEstimator** ()

getComparator

public *Comparator*<*S*> **getComparator** ()

isFull

```
public boolean isFull ()
```

A check for testing whether the archive is full.

Returns true if the archive possesses the maximum number of elements. False otherwise.

prune

```
public void prune ()
```

sortByDensityEstimator

```
public void sortByDensityEstimator ()
```

2.10.3 EnergyArchive.ReplacementStrategy

```
public static enum ReplacementStrategy
```

The replacement strategy defines the rule by which an existing archive member is replaced by a new solution. Computational studies have revealed that *BEST_FEASIBLE_POSITION* is inferior to *LARGEST_DIFFERENCE* and *WORST_IN_ARCHIVE*. No significant performance difference could be found between *LARGEST_DIFFERENCE* and *WORST_IN_ARCHIVE*. See “Obtaining Optimal Pareto Front Approximations” by Braun et al. and “Scalarized Preferences in Multi-objective Optimizaiton” by Braun for details.

Author marlon.braun

Enum Constants**BEST_FEASIBLE_POSITION**

```
public static final EnergyArchive.ReplacementStrategy BEST_FEASIBLE_POSITION
```

Inserts the new solution such that the energy it introduces into the archive is minimized.

LARGEST_DIFFERENCE

```
public static final EnergyArchive.ReplacementStrategy LARGEST_DIFFERENCE
```

Maximizes the energy differences before and after replacement.

WORST_IN_ARCHIVE

```
public static final EnergyArchive.ReplacementStrategy WORST_IN_ARCHIVE
```

Among all eligible archive members that can be replaced the one exhibiting the largest energy contribution is replaced.

2.10.4 ScalarizationUtils

public class **ScalarizationUtils**

A class that contains methods for computing the scalarization values of solutions. A scalarization value is an aggregation of the objective values that maps to the real numbers, e.g. the weighted sum.

Scalarization values are stored as *ScalarizationValue* in the solutions.

Author Marlon Braun

Methods

angleUtility

public static <S extends Solution<?>> void **angleUtility** (*List<S> solutionsList*)

Scalarization values based on angle utility (see Angle-based Preference Models in Multi-objective Optimization by Braun et al.). Extreme points are computed from the list of solutions.

Parameters

- **solutionsList** – A list of solutions.

angleUtility

public static <S extends Solution<?>> void **angleUtility** (*List<S> solutionsList*, *double[][] extremePoints*)

Scalarization values based on angle utility (see Angle-based Preference Models in Multi-objective Optimization by Braun et al.).

Parameters

- **solutionsList** – A list of solutions.
- **extremePoints** – used for angle computation.

chebyshev

public static <S extends Solution<?>> void **chebyshev** (*List<S> solutionsList*)

Scalarization values based on the Chebyshev function. The ideal point is computed from the list of solutions.

Parameters

- **solutionsList** – A list of solutions.

chebyshev

public static <S extends Solution<?>> void **chebyshev** (*List<S> solutionsList*, *double[] idealValues*)

Scalarization values based on the Chebyshev function.

Parameters

- **solutionsList** – A list of solutions.
- **idealValues** – The ideal point

nash

public static <S extends Solution<?>> void **nash** (List<S> *solutionsList*)

Scalarization values based on the Nash bargaining solution. The disagreement point is computed based on the list of solutions.

Parameters

- **solutionsList** – A list of solutions.

nash

public static <S extends Solution<?>> void **nash** (List<S> *solutionsList*, double[] *nadirValues*)

Scalarization values based on the Nash bargaining solution.

Parameters

- **solutionsList** – A list of solutions.
- **nadirValues** – The disagreement point.

productOfObjectives

public static <S extends Solution<?>> void **productOfObjectives** (List<S> *solutionsList*)

Objective values are multiplied.

Parameters

- **solutionsList** – A list of solutions.

sumOfObjectives

public static <S extends Solution<?>> void **sumOfObjectives** (List<S> *solutionsList*)

Scalarization values is computed by summing objective values.

Parameters

- **solutionsList** – A list of solutions.

tradeoffUtility

public static <S extends Solution<?>> void **tradeoffUtility** (List<S> *solutionsList*)

Scalarization values based on tradeoff utility, also known as proper utility (see “Theory and Algorithm for Finding Knees” by Shukla et al.)

Parameters

- **solutionsList** – A list of solutions.

uniform

public static <S extends Solution<?>> void **uniform** (List<S> *solutionsList*)

Uniform preferences. Each solution is assigned a scalarization value of 1.0.

Parameters

- **solutionsList** – A list of solutions.

weightedChebyshev

public static <S extends Solution<?>> void **weightedChebyshev** ([List](#)<S> *solutionsList*, double[] *weights*)

Chebyshev function with weighted objectives.

Parameters

- **solutionsList** – A list of solutions.
- **weights** – Constants by which ideal values and objective values are multiplied.

weightedChebyshev

public static <S extends Solution<?>> void **weightedChebyshev** ([List](#)<S> *solutionsList*, double[] *idealValues*, double[] *weights*)

Scalarization values based on the weighted Chebyshev function.

Parameters

- **solutionsList** – A list of solutions.
- **idealValues** – The ideal point.
- **weights** – Constants by which ideal values and objective values are multiplied.

weightedProduct

public static <S extends Solution<?>> void **weightedProduct** ([List](#)<S> *solutionsList*, double[] *weights*)

Objectives are exponentiated by a positive weight and afterwards multiplied.

Parameters

- **solutionsList** – A list of solutions.
- **weights** – Weights by objectives are exponentiated

weightedSum

public static <S extends Solution<?>> void **weightedSum** ([List](#)<S> *solutionsList*, double[] *weights*)

Objective values are multiplied by weights and summed. Weights should always be positive.

Parameters

- **solutionsList** – A list of solutions.
- **weights** – Positive constants by which objectives are summed.

2.10.5 ScalarizationValue

public class **ScalarizationValue**<S extends Solution<?>> extends [GenericSolutionAttribute](#)<S, Double>
Scalarization attribute. A scalarization value is an aggregation of the objective values.

Author Marlon Braun

Parameters

- **<S>** – The solution type

2.10.6 ScalarizationWrapper

public class **ScalarizationWrapper**

A class for simplifying the access to *ScalarizationUtils*.

Author Marlon Braun

Constructors**ScalarizationWrapper**

public **ScalarizationWrapper** (*ScalarizationType* scalarizationType)

Initialize from scalarization type

Parameters

- **scalarizationType** – Chosen scalarization function

ScalarizationWrapper

public **ScalarizationWrapper** (*Config* config)

Initialize from Config.

Parameters

- **config** – Configuration of the scalarization Wrapper.

Methods**execute**

public <S extends Solution<?>> void **execute** (*List<S>* solutionsList)

Computes scalarization values and assigns them as *ScalarizationValue* attribute to the solutions.

Parameters

- **solutionsList** – Solutions for which scalarization values computed.

2.10.7 ScalarizationWrapper.Config

public static class **Config**

Configuration of the scalarization wrapper.

Author Marlon Braun

2.10.8 `ScalarizationWrapper.ScalarizationType`

public static enum **ScalarizationType**

The scalarization function that is used for computing values.

Author Marlon Braun

Enum Constants

ANGLE_UTILITY

public static final *ScalarizationWrapper.ScalarizationType* **ANGLE_UTILITY**

Scalarization values are based on maximum angles to extreme points (see “Angle based Preferences Models in Multi-objective Optimization” by Braun et al.)

CHEBYSHEV

public static final *ScalarizationWrapper.ScalarizationType* **CHEBYSHEV**

Chebyshev scalarization function.

NASH

public static final *ScalarizationWrapper.ScalarizationType* **NASH**

The Nash bargaining solution

PRODUCT_OF_OBJECTIVES

public static final *ScalarizationWrapper.ScalarizationType* **PRODUCT_OF_OBJECTIVES**

Multiplication of all objectives.

SUM_OF_OBJECTIVES

public static final *ScalarizationWrapper.ScalarizationType* **SUM_OF_OBJECTIVES**

Summing up all objectives.

TRADEOFF_UTILITY

public static final *ScalarizationWrapper.ScalarizationType* **TRADEOFF_UTILITY**

Tradeoff utility also known as proper utility (see “Theory and Algorithms for Finding Knees” by Shukla et al.).

UNIFORM

public static final *ScalarizationWrapper.ScalarizationType* **UNIFORM**

All solutions are assigned a scalarization value of 1.

WEIGHTED_CHEBYSHEV

public static final *ScalarizationWrapper.ScalarizationType* **WEIGHTED_CHEBYSHEV**
 Chebyhsev function with weights.

WEIGHTED_PRODUCT

public static final *ScalarizationWrapper.ScalarizationType* **WEIGHTED_PRODUCT**
 Objectives are exponentiated by weights before being multiplied.

WEIGHTED_SUM

public static final *ScalarizationWrapper.ScalarizationType* **WEIGHTED_SUM**
 Weighted sum.

2.11 org.uma.jmetal.algorithm.multiobjective.gde3

2.11.1 GDE3

public class **GDE3** extends *AbstractDifferentialEvolution<List<DoubleSolution>>*
 This class implements the GDE3 algorithm

Fields

crowdingDistance

protected *DensityEstimator<DoubleSolution>* **crowdingDistance**

dominanceComparator

protected *Comparator<DoubleSolution>* **dominanceComparator**

evaluations

protected int **evaluations**

evaluator

protected *SolutionListEvaluator<DoubleSolution>* **evaluator**

maxEvaluations

protected int **maxEvaluations**

ranking

protected *Ranking*<*DoubleSolution*> **ranking**

Constructors

GDE3

public **GDE3** (*DoubleProblem* problem, int populationSize, int maxEvaluations, *DifferentialEvolutionSelection* selection, *DifferentialEvolutionCrossover* crossover, *SolutionListEvaluator*<*DoubleSolution*> evaluator)
Constructor

Methods

addLastRankedSolutionsToPopulation

protected void **addLastRankedSolutionsToPopulation** (*Ranking*<*DoubleSolution*> ranking, int rank, List<*DoubleSolution*> population)

addRankedSolutionsToPopulation

protected void **addRankedSolutionsToPopulation** (*Ranking*<*DoubleSolution*> ranking, int rank, List<*DoubleSolution*> population)

computeRanking

protected *Ranking*<*DoubleSolution*> **computeRanking** (List<*DoubleSolution*> solutionList)

createInitialPopulation

protected List<*DoubleSolution*> **createInitialPopulation** ()

crowdingDistanceSelection

protected List<*DoubleSolution*> **crowdingDistanceSelection** (*Ranking*<*DoubleSolution*> ranking)

evaluatePopulation

protected List<*DoubleSolution*> **evaluatePopulation** (List<*DoubleSolution*> population)
Evaluate population method

Parameters

- **population** – The list of solutions to be evaluated

Returns A list of evaluated solutions

getDescription

```
public String getDescription ()
```

getMaxPopulationSize

```
public int getMaxPopulationSize ()
```

getName

```
public String getName ()
```

getNonDominatedSolutions

```
protected List<DoubleSolution> getNonDominatedSolutions (List<DoubleSolution> solutionList)
```

getResult

```
public List<DoubleSolution> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

populationIsNotFull

```
protected boolean populationIsNotFull (List<DoubleSolution> population)
```

replacement

```
protected List<DoubleSolution> replacement (List<DoubleSolution> population, List<DoubleSolution>  
offspringPopulation)
```

reproduction

```
protected List<DoubleSolution> reproduction (List<DoubleSolution> matingPopulation)
```

selection

```
protected List<DoubleSolution> selection (List<DoubleSolution> population)
```

setMaxPopulationSize

public void **setMaxPopulationSize** (int *maxPopulationSize*)

subfrontFillsIntoThePopulation

protected boolean **subfrontFillsIntoThePopulation** (*Ranking<DoubleSolution>* ranking, int rank,
List<*DoubleSolution*> population)

updateProgress

protected void **updateProgress** ()

2.11.2 GDE3Builder

public class **GDE3Builder** implements *AlgorithmBuilder<GDE3>*
This class implements the GDE3 algorithm

Fields

crossoverOperator

protected *DifferentialEvolutionCrossover* **crossoverOperator**

evaluator

protected *SolutionListEvaluator<DoubleSolution>* **evaluator**

maxEvaluations

protected int **maxEvaluations**

populationSize

protected int **populationSize**

selectionOperator

protected *DifferentialEvolutionSelection* **selectionOperator**

Constructors

GDE3Builder

public **GDE3Builder** (*DoubleProblem* problem)
Constructor

Methods

build

public *GDE3* **build** ()

getCrossoverOperator

public *CrossoverOperator*<*DoubleSolution*> **getCrossoverOperator** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getPopulationSize

public int **getPopulationSize** ()

getSelectionOperator

public *SelectionOperator*<*List*<*DoubleSolution*>, *List*<*DoubleSolution*>> **getSelectionOperator** ()

setCrossover

public *GDE3Builder* **setCrossover** (*DifferentialEvolutionCrossover* crossover)

setMaxEvaluations

public *GDE3Builder* **setMaxEvaluations** (int *maxEvaluations*)

setPopulationSize

public *GDE3Builder* **setPopulationSize** (int *populationSize*)

setSelection

public *GDE3Builder* **setSelection** (*DifferentialEvolutionSelection* selection)

setSolutionSetEvaluator

```
public GDE3Builder setSolutionSetEvaluator (SolutionListEvaluator<DoubleSolution> evaluator)
```

2.11.3 GDE3TestIT

```
public class GDE3TestIT  
    Created by ajnebro on 3/11/15.
```

Fields

algorithm

```
Algorithm<List<DoubleSolution>> algorithm
```

Methods

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem ()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue ()
```

2.12 org.uma.jmetal.algorithm.multiobjective.gwasfga

2.12.1 GWASFGA

```
public class GWASFGA<S extends Solution<?>> extends WASFGA<S>
```

This class executes the GWASFGA algorithm described in: Saborido, R., Ruiz, A. B. and Luque, M. (2015). Global WASF-GA: An Evolutionary Algorithm in Multiobjective Optimization to Approximate the whole Pareto Optimal Front. Evolutionary Computation Accepted for publication.

Author Juanjo Durillo

Constructors

GWASFGA

```
public GWASFGA (Problem<S> problem, int populationSize, int maxIterations, CrossoverOperator<S>  
    crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>,  
    S> selectionOperator, SolutionListEvaluator<S> evaluator, double epsilon, String  
    weightVectorsFileName)
```

GWASF GA

```
public GWASF GA (Problem<S> problem, int populationSize, int maxIterations, CrossoverOperator<S>
crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>,
S> selectionOperator, SolutionListEvaluator<S> evaluator, double epsilon)
```

Methods

computeRanking

```
protected Ranking<S> computeRanking (List<S> solutionList)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

2.13 org.uma.jmetal.algorithm.multiobjective.gwasfga.util

2.13.1 GWASF GARanking

```
public class GWASF GARanking<S extends Solution<?>> extends GenericSolutionAttribute<S, Integer> implements Ranking<S>
```

Author Rubén Saborido Implementation of the ranking procedure for the algorithm GWASF-GA on jMetal5.0 It classifies solutions into different fronts. If the problem contains constraints, after feasible solutions it classifies the unfeasible solutions into fronts: - Each unfeasible solution goes into a different front. - Unfeasible solutions with lower number of violated constraints are preferred. - If two solutions have equal number of violated constraints it compares the overall constraint values. - If two solutions have equal overall constraint values it compares de values of the utility function.

Constructors

GWASF GARanking

```
public GWASF GARanking (AbstractUtilityFunctionsSet<S> utilityFunctionsUtopia, AbstractUtilityFunction-
sSet<S> utilityFunctionsNadir)
```

Methods

computeRanking

```
public Ranking<S> computeRanking (List<S> population)
```

getNumberOfSubfronts

```
public int getNumberOfSubfronts ()
```

getSubfront

```
public List<S> getSubfront (int rank)
```

rankUnfeasibleSolutions

```
protected int[] rankUnfeasibleSolutions (List<S> population)
```

Obtain the rank of each solution in a list of unfeasible solutions

Parameters

- **population** – List of unfeasible solutions

Returns The rank of each unfeasible solutions

2.14 org.uma.jmetal.algorithm.multiobjective.ibea

2.14.1 IBEA

```
public class IBEA<S extends Solution<?>> implements Algorithm<List<S>>
```

This class implements the IBEA algorithm

Fields**TOURNAMENTS_ROUNDS**

```
public static final int TOURNAMENTS_ROUNDS
```

archive

```
protected List<S> archive
```

archiveSize

```
protected int archiveSize
```

crossoverOperator

```
protected CrossoverOperator<S> crossoverOperator
```

indicatorValues

protected `List<List<Double>>` **indicatorValues**

maxEvaluations

protected int **maxEvaluations**

maxIndicatorValue

protected double **maxIndicatorValue**

mutationOperator

protected *MutationOperator*<S> **mutationOperator**

populationSize

protected int **populationSize**

problem

protected *Problem*<S> **problem**

selectionOperator

protected *SelectionOperator*<List<S>, S> **selectionOperator**

solutionFitness

protected *Fitness*<S> **solutionFitness**

Constructors**IBEA**

```
public IBEA (Problem<S> problem, int populationSize, int archiveSize, int maxEvaluations, SelectionOperator<List<S>, S> selectionOperator, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator)
    Constructor
```

Methods

calculateFitness

public void **calculateFitness** (*List<S>* *solutionSet*)
Calculate the fitness for the entire population.

calculateHypervolumeIndicator

double **calculateHypervolumeIndicator** (*Solution<?>* *solutionA*, *Solution<?>* *solutionB*, int *d*, double[] *maximumValues*, double[] *minimumValues*)
Calculates the hypervolume of that portion of the objective space that is dominated by individual a but not by individual b

computeIndicatorValuesHD

public void **computeIndicatorValuesHD** (*List<S>* *solutionSet*, double[] *maximumValues*, double[] *minimumValues*)
This structure stores the indicator values of each pair of elements

fitness

public void **fitness** (*List<S>* *solutionSet*, int *pos*)
Calculate the fitness for the individual at position pos

getDescription

public *String* **getDescription** ()

getName

public *String* **getName** ()

getResult

public *List<S>* **getResult** ()

removeWorst

public void **removeWorst** (*List<S>* *solutionSet*)
Update the fitness before removing an individual

run

public void **run** ()
Execute() method

2.14.2 IBEABuilder

public class **IBEABuilder** implements *AlgorithmBuilder<IBEA<DoubleSolution>>*
 This class implements the IBEA algorithm

Constructors

IBEABuilder

public **IBEABuilder** (*Problem<DoubleSolution>* problem)
 Constructor

Parameters

- **problem** –

Methods

build

public *IBEA<DoubleSolution>* **build** ()

getArchiveSize

public int **getArchiveSize** ()

getCrossover

public *CrossoverOperator<DoubleSolution>* **getCrossover** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getMutation

public *MutationOperator<DoubleSolution>* **getMutation** ()

getPopulationSize

public int **getPopulationSize** ()

getSelection

public *SelectionOperator<List<DoubleSolution>, DoubleSolution>* **getSelection** ()

setArchiveSize

public *IBEABuilder* **setArchiveSize** (int *archiveSize*)

setCrossover

public *IBEABuilder* **setCrossover** (*CrossoverOperator*<*DoubleSolution*> *crossover*)

setMaxEvaluations

public *IBEABuilder* **setMaxEvaluations** (int *maxEvaluations*)

setMutation

public *IBEABuilder* **setMutation** (*MutationOperator*<*DoubleSolution*> *mutation*)

setPopulationSize

public *IBEABuilder* **setPopulationSize** (int *populationSize*)

setSelection

public *IBEABuilder* **setSelection** (*SelectionOperator*<*List*<*DoubleSolution*>, *DoubleSolution*> *selection*)

2.15 org.uma.jmetal.algorithm.multiobjective.mocell

2.15.1 MOCell

public class **MOCell**<*S* extends *Solution*<?>> extends *AbstractGeneticAlgorithm*<*S*, *List*<*S*>>

Author JuanJo Durillo

Parameters

- <*S*> –

Fields

archive

protected *BoundedArchive*<*S*> **archive**

currentIndividual

protected int **currentIndividual**

currentNeighbors

protected `List<S>` **currentNeighbors**

dominanceComparator

protected `Comparator<S>` **dominanceComparator**

evaluations

protected int **evaluations**

evaluator

protected final `SolutionListEvaluator<S>` **evaluator**

location

protected `LocationAttribute<S>` **location**

maxEvaluations

protected int **maxEvaluations**

neighborhood

protected `Neighborhood<S>` **neighborhood**

Constructors**MOCeII**

```
public MOCeII (Problem<S> problem, int maxEvaluations, int populationSize, BoundedArchive<S> archive,
               Neighborhood<S> neighborhood, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator,
               SelectionOperator<List<S>, S> selectionOperator, SolutionListEvaluator<S> evaluator)
```

Constructor

Parameters

- **problem** –
- **maxEvaluations** –
- **populationSize** –
- **neighborhood** –
- **crossoverOperator** –

- `mutationOperator` –
- `selectionOperator` –
- `evaluator` –

Methods

`createInitialPopulation`

protected `List<S>` `createInitialPopulation` ()

`evaluatePopulation`

protected `List<S>` `evaluatePopulation` (`List<S>` *population*)

`getDescription`

public `String` `getDescription` ()

`getName`

public `String` `getName` ()

`getResult`

public `List<S>` `getResult` ()

`initProgress`

protected void `initProgress` ()

`isStoppingConditionReached`

protected boolean `isStoppingConditionReached` ()

`replacement`

protected `List<S>` `replacement` (`List<S>` *population*, `List<S>` *offspringPopulation*)

`reproduction`

protected `List<S>` `reproduction` (`List<S>` *population*)

selection

protected `List<S> selection (List<S> population)`

updateProgress

protected void `updateProgress ()`

2.15.2 MOCe11Builder

public class **MOCe11Builder**<S extends Solution<?>> implements *AlgorithmBuilder*<*MOCe11*<S>>
Created by juanjo

Fields**archive**

protected *BoundedArchive*<S> **archive**

crossoverOperator

protected *CrossoverOperator*<S> **crossoverOperator**

evaluator

protected *SolutionListEvaluator*<S> **evaluator**

maxEvaluations

protected int **maxEvaluations**

mutationOperator

protected *MutationOperator*<S> **mutationOperator**

neighborhood

protected *Neighborhood*<S> **neighborhood**

populationSize

protected int **populationSize**

problem

protected final *Problem*<S> **problem**
MOCeIIBuilder class

selectionOperator

protected *SelectionOperator*<List<S>, S> **selectionOperator**

Constructors

MOCeIIBuilder

public **MOCeIIBuilder** (*Problem*<S> *problem*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*)
MOCeIIBuilder constructor

Methods

build

public *MOCeII*<S> **build** ()

getArchive

public *BoundedArchive*<S> **getArchive** ()

getCrossoverOperator

public *CrossoverOperator*<S> **getCrossoverOperator** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

getPopulationSize

public int **getPopulationSize** ()

getProblem

```
public Problem<S> getProblem ()
```

getSelectionOperator

```
public SelectionOperator<List<S>, S> getSelectionOperator ()
```

getSolutionListEvaluator

```
public SolutionListEvaluator<S> getSolutionListEvaluator ()
```

setArchive

```
public MOCellBuilder<S> setArchive (BoundedArchive<S> archive)
```

setMaxEvaluations

```
public MOCellBuilder<S> setMaxEvaluations (int maxEvaluations)
```

setNeighborhood

```
public MOCellBuilder<S> setNeighborhood (Neighborhood<S> neighborhood)
```

setPopulationSize

```
public MOCellBuilder<S> setPopulationSize (int populationSize)
```

setSelectionOperator

```
public MOCellBuilder<S> setSelectionOperator (SelectionOperator<List<S>, S> selectionOperator)
```

setSolutionListEvaluator

```
public MOCellBuilder<S> setSolutionListEvaluator (SolutionListEvaluator<S> evaluator)
```

2.15.3 MOCellBuilder.MOCellVariant

```
public enum MOCellVariant
```

Enum Constants

MOCeII

public static final *MOCeIIBuilder.MOCeIIVariant* **MOCeII**

Measures

public static final *MOCeIIBuilder.MOCeIIVariant* **Measures**

SteadyStateMOCeII

public static final *MOCeIIBuilder.MOCeIIVariant* **SteadyStateMOCeII**

2.15.4 MOCeIIIT

public class **MOCeIIIT**

Fields

algorithm

Algorithm<List<DoubleSolution>> **algorithm**

crossover

CrossoverOperator<DoubleSolution> **crossover**

mutation

MutationOperator<DoubleSolution> **mutation**

problem

DoubleProblem **problem**

Methods

setup

public void **setup** ()

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

public void **shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem** ()

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue ()
```

2.16 org.uma.jmetal.algorithm.multiobjective.mochc**2.16.1 MOCHC**

```
public class MOCHC extends AbstractEvolutionaryAlgorithm<BinarySolution, List<BinarySolution>>
```

This class executes the MOCHC algorithm described in: A.J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, J.J. Durillo “Optimal antenna placement using a new multi-objective chc algorithm”. GECCO ‘07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England. July 2007.

Constructors**MOCHC**

```
public MOCHC (BinaryProblem problem, int populationSize, int maxEvaluations, int convergenceValue, double  
    preservedPopulation, double initialConvergenceCount, CrossoverOperator<BinarySolution>  
    crossoverOperator, MutationOperator<BinarySolution> cataclysmicMutation, Selection-  
    Operator<List<BinarySolution>, List<BinarySolution>> newGenerationSelection, Sele-  
    tionOperator<List<BinarySolution>, BinarySolution> parentSelection, SolutionListEvalua-  
    tor<BinarySolution> evaluator)
```

Constructor

Methods**createInitialPopulation**

```
protected List<BinarySolution> createInitialPopulation ()
```

evaluatePopulation

```
protected List<BinarySolution> evaluatePopulation (List<BinarySolution> population)
```

getDescription

```
public String getDescription ()
```

getMaxPopulationSize

```
public int getMaxPopulationSize ()
```

getName

```
public String getName ()
```

getResult

```
public List<BinarySolution> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<BinarySolution> replacement (List<BinarySolution> population, List<BinarySolution> off-  
springPopulation)
```

reproduction

```
protected List<BinarySolution> reproduction (List<BinarySolution> matingPopulation)
```

selection

```
protected List<BinarySolution> selection (List<BinarySolution> population)
```

setMaxPopulationSize

```
public void setMaxPopulationSize (int maxPopulationSize)
```

updateProgress

```
protected void updateProgress ()
```

2.16.2 MOCHC45

```
public class MOCHC45 implements Algorithm<List<BinarySolution>>
```

This class executes the MOCHC algorithm described in: A.J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, J.J. Durillo “Optimal antenna placement using a new multi-objective chc algorithm”. GECCO ‘07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England. July 2007. Implementation of MOCHC following the scheme used in jMetal4.5 and former versions, i.e, without implementing the *AbstractGeneticAlgorithm* interface.

Constructors

MOCHC45

```
public MOCHC45 (BinaryProblem problem, int populationSize, int maxEvaluations, int convergence-
    Value, double preservedPopulation, double initialConvergenceCount, CrossoverOperator<BinarySolution> crossoverOperator, MutationOperator<BinarySolution> cataclysmic-
    Mutation, SelectionOperator<List<BinarySolution>, List<BinarySolution>> newGeneration-
    Selection, SelectionOperator<List<BinarySolution>, BinarySolution> parentSelection,
    SolutionListEvaluator<BinarySolution> evaluator)
```

Constructor

Methods

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public List<BinarySolution> getResult ()
```

run

```
public void run ()
```

2.16.3 MOCHCBuilder

```
public class MOCHCBuilder implements AlgorithmBuilder<MOCHC>
    Builder class
```

Fields

cataclysmicMutation

```
MutationOperator<BinarySolution> cataclysmicMutation
```

convergenceValue

```
int convergenceValue
```

crossoverOperator

CrossoverOperator<BinarySolution> **crossoverOperator**

evaluator

SolutionListEvaluator<BinarySolution> **evaluator**

initialConvergenceCount

double **initialConvergenceCount**

maxEvaluations

int **maxEvaluations**

newGenerationSelection

SelectionOperator<List<BinarySolution>, List<BinarySolution>> **newGenerationSelection**

parentSelection

SelectionOperator<List<BinarySolution>, BinarySolution> **parentSelection**

populationSize

int **populationSize**

preservedPopulation

double **preservedPopulation**

problem

BinaryProblem **problem**

Constructors

MOCHCBuilder

public **MOCHCBuilder** (*BinaryProblem problem*)

Methods

build

public *MOCHC* **build** ()

getCataclysmicMutation

public *MutationOperator*<*BinarySolution*> **getCataclysmicMutation** ()

getConvergenceValue

public int **getConvergenceValue** ()

getCrossover

public *CrossoverOperator*<*BinarySolution*> **getCrossover** ()

getInitialConvergenceCount

public double **getInitialConvergenceCount** ()

getMaxEvaluation

public int **getMaxEvaluation** ()

getNewGenerationSelection

public *SelectionOperator*<*List*<*BinarySolution*>, *List*<*BinarySolution*>> **getNewGenerationSelection** ()

getParentSelection

public *SelectionOperator*<*List*<*BinarySolution*>, *BinarySolution*> **getParentSelection** ()

getPopulationSize

public int **getPopulationSize** ()

getPreservedPopulation

public double **getPreservedPopulation** ()

getProblem

public *BinaryProblem* **getProblem**()

setCataclysmicMutation

public *MOHCBuilder* **setCataclysmicMutation** (*MutationOperator*<*BinarySolution*> *cataclysmicMutation*)

setConvergenceValue

public *MOHCBuilder* **setConvergenceValue** (int *convergenceValue*)

setCrossover

public *MOHCBuilder* **setCrossover** (*CrossoverOperator*<*BinarySolution*> *crossover*)

setEvaluator

public *MOHCBuilder* **setEvaluator** (*SolutionListEvaluator*<*BinarySolution*> *evaluator*)

setInitialConvergenceCount

public *MOHCBuilder* **setInitialConvergenceCount** (double *initialConvergenceCount*)

setMaxEvaluations

public *MOHCBuilder* **setMaxEvaluations** (int *maxEvaluations*)

setNewGenerationSelection

public *MOHCBuilder* **setNewGenerationSelection** (*SelectionOperator*<*List*<*BinarySolution*>, *List*<*BinarySolution*>> *newGenerationSelection*)

setParentSelection

public *MOHCBuilder* **setParentSelection** (*SelectionOperator*<*List*<*BinarySolution*>, *BinarySolution*> *parentSelection*)

setPopulationSize

public *MOHCBuilder* **setPopulationSize** (int *populationSize*)

setPreservedPopulation

public *MOCHCBuilder* **setPreservedPopulation** (double *preservedPopulation*)

2.17 org.uma.jmetal.algorithm.multiobjective.moead**2.17.1 AbstractMOEAD**

public abstract class **AbstractMOEAD**<S extends Solution<?>> implements *Algorithm*<List<S>>
 Abstract class for implementing versions of the MOEA/D algorithm.

Author Antonio J. Nebro

Fields**crossoverOperator**

protected *CrossoverOperator*<S> **crossoverOperator**

dataDirectory

protected String **dataDirectory**

evaluations

protected int **evaluations**

functionType

protected *FunctionType* **functionType**

idealPoint

protected *IdealPoint* **idealPoint**
 Z vector in Zhang & Li paper

indArray

protected *Solution*<?>[] **indArray**

jointPopulation

protected List<S> **jointPopulation**

lambda

protected double[][] **lambda**
Lambda vectors

maxEvaluations

protected int **maxEvaluations**

maximumNumberOfReplacedSolutions

protected int **maximumNumberOfReplacedSolutions**
nr in Zhang & Li paper

mutationOperator

protected *MutationOperator*<S> **mutationOperator**

nadirPoint

protected *NadirPoint* **nadirPoint**

neighborSize

protected int **neighborSize**
T in Zhang & Li paper

neighborhood

protected int[][] **neighborhood**

neighborhoodSelectionProbability

protected double **neighborhoodSelectionProbability**
Delta in Zhang & Li paper

offspringPopulation

protected *List*<S> **offspringPopulation**

population

protected *List*<S> **population**

populationSize

protected int **populationSize**

problem

protected *Problem*<S> **problem**

randomGenerator

protected *JMetalRandom* **randomGenerator**

resultPopulationSize

protected int **resultPopulationSize**

Constructors**AbstractMOEAD**

public **AbstractMOEAD** (*Problem*<S> *problem*, int *populationSize*, int *resultPopulationSize*, int *maxEvaluations*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutation*, *FunctionType* *functionType*, *String* *dataDirectory*, double *neighborhoodSelectionProbability*, int *maximumNumberOfReplacedSolutions*, int *neighborSize*)

Methods**chooseNeighborType**

protected *NeighborType* **chooseNeighborType** ()

fitnessFunction

double **fitnessFunction** (S *individual*, double[] *lambda*)

getResult

public *List*<S> **getResult** ()

initializeNeighborhood

protected void **initializeNeighborhood** ()
Initialize neighborhoods

initializeUniformWeight

protected void **initializeUniformWeight** ()
Initialize weight vectors

matingSelection

protected [List<Integer>](#) **matingSelection** (int *subproblemId*, int *numberOfSolutionsToSelect*, *NeighborType* *neighbourType*)

Parameters

- **subproblemId** – the id of current subproblem
- **neighbourType** – neighbour type

parentSelection

protected [List<S>](#) **parentSelection** (int *subProblemId*, *NeighborType* *neighborType*)

updateNeighborhood

protected void **updateNeighborhood** (S *individual*, int *subProblemId*, *NeighborType* *neighborType*)
Update neighborhood method

Parameters

- **individual** –
- **subProblemId** –
- **neighborType** –

Throws

- *[JMetalException](#)* –

2.17.2 AbstractMOEAD.FunctionType

public enum **FunctionType**

Enum Constants

AGG

public static final *AbstractMOEAD.FunctionType* **AGG**

PBI

public static final *AbstractMOEAD.FunctionType* **PBI**

TCHE

```
public static final AbstractMOEAD.FunctionType TCHE
```

2.17.3 AbstractMOEAD.NeighborType

```
protected enum NeighborType
```

Enum Constants

NEIGHBOR

```
public static final AbstractMOEAD.NeighborType NEIGHBOR
```

POPULATION

```
public static final AbstractMOEAD.NeighborType POPULATION
```

2.17.4 ConstraintMOEAD

```
public class ConstraintMOEAD extends AbstractMOEAD<DoubleSolution>
```

This class implements a constrained version of the MOEAD algorithm based on the one presented in the paper:
“An adaptive constraint handling approach embedded MOEA/D”. DOI: 10.1109/CEC.2012.6252868

Author Antonio J. Nebro, Juan J. Durillo

Constructors

ConstraintMOEAD

```
public ConstraintMOEAD (Problem<DoubleSolution> problem, int populationSize, int resultPopulationSize, int maxEvaluations, MutationOperator<DoubleSolution> mutation, CrossoverOperator<DoubleSolution> crossover, FunctionType functionType, String dataDirectory, double neighborhoodSelectionProbability, int maximumNumberOfReplacedSolutions, int neighborSize)
```

Methods

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

initializePopulation

```
public void initializePopulation()
```

run

```
public void run()
```

updateNeighborhood

```
protected void updateNeighborhood(DoubleSolution individual, int subproblemId, NeighborType neighborType)
```

2.17.5 MOEAD

```
public class MOEAD extends AbstractMOEAD<DoubleSolution>
```

Class implementing the MOEA/D-DE algorithm described in : Hui Li; Qingfu Zhang, “Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II,” Evolutionary Computation, IEEE Transactions on , vol.13, no.2, pp.284,302, April 2009. doi: 10.1109/TEVC.2008.925798

Author Antonio J. Nebro

Fields

differentialEvolutionCrossover

```
protected DifferentialEvolutionCrossover differentialEvolutionCrossover
```

Constructors

MOEAD

```
public MOEAD(Problem<DoubleSolution> problem, int populationSize, int resultPopulationSize, int maxEvaluations, MutationOperator<DoubleSolution> mutation, CrossoverOperator<DoubleSolution> crossover, FunctionType functionType, String dataDirectory, double neighborhoodSelectionProbability, int maximumNumberOfReplacedSolutions, int neighborSize)
```

Methods

getDescription

```
public String getDescription()
```

getName

```
public String getName()
```

initializePopulation

protected void **initializePopulation**()

run

public void **run**()

2.17.6 MOEADBuilder

public class **MOEADBuilder** implements *AlgorithmBuilder*<*AbstractMOEAD*<*DoubleSolution*>>

Builder class for algorithm MOEA/D and variants

Author Antonio J. Nebro

Fields**crossover**

protected *CrossoverOperator*<*DoubleSolution*> **crossover**

dataDirectory

protected *String* **dataDirectory**

functionType

protected *MOEAD*.FunctionType **functionType**

maxEvaluations

protected int **maxEvaluations**

maximumNumberOfReplacedSolutions

protected int **maximumNumberOfReplacedSolutions**
nr in Zhang & Li paper

moeadVariant

protected *Variant* **moeadVariant**

mutation

protected *MutationOperator*<*DoubleSolution*> **mutation**

neighborSize

protected int **neighborSize**
T in Zhang & Li paper

neighborhoodSelectionProbability

protected double **neighborhoodSelectionProbability**
Delta in Zhang & Li paper

numberOfThreads

protected int **numberOfThreads**

populationSize

protected int **populationSize**

problem

protected *Problem*<*DoubleSolution*> **problem**

resultPopulationSize

protected int **resultPopulationSize**

Constructors

MOEADBuilder

public **MOEADBuilder** (*Problem*<*DoubleSolution*> *problem*, *Variant* *variant*)
Constructor

Methods

build

public *AbstractMOEAD*<*DoubleSolution*> **build** ()

getCrossover

public *CrossoverOperator*<*DoubleSolution*> **getCrossover** ()

getDataDirectory

```
public String getDataDirectory()
```

getFunctionType

```
public MOEAD.FunctionType getFunctionType()
```

getMaxEvaluations

```
public int getMaxEvaluations()
```

getMaximumNumberOfReplacedSolutions

```
public int getMaximumNumberOfReplacedSolutions()
```

getMutation

```
public MutationOperator<DoubleSolution> getMutation()
```

getNeighborSize

```
public int getNeighborSize()
```

getNeighborhoodSelectionProbability

```
public double getNeighborhoodSelectionProbability()
```

getNumberOfThreads

```
public int getNumberOfThreads()
```

getPopulationSize

```
public int getPopulationSize()
```

getResultPopulationSize

```
public int getResultPopulationSize()
```

setCrossover

```
public MOEADBuilder setCrossover (CrossoverOperator<DoubleSolution> crossover)
```

setDataDirectory

public *MOEADBuilder* **setDataDirectory** (*String* *dataDirectory*)

setFunctionType

public *MOEADBuilder* **setFunctionType** (*MOEAD.FunctionType* *functionType*)

setMaxEvaluations

public *MOEADBuilder* **setMaxEvaluations** (int *maxEvaluations*)

setMaximumNumberOfReplacedSolutions

public *MOEADBuilder* **setMaximumNumberOfReplacedSolutions** (int *maximumNumberOfReplacedSolutions*)

setMutation

public *MOEADBuilder* **setMutation** (*MutationOperator<DoubleSolution>* *mutation*)

setNeighborSize

public *MOEADBuilder* **setNeighborSize** (int *neighborSize*)

setNeighborhoodSelectionProbability

public *MOEADBuilder* **setNeighborhoodSelectionProbability** (double *neighborhoodSelectionProbability*)

setNumberOfThreads

public *MOEADBuilder* **setNumberOfThreads** (int *numberOfThreads*)

setPopulationSize

public *MOEADBuilder* **setPopulationSize** (int *populationSize*)

setResultPopulationSize

public *MOEADBuilder* **setResultPopulationSize** (int *resultPopulationSize*)

2.17.7 MOEADBuilder.Variant

public enum **Variant**

Enum Constants

ConstraintMOEAD

public static final *MOEADBuilder.Variant* **ConstraintMOEAD**

MOEAD

public static final *MOEADBuilder.Variant* **MOEAD**

MOEADD

public static final *MOEADBuilder.Variant* **MOEADD**

MOEADDRA

public static final *MOEADBuilder.Variant* **MOEADDRA**

MOEADSTM

public static final *MOEADBuilder.Variant* **MOEADSTM**

2.17.8 MOEADD

public class **MOEADD**<S extends DoubleSolution> extends *AbstractMOEAD*<S>

Fields

numRanks

protected int **numRanks**

rankIdx

protected int[][] **rankIdx**

ranking

protected *Ranking* **ranking**

subregionDist

protected double[][] **subregionDist**

subregionIdx

protected int[][] **subregionIdx**

Constructors

MOEADD

```
public MOEADD (Problem<S> problem, int populationSize, int resultPopulationSize, int maxEvaluations, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutation, AbstractMOEAD.FunctionType functionType, String dataDirectory, double neighborhoodSelectionProbability, int maximumNumberOfReplacedSolutions, int neighborSize)
```

Methods

calculateDistance

```
public double calculateDistance (S individual, double[] lambda, double[] z_, double[] nz_)  
    Calculate the perpendicular distance between the solution and reference line
```

calculateDistance2

```
public double calculateDistance2 (S indiv, double[] lambda, double[] z_, double[] nz_)
```

checkDominance

```
public int checkDominance (S a, S b)  
    check the dominance relationship between a and b: 1 -> a dominates b, -1 -> b dominates a 0 -> non-dominated with each other
```

computeRanking

```
protected Ranking<S> computeRanking (List<S> solutionList)
```

countOnes

```
public int countOnes (int location)  
    Count the number of 1s in the 'location'th subregion
```

countRankOnes

public int **countRankOnes** (int *location*)
 count the number of 1s in a row of rank matrix

countTest

public int **countTest** ()

deleteCrowdIndiv_diff

public void **deleteCrowdIndiv_diff** (int *crowdIdx*, int *curLocation*, int *nicheCount*, S *indiv*)
 delete one solution from the most crowded subregion, which is different from *indiv*'s subregion. just use *indiv* to replace the worst solution in that subregion

deleteCrowdIndiv_same

public void **deleteCrowdIndiv_same** (int *crowdIdx*, int *nicheCount*, double *indivFitness*, S *indiv*)
 delete one solution from the most crowded subregion, which is *indiv*'s subregion. Compare *indiv*'s fitness value and the worst one in this subregion

deleteCrowdRegion1

public void **deleteCrowdRegion1** (S *indiv*, int *location*)
 Delete a solution from the most crowded subregion (this function only happens when: it should delete 'indiv' based on traditional method. However, the subregion of 'indiv' only has one solution, so it should be kept)

deleteCrowdRegion2

public void **deleteCrowdRegion2** (S *indiv*, int *location*)
 delete a solution from the most crowded subregion (this function happens when: it should delete the solution in the 'parentLocation' subregion, but since this subregion only has one solution, it should be kept)

deleteRankOne

public void **deleteRankOne** (S *indiv*, int *location*)
 if there is only one non-domination level (i.e., all solutions are non-dominated with each other), we should delete a solution from the most crowded subregion

findPosition

public int **findPosition** (S *indiv*)
 find the index of the solution 'indiv' in the population

findRegion

```
public int findRegion (int idx)  
    find the subregion of the 'idx'th solution in the population
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

initPopulation

```
public void initPopulation ()  
    Initialize the population
```

innerproduct

```
public double innerproduct (double[] vec1, double[] vec2)  
    Calculate the dot product of two vectors
```

matingSelection

```
public List<S> matingSelection (int cid, int type)  
    Select two parents for reproduction
```

nondominated_sorting_add

```
public int nondominated_sorting_add (S indiv)  
    update the non-domination level when adding a solution
```

nondominated_sorting_delete

```
public void nondominated_sorting_delete (S indiv)  
    update the non-domination level structure after deleting a solution
```

norm_vector

```
public double norm_vector (double[] z)  
    Calculate the norm of the vector
```

replace

public void **replace** (int *position*, S *solution*)

run

public void **run** ()

setLocation

public void **setLocation** (S *indiv*, double[] *z_*, double[] *nz_*)
Set the location of a solution based on the orthogonal distance

sumFitness

public double **sumFitness** (int *location*)
calculate the sum of fitnesses of solutions in the location subregion

updateArchive

public void **updateArchive** (S *indiv*)
update the parent population by using the ENLU method, instead of fast non-dominated sorting

2.17.9 MOEADDRA

public class **MOEADDRA** extends *AbstractMOEAD<DoubleSolution>*
Class implementing the MOEA/D-DRA algorithm described in : Q. Zhang, W. Liu, and H Li, The Performance of a New Version of MOEA/D on CEC09 Unconstrained MOP Test Instances, Working Report CES-491, School of CS & EE, University of Essex, 02/2009

Author Juan J. Durillo, Antonio J. Nebro

Fields**differentialEvolutionCrossover**

protected *DifferentialEvolutionCrossover* **differentialEvolutionCrossover**

frequency

protected int[] **frequency**

randomGenerator

JMetalRandom **randomGenerator**

savedValues

protected *DoubleSolution*[] **savedValues**

utility

protected double[] **utility**

Constructors

MOEADDRA

```
public MOEADDRA (Problem<DoubleSolution> problem, int populationSize, int resultPopulationSize, int
maxEvaluations, MutationOperator<DoubleSolution> mutation, CrossoverOperator<DoubleSolution> crossover,
FunctionType functionType, String dataDirectory, double neighborhoodSelectionProbability, int maximumNumberOfReplacedSolutions, int
neighborSize)
```

Methods

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

initializePopulation

```
protected void initializePopulation ()
```

run

```
public void run ()
```

tourSelection

```
public List<Integer> tourSelection (int depth)
```

utilityFunction

```
public void utilityFunction ()
```

2.17.10 MOEADDRAIT

public class **MOEADDRAIT**

Fields

algorithm

Algorithm<List<DoubleSolution>> **algorithm**

Methods

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

public void **shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**()

shouldTheHypervolumeHaveAMinimumValue

public void **shouldTheHypervolumeHaveAMinimumValue**()

2.17.11 MOEADIT

public class **MOEADIT**

Fields

algorithm

Algorithm<List<DoubleSolution>> **algorithm**

Methods

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

public void **shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**()

shouldTheHypervolumeHaveAMinimumValue

public void **shouldTheHypervolumeHaveAMinimumValue**()

2.17.12 MOEADSTM

public class **MOEADSTM** extends *AbstractMOEAD<DoubleSolution>*

Class implementing the MOEA/D-STM algorithm described in : K. Li, Q. Zhang, S. Kwong, M. Li and R. Wang, “Stable Matching-Based Selection in Evolutionary Multiobjective Optimization”, IEEE Transactions on Evolutionary Computation, 18(6): 909-923, 2014. DOI: 10.1109/TEVC.2013.2293776

Author Ke Li

Fields

differentialEvolutionCrossover

protected *DifferentialEvolutionCrossover* **differentialEvolutionCrossover**

frequency

protected int[] **frequency**

randomGenerator

JMetalRandom **randomGenerator**

savedValues

protected *DoubleSolution*[] **savedValues**

utility

protected double[] **utility**

Constructors

MOEADSTM

public **MOEADSTM**(*Problem<DoubleSolution>* problem, int populationSize, int resultPopulationSize, int maxEvaluations, *MutationOperator<DoubleSolution>* mutation, *CrossoverOperator<DoubleSolution>* crossover, *FunctionType* functionType, String dataDirectory, double neighborhoodSelectionProbability, int maximumNumberOfReplacedSolutions, int neighborSize)

Methods

calculateDistance

public double **calculateDistance**(*DoubleSolution* individual, double[] lambda)
Calculate the perpendicular distance between the solution and reference line

calculateDistance2

public double **calculateDistance2** (*DoubleSolution* individual, double[] lambda)
 Calculate the perpendicular distance between the solution and reference line

getDescription

public String **getDescription** ()

getName

public String **getName** ()

getResult

public List<*DoubleSolution*> **getResult** ()

initializePopulation

protected void **initializePopulation** ()

innerproduct

public double **innerproduct** (double[] vec1, double[] vec2)
 Calculate the dot product of two vectors

norm_vector

public double **norm_vector** (double[] z)
 Calculate the norm of the vector

prefers

public boolean **prefers** (int x, int y, int[] womanPref, int size)
 Returns true in case that a given woman prefers x to y.

run

public void **run** ()

stableMatching

public int[] **stableMatching** (int[][] manPref, int[][] womanPref, int menSize, int womenSize)
 Return the stable matching between ‘subproblems’ and ‘solutions’ (‘subproblems’ propose first). It is worth noting that the number of solutions is larger than that of the subproblems.

stmSelection

```
public void stmSelection ()  
    Select the next parent population, based on the stable matching criteria
```

tourSelection

```
public List<Integer> tourSelection (int depth)
```

utilityFunction

```
public void utilityFunction ()
```

2.18 org.uma.jmetal.algorithm.multiobjective.moead.util

2.18.1 MOEADUtils

```
public class MOEADUtils  
    Utilities methods to used by MOEA/D
```

Methods

distVector

```
public static double distVector (double[] vector1, double[] vector2)
```

getSubsetOfEvenlyDistributedSolutions

```
public static <S extends Solution<?>> List<S> getSubsetOfEvenlyDistributedSolutions (List<S>  
                                                                                       solu-  
                                                                                       tion-  
                                                                                       List,  
                                                                                       int  
                                                                                       new-  
                                                                                       Solu-  
                                                                                       tion-  
                                                                                       List-  
                                                                                       Size)
```

This methods select a subset of evenly spread solutions from a solution list. The implementation is based on the method described in the MOEA/D-DRA paper.

Parameters

- **solutionList** –
- **newSolutionListSize** –
- **<S>** –

minFastSort

public static void **minFastSort** (double[] *x*, int[] *idx*, int *n*, int *m*)

quickSort

public static void **quickSort** (double[] *array*, int[] *idx*, int *from*, int *to*)
Quick sort procedure (ascending order)

Parameters

- **array** –
- **idx** –
- **from** –
- **to** –

randomPermutation

public static void **randomPermutation** (int[] *perm*, int *size*)

2.19 org.uma.jmetal.algorithm.multiobjective.mombi

2.19.1 AbstractMOMBI

public abstract class **AbstractMOMBI**<S extends Solution<?>> extends *AbstractGeneticAlgorithm*<S, List<S>>
Abstract class representing variants of the MOMBI algorithm

Author Juan J. Durillo Modified by Antonio J. Nebro

Parameters

- **<S>** –

Fields

evaluator

protected final *SolutionListEvaluator*<S> **evaluator**

iterations

protected int **iterations**

maxIterations

protected final int **maxIterations**

nadirPoint

protected final `List<Double>` **nadirPoint**

referencePoint

protected final `List<Double>` **referencePoint**

Constructors

AbstractMOMBI

```
public AbstractMOMBI (Problem<S> problem, int maxIterations, CrossoverOperator<S> crossover, MutationOperator<S> mutation, SelectionOperator<List<S>, S> selection, SolutionListEvaluator<S> evaluator)
```

Constructor

Parameters

- **problem** – Problem to be solved
- **maxIterations** – Maximum number of iterations the algorithm will perform
- **crossover** – Crossover operator
- **mutation** – Mutation operator
- **selection** – Selection operator
- **evaluator** – Evaluator object for evaluating solution lists

Methods

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getNadirPoint

```
public List<Double> getNadirPoint ()
```

getReferencePoint

```
public List<Double> getReferencePoint ()
```

getResult

```
public List<S> getResult ()
```

initProgress

protected void **initProgress** ()

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

populationIsNotFull

protected boolean **populationIsNotFull** (*List<S> population*)

reproduction

protected *List<S>* **reproduction** (*List<S> population*)

run

public void **run** ()

selection

protected *List<S>* **selection** (*List<S> population*)

setReferencePointValue

protected void **setReferencePointValue** (*Double value*, *int index*)

specificMOEAComputations

public abstract void **specificMOEAComputations** ()

updateNadirPoint

protected void **updateNadirPoint** (*S s*)

updateNadirPoint

public void **updateNadirPoint** (*List<S> population*)

updateProgress

protected void **updateProgress** ()

updateReferencePoint

protected void **updateReferencePoint** (S s)

updateReferencePoint

public void **updateReferencePoint** (List<S> population)

2.19.2 MOMBI

public class **MOMBI**<S extends Solution<?>> extends *AbstractMOMBI*<S>

Fields

utilityFunctions

protected final *AbstractUtilityFunctionsSet*<S> **utilityFunctions**

Constructors

MOMBI

public **MOMBI** (*Problem*<S> problem, int maxIterations, *CrossoverOperator*<S> crossover, *MutationOperator*<S> mutation, *SelectionOperator*<List<S>, S> selection, *SolutionListEvaluator*<S> evaluator, String pathWeights)

Methods

addLastRankedSolutionsToPopulation

protected void **addLastRankedSolutionsToPopulation** (*R2Ranking*<S> ranking, int index, List<S> population)

addRankedSolutionsToPopulation

protected void **addRankedSolutionsToPopulation** (*R2Ranking*<S> ranking, int index, List<S> population)

computeRanking

protected *R2Ranking*<S> **computeRanking** (List<S> solutionList)

createUtilityFunction

public *AbstractUtilityFunctionsSet*<S> **createUtilityFunction** (String pathWeights)

getDescription

```
public String getDescription ()
```

getMaxPopulationSize

```
public int getMaxPopulationSize ()
```

getName

```
public String getName ()
```

getUtilityFunctions

```
protected AbstractUtilityFunctionsSet<S> getUtilityFunctions ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

selectBest

```
protected List<S> selectBest (R2Ranking<S> ranking)
```

specificMOEAComputations

```
public void specificMOEAComputations ()
```

2.19.3 MOMBI2

```
public class MOMBI2<S extends Solution<?>> extends MOMBI<S>
```

Author Juan J. Durillo

Fields**alpha**

```
protected final Double alpha
```

epsilon

```
protected final Double epsilon
```

history

protected final *MOMBI2History*<S> **history**

maxs

protected *List*<*Double*> **maxs**

normalizer

protected *Normalizer* **normalizer**

Constructors

MOMBI2

public **MOMBI2** (*Problem*<S> *problem*, int *maxIterations*, *CrossoverOperator*<S> *crossover*, *MutationOperator*<S> *mutation*, *SelectionOperator*<*List*<S>, S> *selection*, *SolutionListEvaluator*<S> *evaluator*, *String* *pathWeights*)

Creates a new instance of the MOMBI algorithm

Parameters

- **problem** –
- **maxIterations** –
- **crossover** –
- **mutation** –
- **selection** –
- **evaluator** –
- **pathWeights** –

Methods

computeRanking

protected *R2Ranking*<S> **computeRanking** (*List*<S> *solutionList*)

createUtilityFunction

public *AbstractUtilityFunctionsSet*<S> **createUtilityFunction** (*String* *pathWeights*)

getDescription

public *String* **getDescription** ()

getMax

```
public Double getMax (List<Double> list)
```

getName

```
public String getName ()
```

initProgress

```
protected void initProgress ()
```

updateMax

```
protected void updateMax (List<S> population)
```

updateReferencePoint

```
public void updateReferencePoint (List<S> population)
```

2.20 org.uma.jmetal.algorithm.multiobjective.mombi.util

2.20.1 ASFUtilityFunctionSet

```
public class ASFUtilityFunctionSet<S extends Solution<?>> extends AbstractUtilityFunctionsSet<S>
```

Author Juan J. Durillo Modified by Antonio J. Nebro

Parameters

- <S> –

Constructors**ASFUtilityFunctionSet**

```
public ASFUtilityFunctionSet (double[][] weights, List<Double> referencePoint)
```

ASFUtilityFunctionSet

```
public ASFUtilityFunctionSet (double[][] weights)
```

ASFUtilityFunctionSet

```
public ASFUtilityFunctionSet (String file_path, List<Double> referencePoint)
```

ASFUtilityFunctionSet

```
public ASFUtilityFunctionSet (String file_path)
```

Methods

evaluate

```
public Double evaluate (S solution, int vector)
```

setNormalizer

```
public void setNormalizer (Normalizer normalizer)
```

2.20.2 ASFWASFGA

```
public class ASFWASFGA<S extends Solution<?>> extends AbstractUtilityFunctionsSet<S>
```

Author Juan J. Durillo Modified by Antonio J. Nebro

Parameters

- *<S>* –

Constructors

ASFWASFGA

```
public ASFWASFGA (double[][] weights, List<Double> interestPoint)
```

ASFWASFGA

```
public ASFWASFGA (double[][] weights)
```

ASFWASFGA

```
public ASFWASFGA (String file_path, List<Double> interestPoint)
```

ASFWASFGA

```
public ASFWASFGA (String file_path)
```

Methods

evaluate

```
public Double evaluate (S solution, int vector)
```

setNadir

```
public void setNadir (List<Double> nadir)
```

setUtopia

```
public void setUtopia (List<Double> utopia)
```

updatePointOfInterest

```
public void updatePointOfInterest (List<Double> newInterestPoint)
```

2.20.3 AbstractUtilityFunctionsSet

public abstract class **AbstractUtilityFunctionsSet**<S extends Solution<?>> implements Serializable

Author Juan J. Durillo Modified by Antonio J. Nebro

Parameters

- *<S>* –

Constructors**AbstractUtilityFunctionsSet**

```
public AbstractUtilityFunctionsSet (double[][] weights)
```

AbstractUtilityFunctionsSet

```
public AbstractUtilityFunctionsSet (String file_path)
```

Methods**evaluate**

```
public List<Double> evaluate (S solution)
```

Evaluates a solution using all the utility functions stored in this set

Parameters

- *solution* –

evaluate

```
public abstract Double evaluate (S solution, int vector)
```

Evaluates a solution using the i-th utility function stored in this set

Parameters

- **solution** –
- **vector** –

getSize

public int **getSize** ()

Returns the number of utility functions stored in this set

Returns The number of vectors

getVectorSize

public int **getVectorSize** ()

Returns the number of components for all weight vectors

getWeightVector

public [List<Double>](#) **getWeightVector** (int *index*)

Returns a given weight vector

loadWeightsFromFile

public void **loadWeightsFromFile** ([String](#) *filePath*)

Reads a set of weight vectors from a file. The expected format for the file is as follows. The first line should start with at least the following three tokens # Any other token on this line will be ignored. indicates how many weight vectors are included in this file indicates how many component has each included vector Each of the following lines of the file represents a weight vector of at least components If more components are provided, they will be ignored by the program

Parameters

- **filePath** – The path in the file system of the file containing the weight vectors

2.20.4 MOMBI2History

public class **MOMBI2History**<T extends [Solution](#)<?>> implements [Serializable](#)

Created by ajnebro on 10/9/15.

Fields

MAX LENGHT

public static final int **MAX LENGHT**

Constructors

MOMBI2History

```
public MOMBI2History (int numberOfObjectives)
```

Methods

add

```
public void add (List<Double> maxs)
```

Adds a new vector of maxs values to the history. The method ensures that only the newest MAX_LENGTH vectors will be kept in the history

Parameters

- **maxs** –

decreaseMark

```
public void decreaseMark (int index)
```

getMaxObjective

```
public Double getMaxObjective (int index)
```

isUnMarked

```
public boolean isUnMarked (int index)
```

mark

```
public void mark (int index)
```

mean

```
public List<Double> mean ()
```

Returns the mean of the values contained in the history

std

```
public List<Double> std (List<Double> mean)
```

Return the std of the values contained in the history

variance

```
public List<Double> variance (List<Double> mean)
```

Returns the variance of the values contained in the history

2.20.5 Normalizer

```
public class Normalizer
```

Constructors

Normalizer

```
public Normalizer (List<Double> min, List<Double> max)
```

Methods

normalize

```
public Double normalize (Double input, int index)
```

2.20.6 R2Ranking

```
public class R2Ranking<S extends Solution<?>> extends GenericSolutionAttribute<S, R2SolutionData>
```

Constructors

R2Ranking

```
public R2Ranking (AbstractUtilityFunctionsSet<S> utilityFunctions)
```

Methods

computeRanking

```
public R2Ranking<S> computeRanking (List<S> population)
```

getAttribute

```
public R2SolutionData getAttribute (S solution)
```

getAttributeIdentifier

```
public Object getAttributeIdentifier ()
```

getNumberOfSubfronts

```
public int getNumberOfSubfronts ()
```

getSubfront

```
public List<S> getSubfront (int rank)
```

getUtilityFunctions

```
public AbstractUtilityFunctionsSet<S> getUtilityFunctions ()
```

setAttribute

```
public void setAttribute (S solution, R2SolutionData value)
```

2.20.7 R2RankingAttribute

```
public class R2RankingAttribute<T extends Solution<?>> extends GenericSolutionAttribute<T, R2SolutionData>  
    Created by ajnebro on 10/9/15.
```

2.20.8 R2RankingNormalized

```
public class R2RankingNormalized<S extends Solution<?>> extends R2Ranking<S>
```

Constructors**R2RankingNormalized**

```
public R2RankingNormalized (AbstractUtilityFunctionsSet<S> utilityFunctions, Normalizer normalizer)
```

Methods**computeRanking**

```
public R2RankingNormalized<S> computeRanking (List<S> population)
```

getNumberOfSubfronts

```
public int getNumberOfSubfronts ()
```

getSubfront

```
public List<S> getSubfront (int rank)
```

2.20.9 R2SolutionData

```
public class R2SolutionData
```

Fields

alpha

```
public double alpha
```

rank

```
public int rank
```

utility

```
public double utility
```

2.20.10 TchebycheffUtilityFunctionsSet

```
public class TchebycheffUtilityFunctionsSet<S extends Solution<?>> extends AbstractUtilityFunctionsSet<S>
```

This class implements a set of utility functions based on the Tchebycheff aggregation approach

Author Juan J. Durillo ToDo List: + check the size of nadir and reference points are the correct ones
+ check that the function that needs to be evaluated is the correct one

Parameters

- <S> –

Constructors

TchebycheffUtilityFunctionsSet

```
public TchebycheffUtilityFunctionsSet (String file_path, List<Double> referencePoint)
```

TchebycheffUtilityFunctionsSet

```
public TchebycheffUtilityFunctionsSet (String file_path)
```

Methods

evaluate

```
public Double evaluate (S solution, int vector)
```


2.21 org.uma.jmetal.algorithm.multiobjective.mombi2

2.21.1 MOMBI2IT

public class **MOMBI2IT**

Fields

algorithm

Algorithm<List<*DoubleSolution*>> **algorithm**

Methods

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

public void **shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**()

shouldTheHypervolumeHaveAMinimumValue

public void **shouldTheHypervolumeHaveAMinimumValue**()

2.22 org.uma.jmetal.algorithm.multiobjective.nsgaii

2.22.1 NSGAI

public class **NSGAI**<S extends *Solution*<?>> extends *AbstractGeneticAlgorithm*<S, List<S>>

Author Antonio J. Nebro

Fields

dominanceComparator

protected *Comparator*<S> **dominanceComparator**

evaluations

protected int **evaluations**

evaluator

protected final *SolutionListEvaluator*<S> **evaluator**

maxEvaluations

protected final int **maxEvaluations**

Constructors

NSGAI

```
public NSGAI (Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S>
             crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>,
             S> selectionOperator, SolutionListEvaluator<S> evaluator)
    Constructor
```

NSGAI

```
public NSGAI (Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S>
             crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>,
             S> selectionOperator, Comparator<S> dominanceComparator, SolutionListEvaluator<S>
             evaluator)
    Constructor
```

Methods

evaluatePopulation

protected List<S> **evaluatePopulation** (List<S> population)

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getNonDominatedSolutions

```
protected List<S> getNonDominatedSolutions (List<S> solutionList)
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected [List<S>](#) **replacement** ([List<S>](#) *population*, [List<S>](#) *offspringPopulation*)

updateProgress

protected void **updateProgress** ()

2.22.2 NSGAI45

public class **NSGAI45**<S extends [Solution](#)<?>> implements [Algorithm](#)<[List](#)<S>>

Implementation of NSGA-II following the scheme used in jMetal4.5 and former versions, i.e, without implementing the [AbstractGeneticAlgorithm](#) interface.

Author Antonio J. Nebro

Fields**crossoverOperator**

protected [CrossoverOperator](#)<S> **crossoverOperator**

evaluations

protected int **evaluations**

evaluator

protected final [SolutionListEvaluator](#)<S> **evaluator**

maxEvaluations

protected final int **maxEvaluations**

mutationOperator

protected [MutationOperator](#)<S> **mutationOperator**

population

protected [List](#)<S> **population**

populationSize

protected final int **populationSize**

problem

protected final *Problem*<S> **problem**

selectionOperator

protected *SelectionOperator*<List<S>, S> **selectionOperator**

Constructors**NSGAI45**

```
public NSGAI45 (Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, SolutionListEvaluator<S> evaluator)
```

Constructor

Methods**addLastRankedSolutionsToPopulation**

```
protected void addLastRankedSolutionsToPopulation (Ranking<S> ranking, int rank, List<S> population)
```

addRankedSolutionsToPopulation

```
protected void addRankedSolutionsToPopulation (Ranking<S> ranking, int rank, List<S> population)
```

computeRanking

```
protected Ranking<S> computeRanking (List<S> solutionList)
```

createInitialPopulation

```
protected List<S> createInitialPopulation ()
```

crowdingDistanceSelection

```
protected List<S> crowdingDistanceSelection (Ranking<S> ranking)
```

evaluatePopulation

protected `List<S>` **evaluatePopulation** (`List<S>` *population*)

getDescription

public `String` **getDescription** ()

getName

public `String` **getName** ()

getNonDominatedSolutions

protected `List<S>` **getNonDominatedSolutions** (`List<S>` *solutionList*)

getResult

public `List<S>` **getResult** ()

populationIsNotFull

protected boolean **populationIsNotFull** (`List<S>` *population*)

run

public void **run** ()
Run method

subfrontFillsIntoThePopulation

protected boolean **subfrontFillsIntoThePopulation** (`Ranking<S>` *ranking*, int *rank*, `List<S>` *population*)

2.22.3 NSGAIIBuilder

public class **NSGAIIBuilder**<S extends `Solution<?>`> implements `AlgorithmBuilder<NSGAII<S>>`

Author Antonio J. Nebro

Constructors

NSGAIIBuilder

```
public NSGAIIBuilder (Problem<S> problem, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator)  
    NSGAIIBuilder constructor
```

Methods

build

```
public NSGAII<S> build ()
```

getCrossoverOperator

```
public CrossoverOperator<S> getCrossoverOperator ()
```

getMaxIterations

```
public int getMaxIterations ()
```

getMutationOperator

```
public MutationOperator<S> getMutationOperator ()
```

getPopulationSize

```
public int getPopulationSize ()
```

getProblem

```
public Problem<S> getProblem ()
```

getSelectionOperator

```
public SelectionOperator<List<S>, S> getSelectionOperator ()
```

getSolutionListEvaluator

```
public SolutionListEvaluator<S> getSolutionListEvaluator ()
```

setDominanceComparator

```
public NSGAIIBuilder<S> setDominanceComparator (Comparator<S> dominanceComparator)
```

setMaxEvaluations

```
public NSGAIIBuilder<S> setMaxEvaluations (int maxEvaluations)
```

setPopulationSize

```
public NSGAIIBuilder<S> setPopulationSize (int populationSize)
```

setSelectionOperator

```
public NSGAIIBuilder<S> setSelectionOperator (SelectionOperator<List<S>, S> selectionOperator)
```

setSolutionListEvaluator

```
public NSGAIIBuilder<S> setSolutionListEvaluator (SolutionListEvaluator<S> evaluator)
```

setVariant

```
public NSGAIIBuilder<S> setVariant (NSGAIIVariant variant)
```

2.22.4 NSGAIIBuilder.NSGAIIVariant

```
public enum NSGAIIVariant
```

Enum Constants**Measures**

```
public static final NSGAIIBuilder.NSGAIIVariant Measures
```

NSGAII

```
public static final NSGAIIBuilder.NSGAIIVariant NSGAII
```

NSGAII45

```
public static final NSGAIIBuilder.NSGAIIVariant NSGAII45
```

SteadyStateNSGAII

```
public static final NSGAIIBuilder.NSGAIIVariant SteadyStateNSGAII
```

2.22.5 NSGAIIBuilderTest

public class **NSGAIIBuilderTest**

Created by Antonio J. Nebro on 25/11/14.

Methods

buildAlgorithm

public void **buildAlgorithm**()

cleanup

public void **cleanup**()

getProblem

public void **getProblem**()

setNegativeMaxNumberOfIterations

public void **setNegativeMaxNumberOfIterations**()

setNegativePopulationSize

public void **setNegativePopulationSize**()

setNewEvaluator

public void **setNewEvaluator**()

setNewSelectionOperator

public void **setNewSelectionOperator**()

setNullEvaluator

public void **setNullEvaluator**()

setNullSelectionOperator

public void **setNullSelectionOperator**()

setPositiveMaxNumberOfIterations

```
public void setPositiveMaxNumberOfIterations()
```

setValidPopulationSize

```
public void setValidPopulationSize()
```

startup

```
public void startup()
```

testDefaultConfiguration

```
public void testDefaultConfiguration()
```

2.22.6 NSGAIIT

```
public class NSGAIIT
```

Fields**algorithm**

```
Algorithm<List<DoubleSolution>> algorithm
```

Methods**shouldTheAlgorithmReturnAGoodQualityFrontWhenSolvingAConstrainedProblem**

```
public void shouldTheAlgorithmReturnAGoodQualityFrontWhenSolvingAConstrainedProblem()
```

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem()
```

2.22.7 NSGAII Measures

```
public class NSGAII Measures<S extends Solution<?>> extends NSGAI<S> implements Measurable
```

Author Antonio J. Nebro

Fields

durationMeasure

protected *DurationMeasure* **durationMeasure**

evaluations

protected *CountingMeasure* **evaluations**

hypervolumeValue

protected *BasicMeasure*<Double> **hypervolumeValue**

measureManager

protected *SimpleMeasureManager* **measureManager**

numberOfNonDominatedSolutionsInPopulation

protected *BasicMeasure*<Integer> **numberOfNonDominatedSolutionsInPopulation**

referenceFront

protected *Front* **referenceFront**

solutionListMeasure

protected *BasicMeasure*<List<S>> **solutionListMeasure**

Constructors

NSGAII Measures

```
public NSGAII Measures (Problem<S> problem, int maxIterations, int populationSize, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, Comparator<S> dominanceComparator, SolutionListEvaluator<S> evaluator)
```

Constructor

Methods

getDescription

```
public String getDescription ()
```

getEvaluations

```
public CountingMeasure getEvaluations ()
```

getMeasureManager

```
public MeasureManager getMeasureManager ()
```

getName

```
public String getName ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

run

```
public void run ()
```

setReferenceFront

```
public void setReferenceFront (Front referenceFront)
```

updateProgress

```
protected void updateProgress ()
```

2.22.8 NSGAIIStoppingByTime

```
public class NSGAIIStoppingByTime<S extends Solution<?>> extends NSGAII<S>
```

This class shows a version of NSGA-II having a stopping condition depending on run-time

Author Antonio J. Nebro

Constructors

NSGAIIStoppingByTime

```
public NSGAIIStoppingByTime (Problem<S> problem, int populationSize, long maxComputingTime,  
                             CrossoverOperator<S> crossoverOperator, MutationOperator<S> mu-  
                             tationOperator, SelectionOperator<List<S>, S> selectionOperator,  
                             Comparator<S> dominanceComparator)
```

Constructor

Methods

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

updateProgress

```
protected void updateProgress ()
```

2.22.9 SteadyStateNSGAIIS

```
public class SteadyStateNSGAIIS<S extends Solution<?>> extends NSGAIIS<S>
```

Author Antonio J. Nebro

Constructors

SteadyStateNSGAI

```
public SteadyStateNSGAI (Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, Comparator<S> dominanceComparator, SolutionListEvaluator<S> evaluator)
```

Constructor

Methods

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

reproduction

```
protected List<S> reproduction (List<S> population)
```

selection

```
protected List<S> selection (List<S> population)
```

updateProgress

```
protected void updateProgress ()
```

2.23 org.uma.jmetal.algorithm.multiobjective.nsgaiii

2.23.1 NSGAI

```
public class NSGAI<S extends Solution<?>> extends AbstractGeneticAlgorithm<S, List<S>>
    Created by ajnebro on 30/10/14. Modified by Juanjo on 13/11/14 This implementation is based on the code of
    Tsung-Che Chiang http://web.ntnu.edu.tw/~tcchiang/publications/nsga3cpp/nsga3cpp.htm
```

Fields

evaluator

```
protected SolutionListEvaluator<S> evaluator
```

iterations

protected int **iterations**

maxIterations

protected int **maxIterations**

numberOfDivisions

protected `Vector<Integer>` **numberOfDivisions**

referencePoints

protected `List<ReferencePoint<S>>` **referencePoints**

Constructors

NSGAIII

```
public NSGAIII (NSGAIIIBuilder<S> builder)
    Constructor
```

Methods

addRankedSolutionsToPopulation

```
protected void addRankedSolutionsToPopulation (Ranking<S> ranking, int rank, List<S> population)
```

computeRanking

```
protected Ranking<S> computeRanking (List<S> solutionList)
```

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getNonDominatedSolutions

protected `List<S>` **getNonDominatedSolutions** (`List<S>` *solutionList*)

getResult

public `List<S>` **getResult** ()

initProgress

protected void **initProgress** ()

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected `List<S>` **replacement** (`List<S>` *population*, `List<S>` *offspringPopulation*)

reproduction

protected `List<S>` **reproduction** (`List<S>` *population*)

selection

protected `List<S>` **selection** (`List<S>` *population*)

updateProgress

protected void **updateProgress** ()

2.23.2 NSGAIIIBuilder

public class **NSGAIIIBuilder**<S extends `Solution`<?>> implements *AlgorithmBuilder*<*NSGAIII*<S>>
Builder class

Constructors

NSGAIIIBuilder

public **NSGAIIIBuilder** (*Problem*<S> *problem*)
Builder constructor

Methods

build

```
public NSGAIII<S> build()
```

getCrossoverOperator

```
public CrossoverOperator<S> getCrossoverOperator()
```

getEvaluator

```
public SolutionListEvaluator<S> getEvaluator()
```

getMaxIterations

```
public int getMaxIterations()
```

getMutationOperator

```
public MutationOperator<S> getMutationOperator()
```

getPopulationSize

```
public int getPopulationSize()
```

getProblem

```
public Problem<S> getProblem()
```

getSelectionOperator

```
public SelectionOperator<List<S>, S> getSelectionOperator()
```

setCrossoverOperator

```
public NSGAIIIBuilder<S> setCrossoverOperator(CrossoverOperator<S> crossoverOperator)
```

setMaxIterations

```
public NSGAIIIBuilder<S> setMaxIterations(int maxIterations)
```


setMutationOperator

```
public NSGAIIIBuilder<S> setMutationOperator (MutationOperator<S> mutationOperator)
```

setPopulationSize

```
public NSGAIIIBuilder<S> setPopulationSize (int populationSize)
```

setSelectionOperator

```
public NSGAIIIBuilder<S> setSelectionOperator (SelectionOperator<List<S>, S> selectionOperator)
```

setSolutionListEvaluator

```
public NSGAIIIBuilder<S> setSolutionListEvaluator (SolutionListEvaluator<S> evaluator)
```

2.24 org.uma.jmetal.algorithm.multiobjective.nsgaiii.util

2.24.1 EnvironmentalSelection

```
public class EnvironmentalSelection<S extends Solution<?>> implements SelectionOperator<List<S>, List<S>>>, SolutionAttr
```

Constructors**EnvironmentalSelection**

```
public EnvironmentalSelection (Builder<S> builder)
```

EnvironmentalSelection

```
public EnvironmentalSelection (List<List<S>> fronts, int solutionsToSelect,  
List<ReferencePoint<S>> referencePoints, int numberOfObjectives)
```

Methods**FindNicheReferencePoint**

```
int FindNicheReferencePoint ()
```

SelectClusterMember

```
S SelectClusterMember (ReferencePoint<S> rp)
```

associate

public void **associate** (*List<S> population*)

constructHyperplane

public *List<Double>* **constructHyperplane** (*List<S> population*, *List<S> extreme_points*)

execute

public *List<S>* **execute** (*List<S> source*)

getAttribute

public *List<Double>* **getAttribute** (*S solution*)

getAttributeIdentifier

public *Object* **getAttributeIdentifier** ()

guassianElimination

public *List<Double>* **guassianElimination** (*List<List<Double>> A*, *List<Double> b*)

normalizeObjectives

public void **normalizeObjectives** (*List<S> population*, *List<Double> intercepts*, *List<Double> ideal_point*)

perpendicularDistance

public double **perpendicularDistance** (*List<Double> direction*, *List<Double> point*)

setAttribute

public void **setAttribute** (*S solution*, *List<Double> value*)

translateObjectives

public *List<Double>* **translateObjectives** (*List<S> population*)

2.24.2 EnvironmentalSelection.Builder

public static class **Builder**<S extends *Solution*<?>>

Methods

build

```
public EnvironmentalSelection<S> build ()
```

getFronts

```
public List<List<S>> getFronts ()
```

getNumberOfObjectives

```
public int getNumberOfObjectives ()
```

getReferencePoints

```
public List<ReferencePoint<S>> getReferencePoints ()
```

getSolutionsToSelect

```
public int getSolutionsToSelect ()
```

setFronts

```
public Builder<S> setFronts (List<List<S>> f)
```

setNumberOfObjectives

```
public Builder<S> setNumberOfObjectives (int n)
```

setReferencePoints

```
public Builder<S> setReferencePoints (List<ReferencePoint<S>> referencePoints)
```

setSolutionsToSelect

```
public Builder<S> setSolutionsToSelect (int solutions)
```

2.24.3 ReferencePoint

```
public class ReferencePoint<S extends Solution<?>>
```

Created by ajnebro on 5/11/14. Modified by Juanjo on 13/11/14 This implementation is based on the code of
Tsung-Che Chiang <http://web.ntnu.edu.tw/~tcchiang/publications/nsga3cpp/nsga3cpp.htm>

Fields

position

```
public List<Double> position
```

Constructors

ReferencePoint

```
public ReferencePoint ()
```

ReferencePoint

```
public ReferencePoint (int size)  
    Constructor
```

ReferencePoint

```
public ReferencePoint (ReferencePoint<S> point)
```

Methods

AddMember

```
public void AddMember ()
```

AddPotentialMember

```
public void AddPotentialMember (S member_ind, double distance)
```

FindClosestMember

```
public S FindClosestMember ()
```

HasPotentialMember

```
public boolean HasPotentialMember ()
```

MemberSize

```
public int MemberSize ()
```

RandomMember

```
public S RandomMember ()
```

RemovePotentialMember

```
public void RemovePotentialMember (S solution)
```

clear

```
public void clear ()
```

generateReferencePoints

```
public void generateReferencePoints (List<ReferencePoint<S>> referencePoints, int numberOfObjectives, List<Integer> numberOfDivisions)
```

pos

```
public List<Double> pos ()
```

2.25 org.uma.jmetal.algorithm.multiobjective.omopso**2.25.1 OMOPSO**

```
public class OMOPSO extends AbstractParticleSwarmOptimization<DoubleSolution, List<DoubleSolution>>
    Class implementing the OMOPSO algorithm
```

Fields**evaluator**

```
SolutionListEvaluator<DoubleSolution> evaluator
```

Constructors**OMOPSO**

```
public OMOPSO (DoubleProblem problem, SolutionListEvaluator<DoubleSolution> evaluator, int swarmSize,
               int maxIterations, int archiveSize, UniformMutation uniformMutation, NonUniformMutation
               nonUniformMutation)
    Constructor
```

Methods

createInitialSwarm

protected `List<DoubleSolution>` **createInitialSwarm** ()

evaluateSwarm

protected `List<DoubleSolution>` **evaluateSwarm** (`List<DoubleSolution>` *swarm*)

getDescription

public `String` **getDescription** ()

getName

public `String` **getName** ()

getResult

public `List<DoubleSolution>` **getResult** ()

initProgress

protected void **initProgress** ()

initializeLeader

protected void **initializeLeader** (`List<DoubleSolution>` *swarm*)

initializeParticlesMemory

protected void **initializeParticlesMemory** (`List<DoubleSolution>` *swarm*)

initializeVelocity

protected void **initializeVelocity** (`List<DoubleSolution>` *swarm*)

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

perturbation

protected void **perturbation** (*List<DoubleSolution> swarm*)
Apply a mutation operator to all particles in the swarm (perturbation)

tearDown

protected void **tearDown** ()

updateLeaders

protected void **updateLeaders** (*List<DoubleSolution> swarm*)
Update leaders method

Parameters

- **swarm** – List of solutions (swarm)

updateParticlesMemory

protected void **updateParticlesMemory** (*List<DoubleSolution> swarm*)

updatePosition

protected void **updatePosition** (*List<DoubleSolution> swarm*)
Update the position of each particle

updateProgress

protected void **updateProgress** ()

updateVelocity

protected void **updateVelocity** (*List<DoubleSolution> swarm*)

2.25.2 OMOPSOBuilder

public class **OMOPSOBuilder** implements *AlgorithmBuilder<OMOPSO>*
Class implementing the OMOPSO algorithm

Fields

evaluator

protected *SolutionListEvaluator<DoubleSolution>* **evaluator**

problem

protected *DoubleProblem* **problem**

Constructors

OMOPSOBuilder

public **OMOPSOBuilder** (*DoubleProblem* problem, *SolutionListEvaluator*<*DoubleSolution*> evaluator)

Methods

build

public *OMOPSO* **build** ()

getArchiveSize

public int **getArchiveSize** ()

getMaxIterations

public int **getMaxIterations** ()

getNonUniformMutation

public *NonUniformMutation* **getNonUniformMutation** ()

getSwarmSize

public int **getSwarmSize** ()

getUniformMutation

public *UniformMutation* **getUniformMutation** ()

setArchiveSize

public *OMOPSOBuilder* **setArchiveSize** (int archiveSize)

setMaxIterations

public *OMOPSOBuilder* **setMaxIterations** (int maxIterations)

setNonUniformMutation

```
public OMOPSOBuilder setNonUniformMutation (MutationOperator<DoubleSolution> nonUniformMutation)
```

setSwarmSize

```
public OMOPSOBuilder setSwarmSize (int swarmSize)
```

setUniformMutation

```
public OMOPSOBuilder setUniformMutation (MutationOperator<DoubleSolution> uniformMutation)
```

2.25.3 OMPSOIT

```
public class OMPSOIT
    Integration tests for algorithm OMPSO
    Author Antonio J. Nebro
```

Fields**algorithm**

```
Algorithm<List<DoubleSolution>> algorithm
```

Methods**shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem ()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue ()
```

2.26 org.uma.jmetal.algorithm.multiobjective.paes**2.26.1 PAES**

```
public class PAES<S extends Solution<?>> extends AbstractEvolutionStrategy<S, List<S>>
    Author Antonio J. Nebro, Juan J. Durillo
```

Fields

archive

protected *AdaptiveGridArchive*<S> **archive**

archiveSize

protected int **archiveSize**

biSections

protected int **biSections**

comparator

protected *Comparator*<S> **comparator**

evaluations

protected int **evaluations**

maxEvaluations

protected int **maxEvaluations**

Constructors

PAES

public **PAES** (*Problem*<S> *problem*, int *archiveSize*, int *maxEvaluations*, int *biSections*, *MutationOperator*<S>
 mutationOperator)
 Constructor

Methods

createInitialPopulation

protected *List*<S> **createInitialPopulation** ()

evaluatePopulation

protected *List*<S> **evaluatePopulation** (*List*<S> *population*)

getArchiveSize

```
public int getArchiveSize ()
```

getBiSections

```
public int getBiSections ()
```

getDescription

```
public String getDescription ()
```

getMaxEvaluations

```
public int getMaxEvaluations ()
```

getMutationOperator

```
public MutationOperator<S> getMutationOperator ()
```

getName

```
public String getName ()
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

reproduction

```
protected List<S> reproduction (List<S> population)
```

selection

protected `List<S> selection (List<S> population)`

test

public `S test (S solution, S mutatedSolution, AdaptiveGridArchive<S> archive)`

Tests two solutions to determine which one becomes be the guide of PAES algorithm

Parameters

- **solution** – The actual guide of PAES
- **mutatedSolution** – A candidate guide

updateProgress

protected void `updateProgress ()`

2.26.2 PAESBuilder

public class **PAESBuilder**<S extends Solution<?>> implements *AlgorithmBuilder*<PAES<S>>

Author Antonio J. Nebro

Constructors

PAESBuilder

public **PAESBuilder** (*Problem*<S> problem)

Methods

build

public *PAES*<S> **build** ()

getArchiveSize

public int **getArchiveSize** ()

getBiSections

public int **getBiSections** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getMutationOperator

```
public MutationOperator<S> getMutationOperator ()
```

getProblem

```
public Problem<S> getProblem ()
```

setArchiveSize

```
public PAESBuilder<S> setArchiveSize (int archiveSize)
```

setBiSections

```
public PAESBuilder<S> setBiSections (int biSections)
```

setMaxEvaluations

```
public PAESBuilder<S> setMaxEvaluations (int maxEvaluations)
```

setMutationOperator

```
public PAESBuilder<S> setMutationOperator (MutationOperator<S> mutation)
```

2.27 org.uma.jmetal.algorithm.multiobjective.pesa2

2.27.1 PESA2

```
public class PESA2<S extends Solution<?>> extends AbstractGeneticAlgorithm<S, List<S>>
```

Author Antonio J. Nebro

Fields**evaluator**

```
protected final SolutionListEvaluator<S> evaluator
```

selectionOperator

```
protected SelectionOperator<AdaptiveGridArchive<S>, S> selectionOperator
```

Constructors

PESA2

```
public PESA2 (Problem<S> problem, int maxEvaluations, int populationSize, int archiveSize, int biSections,
             CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SolutionListEvaluator<S> evaluator)
```

Methods

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

reproduction

```
protected List<S> reproduction (List<S> population)
```

selection

```
protected List<S> selection (List<S> population)
```

updateProgress

protected void **updateProgress** ()

2.27.2 PESA2Builder

public class **PESA2Builder**<S extends Solution<?>> implements *AlgorithmBuilder*<*PESA2*<S>>
Created by Antonio J. Nebro

Constructors

PESA2Builder

public **PESA2Builder** (*Problem*<S> problem, *CrossoverOperator*<S> crossoverOperator, *MutationOperator*<S> mutationOperator)
Constructor

Methods

build

public *PESA2*<S> **build** ()

getArchiveSize

public int **getArchiveSize** ()

getBiSections

public int **getBiSections** ()

getCrossoverOperator

public *CrossoverOperator*<S> **getCrossoverOperator** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

getPopulationSize

```
public int getPopulationSize ()
```

getProblem

```
public Problem<S> getProblem ()
```

getSolutionListEvaluator

```
public SolutionListEvaluator<S> getSolutionListEvaluator ()
```

setArchiveSize

```
public PESA2Builder<S> setArchiveSize (int archiveSize)
```

setBisections

```
public PESA2Builder<S> setBisections (int biSections)
```

setMaxEvaluations

```
public PESA2Builder<S> setMaxEvaluations (int maxEvaluations)
```

setPopulationSize

```
public PESA2Builder<S> setPopulationSize (int populationSize)
```

setSolutionListEvaluator

```
public PESA2Builder<S> setSolutionListEvaluator (SolutionListEvaluator<S> evaluator)
```

2.28 org.uma.jmetal.algorithm.multiobjective.pesa2.util

2.28.1 PESA2Selection

```
public class PESA2Selection<S extends Solution<?>> implements SelectionOperator<AdaptiveGridArchive<S>, S>
    This class implements a selection operator as the used in the PESA-II algorithm
```

Constructors

PESA2Selection

```
public PESA2Selection ()
```


Methods

execute

public S **execute** (*AdaptiveGridArchive*<S> archive)

2.29 org.uma.jmetal.algorithm.multiobjective.randomsearch

2.29.1 RandomSearch

public class **RandomSearch**<S extends *Solution*<?>> implements *Algorithm*<*List*<S>>
This class implements a simple random search algorithm.

Author Antonio J. Nebro

Fields

nonDominatedArchive

NonDominatedSolutionListArchive<S> **nonDominatedArchive**

Constructors

RandomSearch

public **RandomSearch** (*Problem*<S> problem, int maxEvaluations)
Constructor

Methods

getDescription

public *String* **getDescription** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getName

public *String* **getName** ()

getResult

public *List*<S> **getResult** ()

run

```
public void run ()
```

2.29.2 RandomSearchBuilder

public class **RandomSearchBuilder**<S extends Solution<?>> implements *AlgorithmBuilder*<*RandomSearch*<S>>
This class implements a simple random search algorithm.

Author Antonio J. Nebro

Constructors

RandomSearchBuilder

```
public RandomSearchBuilder (Problem<S> problem)
```

Methods

build

```
public RandomSearch<S> build ()
```

getMaxEvaluations

```
public int getMaxEvaluations ()
```

setMaxEvaluations

```
public RandomSearchBuilder<S> setMaxEvaluations (int maxEvaluations)
```

2.30 org.uma.jmetal.algorithm.multiobjective.rnsgaii

2.30.1 RNSGAI

public class **RNSGAI**<S extends Solution<?>> extends *RNSGAI*<S> implements *InteractiveAlgorithm*<S, List<S>>

Author Antonio J. Nebro

Constructors

RNSGAI

```
public RNSGAI (Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, SolutionListEvaluator<S> evaluator, List<Double> interestPoint, double epsilon)
```

Constructor

Methods

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

updatePointOfInterest

```
public void updatePointOfInterest (List<Double> newReferencePoints)
```

updateProgress

```
protected void updateProgress ()
```

2.30.2 RNSGAIIBuilder

public class **RNSGAIIBuilder**<S extends Solution<?>> implements *AlgorithmBuilder*<*RNSGAII*<S>>

Author Antonio J. Nebro

Constructors

RNSGAIIBuilder

public **RNSGAIIBuilder** (*Problem*<S> *problem*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*, *List*<*Double*> *interestPoint*, double *epsilon*)
RNSGAIIBuilder constructor

Methods

build

public *RNSGAII*<S> **build** ()

getCrossoverOperator

public *CrossoverOperator*<S> **getCrossoverOperator** ()

getMaxIterations

public int **getMaxIterations** ()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

getPopulationSize

public int **getPopulationSize** ()

getProblem

public *Problem*<S> **getProblem** ()

getSelectionOperator

public *SelectionOperator*<*List*<S>, S> **getSelectionOperator** ()

getSolutionListEvaluator

```
public SolutionListEvaluator<S> getSolutionListEvaluator ()
```

setMaxEvaluations

```
public RNSGAIIBuilder<S> setMaxEvaluations (int maxEvaluations)
```

setPopulationSize

```
public RNSGAIIBuilder<S> setPopulationSize (int populationSize)
```

setSelectionOperator

```
public RNSGAIIBuilder<S> setSelectionOperator (SelectionOperator<List<S>, S> selectionOperator)
```

setSolutionListEvaluator

```
public RNSGAIIBuilder<S> setSolutionListEvaluator (SolutionListEvaluator<S> evaluator)
```

setVariant

```
public RNSGAIIBuilder<S> setVariant (NSGAIIVariant variant)
```

2.30.3 RNSGAIIBuilder.NSGAIIVariant

```
public enum NSGAIIVariant
```

Enum Constants**Measures**

```
public static final RNSGAIIBuilder.NSGAIIVariant Measures
```

NSGAII

```
public static final RNSGAIIBuilder.NSGAIIVariant NSGAII
```

NSGAII45

```
public static final RNSGAIIBuilder.NSGAIIVariant NSGAII45
```

SteadyStateNSGAI

```
public static final RNSGAIIBuilder.NSGAIIVariant SteadyStateNSGAI
```

2.31 org.uma.jmetal.algorithm.multiobjective.rnsgaii.util

2.31.1 PreferenceNSGAI

```
public class PreferenceNSGAI<S extends Solution<?>>
```

Constructors

PreferenceNSGAI

```
public PreferenceNSGAI (List<Double> weights)
```

Methods

evaluate

```
public Double evaluate (S solution)
```

getSize

```
public int getSize ()
```

setLowerBounds

```
public void setLowerBounds (List<Double> lowerBounds)
```

setUpperBounds

```
public void setUpperBounds (List<Double> upperBounds)
```

updatePointOfInterest

```
public void updatePointOfInterest (List<Double> newInterestPoint)
```

2.31.2 RNSGAIIRanking

```
public class RNSGAIIRanking<S extends Solution<?>> extends GenericSolutionAttribute<S, Integer> implements Ranking<S>
```

Constructors

RNSGAIIRanking

```
public RNSGAIIRanking (PreferenceNSGAII<S> utilityFunctions, double epsilon, List<Double> interest-
    Point)
```

Methods

computeRanking

```
public Ranking<S> computeRanking (List<S> population)
```

getNumberOfSubfronts

```
public int getNumberOfSubfronts ()
```

getSubfront

```
public List<S> getSubfront (int rank)
```

2.32 org.uma.jmetal.algorithm.multiobjective.smpso

2.32.1 SMPSO

```
public class SMPSO extends AbstractParticleSwarmOptimization<DoubleSolution, List<DoubleSolution>>
    This class implements the SMPSO algorithm described in: SMPSO: A new PSO-based metaheuristic for multi-
    objective optimization MCDM 2009. DOI: http://dx.doi.org/10.1109/MCDM.2009.4938830
```

Author Antonio J. Nebro

Constructors

SMPSO

```
public SMPSO (DoubleProblem problem, int swarmSize, BoundedArchive<DoubleSolution> leaders, Mutation-
    Operator<DoubleSolution> mutationOperator, int maxIterations, double r1Min, double r1Max,
    double r2Min, double r2Max, double c1Min, double c1Max, double c2Min, double c2Max, dou-
    ble weightMin, double weightMax, double changeVelocity1, double changeVelocity2, Solution-
    ListEvaluator<DoubleSolution> evaluator)
    Constructor
```

Methods

constrictionCoefficient

```
protected double constrictionCoefficient (double c1, double c2)
```

createInitialSwarm

```
protected List<DoubleSolution> createInitialSwarm ()
```

evaluateSwarm

```
protected List<DoubleSolution> evaluateSwarm (List<DoubleSolution> swarm)
```

getDescription

```
public String getDescription ()
```

getIterations

```
public int getIterations ()
```

getMaxIterations

```
public int getMaxIterations ()
```

getName

```
public String getName ()
```

getResult

```
public List<DoubleSolution> getResult ()
```

getSwarmSize

```
public int getSwarmSize ()
```

initProgress

```
protected void initProgress ()
```

initializeLeader

```
protected void initializeLeader (List<DoubleSolution> swarm)
```

initializeParticlesMemory

```
protected void initializeParticlesMemory (List<DoubleSolution> swarm)
```


initializeVelocity

protected void **initializeVelocity** (*List<DoubleSolution> swarm*)

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

perturbation

protected void **perturbation** (*List<DoubleSolution> swarm*)

selectGlobalBest

protected *DoubleSolution* **selectGlobalBest** ()

setIterations

public void **setIterations** (int *iterations*)

updateLeaders

protected void **updateLeaders** (*List<DoubleSolution> swarm*)

updateLeadersDensityEstimator

protected void **updateLeadersDensityEstimator** ()

updateParticlesMemory

protected void **updateParticlesMemory** (*List<DoubleSolution> swarm*)

updatePosition

protected void **updatePosition** (*List<DoubleSolution> swarm*)

updateProgress

protected void **updateProgress** ()

updateVelocity

protected void **updateVelocity** (*List<DoubleSolution> swarm*)

2.32.2 SMPSOBuilder

public class **SMPSOBuilder** implements *AlgorithmBuilder*<*SMPSO*>

Author Antonio J. Nebro

Fields

archiveSize

protected int **archiveSize**

evaluator

protected *SolutionListEvaluator*<*DoubleSolution*> **evaluator**

leaders

protected *BoundedArchive*<*DoubleSolution*> **leaders**

mutationOperator

protected *MutationOperator*<*DoubleSolution*> **mutationOperator**

variant

protected *SMPSOVariant* **variant**

Constructors

SMPSOBuilder

public **SMPSOBuilder** (*DoubleProblem* problem, *BoundedArchive*<*DoubleSolution*> leaders)

Methods

build

public *SMPSO* **build** ()

getArchiveSize

public int **getArchiveSize** ()

getC1Max

public double **getC1Max** ()

getC1Min

public double **getC1Min** ()

getC2Max

public double **getC2Max** ()

getC2Min

public double **getC2Min** ()

getChangeVelocity1

public double **getChangeVelocity1** ()

getChangeVelocity2

public double **getChangeVelocity2** ()

getEvaluator

public *SolutionListEvaluator*<*DoubleSolution*> **getEvaluator** ()

getLeaders

public *BoundedArchive*<*DoubleSolution*> **getLeaders** ()

getMaxIterations

public int **getMaxIterations** ()

getMutation

public *MutationOperator*<*DoubleSolution*> **getMutation** ()

getMutationOperator

public *MutationOperator*<*DoubleSolution*> **getMutationOperator** ()

getProblem

public *DoubleProblem* **getProblem** ()

getR1Max

public double **getR1Max** ()

getR1Min

public double **getR1Min** ()

getR2Max

public double **getR2Max** ()

getR2Min

public double **getR2Min** ()

getSwarmSize

public int **getSwarmSize** ()

getWeightMax

public double **getWeightMax** ()

getWeightMin

public double **getWeightMin** ()

setC1Max

public *SMPSOBuilder* **setC1Max** (double *c1Max*)

setC1Min

public *SMPSOBuilder* **setC1Min** (double *c1Min*)

setC2Max

public *SMPSOBuilder* **setC2Max** (double *c2Max*)

setC2Min

public *SMPSOBuilder* **setC2Min** (double *c2Min*)

setChangeVelocity1

public *SMPSOBuilder* **setChangeVelocity1** (double *changeVelocity1*)

setChangeVelocity2

public *SMPSOBuilder* **setChangeVelocity2** (double *changeVelocity2*)

setMaxIterations

public *SMPSOBuilder* **setMaxIterations** (int *maxIterations*)

setMutation

public *SMPSOBuilder* **setMutation** (*MutationOperator*<*DoubleSolution*> *mutation*)

setR1Max

public *SMPSOBuilder* **setR1Max** (double *r1Max*)

setR1Min

public *SMPSOBuilder* **setR1Min** (double *r1Min*)

setR2Max

public *SMPSOBuilder* **setR2Max** (double *r2Max*)

setR2Min

public *SMPSOBuilder* **setR2Min** (double *r2Min*)

setRandomGenerator

public *SMPSOBuilder* **setRandomGenerator** (*PseudoRandomGenerator* *randomGenerator*)

setSolutionListEvaluator

public *SMPSOBuilder* **setSolutionListEvaluator** (*SolutionListEvaluator*<*DoubleSolution*> *evaluator*)

setSwarmSize

```
public SMPSOBuilder setSwarmSize (int swarmSize)
```

setVariant

```
public SMPSOBuilder setVariant (SMPSOVariant variant)
```

setWeightMax

```
public SMPSOBuilder setWeightMax (double weightMax)
```

setWeightMin

```
public SMPSOBuilder setWeightMin (double weightMin)
```

2.32.3 SMPSOBuilder.SMP SOVariant

```
public enum SMP SOVariant
```

Enum Constants

Measures

```
public static final SMPSOBuilder.SMP SOVariant Measures
```

SMPSO

```
public static final SMPSOBuilder.SMP SOVariant SMPSO
```

2.32.4 SMP SOIT

```
public class SMP SOIT
```

Fields

algorithm

```
Algorithm<List<DoubleSolution>> algorithm
```

Methods

shouldTheAlgorithmReturnAGoodQualityFrontWhenSolvingAConstrainedProblem

```
public void shouldTheAlgorithmReturnAGoodQualityFrontWhenSolvingAConstrainedProblem ()
```

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue()
```

2.32.5 SMPSOMeasures

public class **SMPSOMeasures** extends *SMPSO* implements *Measurable*

This class implements a version of SMPSO using measures

Author Antonio J. Nebro

Fields**durationMeasure**

```
protected DurationMeasure durationMeasure
```

iterations

```
protected CountingMeasure iterations
```

measureManager

```
protected SimpleMeasureManager measureManager
```

solutionListMeasure

```
protected BasicMeasure<List<DoubleSolution>> solutionListMeasure
```

Constructors**SMPSOMeasures**

```
public SMPSOMeasures (DoubleProblem problem, int swarmSize, BoundedArchive<DoubleSolution> leaders, MutationOperator<DoubleSolution> mutationOperator, int maxIterations, double r1Min, double r1Max, double r2Min, double r2Max, double c1Min, double c1Max, double c2Min, double c2Max, double weightMin, double weightMax, double changeVelocity1, double changeVelocity2, SolutionListEvaluator<DoubleSolution> evaluator)
```

Constructor

Parameters

- **problem** –

- `swarmSize` –
- `leaders` –
- `mutationOperator` –
- `maxIterations` –
- `r1Min` –
- `r1Max` –
- `r2Min` –
- `r2Max` –
- `c1Min` –
- `c1Max` –
- `c2Min` –
- `c2Max` –
- `weightMin` –
- `weightMax` –
- `changeVelocity1` –
- `changeVelocity2` –
- `evaluator` –

Methods

`getDescription`

```
public String getDescription()
```

`getMeasureManager`

```
public MeasureManager getMeasureManager()
```

`getName`

```
public String getName()
```

`initProgress`

```
protected void initProgress()
```

`isStoppingConditionReached`

```
protected boolean isStoppingConditionReached()
```


run

public void **run** ()

updateProgress

protected void **updateProgress** ()

2.32.6 SMPSORP

public class **SMPSORP** extends *AbstractParticleSwarmOptimization<DoubleSolution, List<DoubleSolution>>* implements *Measurable*

This class implements the SMPSORP algorithm described in: “Extending the Speed-constrained Multi-Objective PSO (SMPSO) With Reference Point Based Preference Articulation. Antonio J. Nebro, Juan J. Durillo, José García-Nieto, Cristóbal Barba-González, Javier Del Ser, Carlos A. Coello Coello, Antonio Benítez-Hidalgo, José F. Aldana-Montes. Parallel Problem Solving from Nature – PPSN XV. Lecture Notes In Computer Science, Vol. 11101, pp. 298-310. 2018”.

Author Antonio J. Nebro

Fields

currentIteration

protected *CountingMeasure* **currentIteration**

deltaMax

protected double **deltaMax**

deltaMin

protected double **deltaMin**

durationMeasure

protected *DurationMeasure* **durationMeasure**

evaluator

protected *SolutionListEvaluator<DoubleSolution>* **evaluator**

iterations

protected int **iterations**

leaders

public `List<ArchiveWithReferencePoint<DoubleSolution>>` **leaders**

maxIterations

protected int **maxIterations**

measureManager

protected `SimpleMeasureManager` **measureManager**

referencePoints

protected `List<List<Double>>` **referencePoints**

solutionListMeasure

protected `BasicMeasure<List<DoubleSolution>>` **solutionListMeasure**

swarmSize

protected int **swarmSize**

Constructors**SMPSORP**

```
public SMPSORP (DoubleProblem problem, int swarmSize, List<ArchiveWithReferencePoint<DoubleSolution>>
    leaders, List<List<Double>> referencePoints, MutationOperator<DoubleSolution> mu-
    tationOperator, int maxIterations, double r1Min, double r1Max, double r2Min, double
    r2Max, double c1Min, double c1Max, double c2Min, double c2Max, double weightMin,
    double weightMax, double changeVelocity1, double changeVelocity2, SolutionListEvalua-
    tor<DoubleSolution> evaluator)
```

Constructor

Methods**changeReferencePoints**

```
public synchronized void changeReferencePoints (List<List<Double>> referencePoints)
```

createInitialSwarm

```
protected List<DoubleSolution> createInitialSwarm ()
```

evaluateSwarm

```
protected List<DoubleSolution> evaluateSwarm (List<DoubleSolution> swarm)
```

getDescription

```
public String getDescription ()
```

getMeasureManager

```
public MeasureManager getMeasureManager ()
```

getName

```
public String getName ()
```

getResult

```
public List<DoubleSolution> getResult ()
```

initProgress

```
protected void initProgress ()
```

initializeLeader

```
protected void initializeLeader (List<DoubleSolution> swarm)
```

initializeParticlesMemory

```
protected void initializeParticlesMemory (List<DoubleSolution> swarm)
```

initializeVelocity

```
protected void initializeVelocity (List<DoubleSolution> swarm)
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

perturbation

```
protected void perturbation (List<DoubleSolution> swarm)
```

removeDominatedSolutionsInArchives

public void **removeDominatedSolutionsInArchives** ()

selectGlobalBest

protected *DoubleSolution* **selectGlobalBest** ()

updateLeaders

protected void **updateLeaders** (*List<DoubleSolution>* swarm)

updateLeadersDensityEstimator

protected void **updateLeadersDensityEstimator** ()

updateParticlesMemory

protected void **updateParticlesMemory** (*List<DoubleSolution>* swarm)

updatePosition

protected void **updatePosition** (*List<DoubleSolution>* swarm)

updateProgress

protected void **updateProgress** ()

updateVelocity

protected void **updateVelocity** (*List<DoubleSolution>* swarm)

2.32.7 SMPSOhv2IT

public class **SMPSOhv2IT**

Methods

setup

public void **setup** ()

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue()
```

2.32.8 SMPSOhvIT

```
public class SMPSOhvIT
```

Methods**setup**

```
public void setup()
```

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue()
```

2.33 org.uma.jmetal.algorithm.multiobjective.smsemoa**2.33.1 SMSEMOA**

```
public class SMSEMOA<S> extends Solution<?> extends AbstractGeneticAlgorithm<S, List<S>>
```

Author Antonio J. Nebro

Fields**dominanceComparator**

```
protected Comparator<S> dominanceComparator
```

evaluations

```
protected int evaluations
```

maxEvaluations

protected final int **maxEvaluations**

offset

protected final double **offset**

Constructors

SMSEMOA

```
public SMSEMOA (Problem<S> problem, int maxEvaluations, int populationSize, double offset, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, Comparator<S> dominanceComparator, Hypervolume<S> hypervolumeImplementation)
```

Constructor

Methods

computeRanking

```
protected Ranking<S> computeRanking (List<S> solutionList)
```

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected List<S> **replacement** (List<S> *population*, List<S> *offspringPopulation*)

reproduction

protected List<S> **reproduction** (List<S> *population*)

selection

protected List<S> **selection** (List<S> *population*)

updateProgress

protected void **updateProgress** ()

2.33.2 SMSEMOABuilder

public class **SMSEMOABuilder**<S extends Solution<?>> implements *AlgorithmBuilder*<*SMSEMOA*<S>>

Author Antonio J. Nebro

Fields**crossoverOperator**

protected *CrossoverOperator*<S> **crossoverOperator**

dominanceComparator

protected *Comparator*<S> **dominanceComparator**

hypervolumeImplementation

protected *Hypervolume*<S> **hypervolumeImplementation**

maxEvaluations

protected int **maxEvaluations**

mutationOperator

protected *MutationOperator*<S> **mutationOperator**

offset

protected double **offset**

populationSize

protected int **populationSize**

problem

protected *Problem*<S> **problem**

selectionOperator

protected *SelectionOperator*<*List*<S>, S> **selectionOperator**

Constructors

SMSEMOABuilder

public **SMSEMOABuilder** (*Problem*<S> *problem*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*)

Methods

build

public *SMSEMOA*<S> **build** ()

getCrossoverOperator

public *CrossoverOperator*<S> **getCrossoverOperator** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

getOffset

```
public double getOffset ()
```

getPopulationSize

```
public int getPopulationSize ()
```

getProblem

```
public Problem<S> getProblem ()
```

getSelectionOperator

```
public SelectionOperator<List<S>, S> getSelectionOperator ()
```

setCrossoverOperator

```
public SMSEMOABuilder<S> setCrossoverOperator (CrossoverOperator<S> crossover)
```

setDominanceComparator

```
public SMSEMOABuilder<S> setDominanceComparator (Comparator<S> dominanceComparator)
```

setHypervolumeImplementation

```
public SMSEMOABuilder<S> setHypervolumeImplementation (Hypervolume<S> hypervolumeImplementation)
```

setMaxEvaluations

```
public SMSEMOABuilder<S> setMaxEvaluations (int maxEvaluations)
```

setMutationOperator

```
public SMSEMOABuilder<S> setMutationOperator (MutationOperator<S> mutation)
```

setOffset

```
public SMSEMOABuilder<S> setOffset (double offset)
```

setPopulationSize

```
public SMSEMOABuilder<S> setPopulationSize (int populationSize)
```

setSelectionOperator

```
public SMSEMOABuilder<S> setSelectionOperator (SelectionOperator<List<S>, S> selection)
```

2.33.3 SMSEMOAIT

```
public class SMSEMOAIT
```

Fields

algorithm

```
Algorithm<List<DoubleSolution>> algorithm
```

Methods

shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue()
```

2.34 org.uma.jmetal.algorithm.multiobjective.spea2

2.34.1 SPEA2

```
public class SPEA2<S extends Solution<?>> extends AbstractGeneticAlgorithm<S, List<S>>
```

Author Juan J. Durillo

Fields

archive

```
protected List<S> archive
```

environmentalSelection

```
protected final EnvironmentalSelection<S> environmentalSelection
```

evaluator

```
protected final SolutionListEvaluator<S> evaluator
```

iterations

protected int **iterations**

maxIterations

protected final int **maxIterations**

strenghtRawFitness

protected final *StrengthRawFitness*<S> **strenghtRawFitness**

Constructors**SPEA2**

public **SPEA2** (*Problem*<S> *problem*, int *maxIterations*, int *populationSize*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*, *SelectionOperator*<List<S>, S> *selectionOperator*, *SolutionListEvaluator*<S> *evaluator*)

Methods**evaluatePopulation**

protected List<S> **evaluatePopulation** (List<S> *population*)

getDescription

public String **getDescription** ()

getName

public String **getName** ()

getResult

public List<S> **getResult** ()

initProgress

protected void **initProgress** ()

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected `List<S> replacement (List<S> population, List<S> offspringPopulation)`

reproduction

protected `List<S> reproduction (List<S> population)`

selection

protected `List<S> selection (List<S> population)`

updateProgress

protected void `updateProgress ()`

2.34.2 SPEA2Builder

public class **SPEA2Builder**<S extends Solution<?>> implements *AlgorithmBuilder*<*SPEA2*<S>>

Author Juan J. Durillo

Fields

crossoverOperator

protected *CrossoverOperator*<S> **crossoverOperator**

evaluator

protected *SolutionListEvaluator*<S> **evaluator**

maxIterations

protected int **maxIterations**

mutationOperator

protected *MutationOperator*<S> **mutationOperator**

populationSize

protected int **populationSize**

problem

protected final *Problem*<S> **problem**
 SPEA2Builder class

selectionOperator

protected *SelectionOperator*<*List*<S>, S> **selectionOperator**

Constructors**SPEA2Builder**

public **SPEA2Builder** (*Problem*<S> *problem*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*)
 SPEA2Builder constructor

Methods**build**

public *SPEA2*<S> **build** ()

getCrossoverOperator

public *CrossoverOperator*<S> **getCrossoverOperator** ()

getMaxIterations

public int **getMaxIterations** ()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

getPopulationSize

public int **getPopulationSize** ()

getProblem

public *Problem*<S> **getProblem** ()

getSelectionOperator

```
public SelectionOperator<List<S>, S> getSelectionOperator ()
```

getSolutionListEvaluator

```
public SolutionListEvaluator<S> getSolutionListEvaluator ()
```

setMaxIterations

```
public SPEA2Builder<S> setMaxIterations (int maxIterations)
```

setPopulationSize

```
public SPEA2Builder<S> setPopulationSize (int populationSize)
```

setSelectionOperator

```
public SPEA2Builder<S> setSelectionOperator (SelectionOperator<List<S>, S> selectionOperator)
```

setSolutionListEvaluator

```
public SPEA2Builder<S> setSolutionListEvaluator (SolutionListEvaluator<S> evaluator)
```

2.35 org.uma.jmetal.algorithm.multiobjective.spea2.util

2.35.1 EnvironmentalSelection

```
public class EnvironmentalSelection<S extends Solution<?>> implements SelectionOperator<List<S>, List<S>>
```

Author Juanjo Durillo

Parameters

- <S> –

Constructors**EnvironmentalSelection**

```
public EnvironmentalSelection (int solutionsToSelect)
```

Methods

execute

```
public List<S> execute (List<S> source2)
```

2.36 org.uma.jmetal.algorithm.multiobjective.wasfga

2.36.1 WASFGA

```
public class WASFGA<S extends Solution<?>> extends AbstractMOMBI<S> implements InteractiveAlgorithm<S, List<S>>
    Implementation of the preference based algorithm named WASF-GA on jMetal5.0
```

Author Juanjo Durillo This algorithm is described in the paper: A.B. Ruiz, R. Saborido, M. Luque “A Preference-based Evolutionary Algorithm for Multiobjective Optimization: The Weighting Achievement Scalarizing Function Genetic Algorithm”. Journal of Global Optimization. May 2015, Volume 62, Issue 1, pp 101-129 DOI = { 10.1007/s10898-014-0214-y }

Fields

epsilon

```
protected double epsilon
```

evaluations

```
protected int evaluations
```

maxEvaluations

```
protected int maxEvaluations
```

weights

```
protected double[][] weights
```

Constructors

WASFGA

```
public WASFGA (Problem<S> problem, int populationSize, int maxIterations, CrossoverOperator<S>
    crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>,
    S> selectionOperator, SolutionListEvaluator<S> evaluator, double epsilon, List<Double> ref-
    erencePoint, String weightVectorsFileName)
```

Constructor

Parameters

- **problem** – Problem to solve

WASFGA

```
public WASFGA (Problem<S> problem, int populationSize, int maxIterations, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, SolutionListEvaluator<S> evaluator, double epsilon, List<Double> referencePoint)
```

Constructor

Parameters

- **problem** – Problem to solve

Methods

addLastRankedSolutionsToPopulation

```
protected void addLastRankedSolutionsToPopulation (Ranking<S> ranking, int index, List<S> population)
```

addRankedSolutionsToPopulation

```
protected void addRankedSolutionsToPopulation (Ranking<S> ranking, int index, List<S> population)
```

computeRanking

```
protected Ranking<S> computeRanking (List<S> solutionList)
```

createUtilityFunction

```
public AbstractUtilityFunctionsSet<S> createUtilityFunction ()
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getNonDominatedSolutions

```
protected List<S> getNonDominatedSolutions (List<S> solutionList)
```


getPopulationSize

```
public int getPopulationSize ()
```

getResult

```
public List<S> getResult ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

selectBest

```
protected List<S> selectBest (Ranking<S> ranking)
```

specificMOEAComputations

```
public void specificMOEAComputations ()
```

updatePointOfInterest

```
public void updatePointOfInterest (List<Double> newPointOfInterest)
```

2.36.2 WASFGAIT

```
public class WASFGAIT
```

Methods**shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem**

```
public void shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem ()
```

shouldTheAlgorithmReturnAnExceptionIfIndicatingANonExistingWeightVectorFile

```
public void shouldTheAlgorithmReturnAnExceptionIfIndicatingANonExistingWeightVectorFile ()
```

shouldTheHypervolumeHaveAMinimumValue

```
public void shouldTheHypervolumeHaveAMinimumValue ()
```

2.36.3 WASFGAMEasures

public class **WASFGAMEasures**<S extends Solution<?>> extends *WASFGA*<S> implements *Measurable*
Implementation of the preference based algorithm named WASF-GA on jMetal5.0

Author Jorge Rodriguez

Fields

durationMeasure

protected *DurationMeasure* **durationMeasure**

iterations

protected *CountingMeasure* **iterations**

measureManager

protected *SimpleMeasureManager* **measureManager**

solutionListMeasure

protected *BasicMeasure*<List<S>> **solutionListMeasure**

Constructors

WASFGAMEasures

public **WASFGAMEasures** (*Problem*<S> *problem*, int *populationSize*, int *maxIterations*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*, *SelectionOperator*<List<S>, S> *selectionOperator*, *SolutionListEvaluator*<S> *evaluator*, double *epsilon*, List<Double> *referencePoint*, String *weightVectorsFileName*)

Constructor

Parameters

- **problem** – Problem to solve

WASFGAMEasures

public **WASFGAMEasures** (*Problem*<S> *problem*, int *populationSize*, int *maxIterations*, *CrossoverOperator*<S> *crossoverOperator*, *MutationOperator*<S> *mutationOperator*, *SelectionOperator*<List<S>, S> *selectionOperator*, *SolutionListEvaluator*<S> *evaluator*, double *epsilon*, List<Double> *referencePoint*)

Constructor

Parameters

- **problem** – Problem to solve

Methods

getDescription

```
public String getDescription ()
```

getMeasureManager

```
public MeasureManager getMeasureManager ()
```

getName

```
public String getName ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

run

```
public void run ()
```

updateProgress

```
protected void updateProgress ()
```

2.37 org.uma.jmetal.algorithm.multiobjective.wasfga.util

2.37.1 WASFGARanking

```
public class WASFGARanking<S extends Solution<?>> extends GenericSolutionAttribute<S, Integer> implements Ranking<S>
```

Author Rubén Saborido Implementation of the ranking procedure for the preference based algorithm named WASF-GA on jMetal5.0 It classifies solutions into different fronts. If the problem contains constraints, after feasible solutions it classifies the unfeasible solutions into fronts: - Each unfeasible solutions goes into a different front. - Unfeasible solutions with lower number of violated constraints are preferred. - If two solutions have equal number of violated constraints it compares the overall constraint values. - If two solutions have equal overall constraint values it compares de values of the utility function.

Constructors

WASFGARanking

```
public WASFGARanking (AbstractUtilityFunctionsSet<S> utilityFunctions)
```

Methods

computeRanking

```
public Ranking<S> computeRanking (List<S> population)
```

getNumberOfSubfronts

```
public int getNumberOfSubfronts ()
```

getSubfront

```
public List<S> getSubfront (int rank)
```

getUtilityFunctions

```
public AbstractUtilityFunctionsSet<S> getUtilityFunctions ()
```

rankUnfeasibleSolutions

```
protected int[] rankUnfeasibleSolutions (List<S> population)  
    Obtain the rank of each solution in a list of unfeasible solutions
```

Parameters

- **population** – List of unfeasible solutions

Returns The rank of each unfeasible solutions

2.37.2 WeightVectors

```
public class WeightVectors
```

Author Rubén Saborido Infantes This class offers different methods to manipulate weight vectors.

Methods

initializeUniformlyInTwoDimensions

```
public static double[][] initializeUniformlyInTwoDimensions (double epsilon, int num-  
                                                             berOfWeights)  
    Generate uniform weight vectors in two dimension
```

Parameters

- **epsilon** – Distance between each component of the weight vector
- **numberOfWeights** – Number of weight vectors to generate

Returns A set of weight vectors

invert

public static double[][] **invert** (double[][] *weights*, boolean *normalize*)

Calculate the inverse of a set of weight vectors

Parameters

- **weights** – A set of weight vectors
- **normalize** – True if the weights should be normalize by the sum of the components

Returns A set of weight vectors

readFromFile

public static double[][] **readFromFile** (String *filePath*)

Read a set of weight vector from a file

Parameters

- **filePath** – A file containing the weight vectors

Returns A set of weight vectors

readFromResourcesInJMetal

public static double[][] **readFromResourcesInJMetal** (String *filePath*)

Read a set of weight vector from a file in the resources folder in jMetal

Parameters

- **filePath** – The name of file in the resources folder of jMetal

Returns A set of weight vectors

validate

public static boolean **validate** (double[][] *weights*, int *numberOfComponents*)

Validate if the number of components of all weight vectors has the expected dimensionality.

Parameters

- **weights** – Weight vectors to validate
- **numberOfComponents** – Number of components each weight vector must have

Returns True if the weight vectors are correct, False if the weight vectors are incorrect

2.38 org.uma.jmetal.algorithm.singleobjective.coralreefsoptimization

2.38.1 CoralReefsOptimization

public class **CoralReefsOptimization**<S> extends *AbstractCoralReefsOptimization*<S, List<S>>

Author Inacio Medeiros

Constructors

CoralReefsOptimization

public **CoralReefsOptimization** (*Problem*<S> problem, int maxEvaluations, *Comparator*<S> comparator, *SelectionOperator*<List<S>, S> selectionOperator, *CrossoverOperator*<S> crossoverOperator, *MutationOperator*<S> mutationOperator, int n, int m, double rho, double fbs, double fa, double pd, int attemptsToSettle)

Methods

asexualReproduction

protected List<S> **asexualReproduction** (List<S> brooders)

createInitialPopulation

protected List<S> **createInitialPopulation** ()

depredation

protected List<S> **depredation** (List<S> population, List<*Coordinate*> coordinates)

evaluatePopulation

protected List<S> **evaluatePopulation** (List<S> population)

generateCoordinates

protected List<*Coordinate*> **generateCoordinates** ()

getDescription

public String **getDescription** ()

getName

```
public String getName ()
```

getResult

```
public List<S> getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

larvaeSettlementPhase

```
protected List<S> larvaeSettlementPhase (List<S> larvae, List<S> population, List<Coordinate> co-ordinates)
```

selectBroadcastSpawners

```
protected List<S> selectBroadcastSpawners (List<S> population)
```

sexualReproduction

```
protected List<S> sexualReproduction (List<S> broadcastSpawners)
```

updateProgress

```
protected void updateProgress ()
```

2.38.2 CoralReefsOptimizationBuilder

```
public class CoralReefsOptimizationBuilder<S> extends Solution<?>> implements AlgorithmBuilder<CoralReefsOptimization>
```

Author Inacio Medeiros

Constructors

CoralReefsOptimizationBuilder

```
public CoralReefsOptimizationBuilder (Problem<S> problem, SelectionOperator<List<S>, S> se-  
lectionOperator, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator)  
    CoralReefsOptimizationBuilder constructor
```

Methods

build

```
public CoralReefsOptimization<S> build ()
```

getAttemptsToSettle

```
public int getAttemptsToSettle ()
```

getComparator

```
public Comparator<S> getComparator ()
```

getCrossoverOperator

```
public CrossoverOperator<S> getCrossoverOperator ()
```

getFa

```
public double getFa ()
```

getFbr

```
public double getFbr ()
```

getFbs

```
public double getFbs ()
```

getFd

```
public double getFd ()
```


getM

public int **getM**()

getMaxEvaluations

public int **getMaxEvaluations**()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator**()

getN

public int **getN**()

getPd

public double **getPd**()

getProblem

public *Problem*<S> **getProblem**()

getRho

public double **getRho**()

getSelectionOperator

public *SelectionOperator*<List<S>, S> **getSelectionOperator**()

setAttemptsToSettle

public *CoralReefsOptimizationBuilder*<S> **setAttemptsToSettle** (int *attemptsToSettle*)

setComparator

public *CoralReefsOptimizationBuilder*<S> **setComparator** (*Comparator*<S> *comparator*)

setFa

public *CoralReefsOptimizationBuilder*<S> **setFa** (double *fa*)

setFbr

public *CoralReefsOptimizationBuilder*<S> **setFbr** (double *fbr*)

setFbs

public *CoralReefsOptimizationBuilder*<S> **setFbs** (double *fbs*)

setFd

public *CoralReefsOptimizationBuilder*<S> **setFd** (double *fd*)

setM

public *CoralReefsOptimizationBuilder*<S> **setM** (int *m*)

setMaxEvaluations

public *CoralReefsOptimizationBuilder*<S> **setMaxEvaluations** (int *maxEvaluations*)

setN

public *CoralReefsOptimizationBuilder*<S> **setN** (int *n*)

setPd

public *CoralReefsOptimizationBuilder*<S> **setPd** (double *pd*)

setRho

public *CoralReefsOptimizationBuilder*<S> **setRho** (double *rho*)

2.39 org.uma.jmetal.algorithm.singleobjective.differentialevolution

2.39.1 DifferentialEvolution

public class **DifferentialEvolution** extends *AbstractDifferentialEvolution*<*DoubleSolution*>

This class implements a differential evolution algorithm.

Author Antonio J. Nebro

Constructors

DifferentialEvolution

```
public DifferentialEvolution (DoubleProblem problem, int maxEvaluations, int populationSize, DifferentialEvolutionCrossover crossoverOperator, DifferentialEvolutionSelection selectionOperator, SolutionListEvaluator<DoubleSolution> evaluator)
```

Constructor

Parameters

- **problem** – Problem to solve
- **maxEvaluations** – Maximum number of evaluations to perform
- **populationSize** –
- **crossoverOperator** –
- **selectionOperator** –
- **evaluator** –

Methods

createInitialPopulation

```
protected List<DoubleSolution> createInitialPopulation ()
```

evaluatePopulation

```
protected List<DoubleSolution> evaluatePopulation (List<DoubleSolution> population)
```

getDescription

```
public String getDescription ()
```

getEvaluations

```
public int getEvaluations ()
```

getName

```
public String getName ()
```

getResult

```
public DoubleSolution getResult ()  
    Returns the best individual
```

initProgress

protected void **initProgress** ()

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected *List<DoubleSolution>* **replacement** (*List<DoubleSolution>* population, *List<DoubleSolution>* offspringPopulation)

reproduction

protected *List<DoubleSolution>* **reproduction** (*List<DoubleSolution>* matingPopulation)

selection

protected *List<DoubleSolution>* **selection** (*List<DoubleSolution>* population)

setEvaluations

public void **setEvaluations** (int evaluations)

updateProgress

protected void **updateProgress** ()

2.39.2 DifferentialEvolutionBuilder

public class **DifferentialEvolutionBuilder**
DifferentialEvolutionBuilder class

Author Antonio J. Nebro

Constructors

DifferentialEvolutionBuilder

public **DifferentialEvolutionBuilder** (*DoubleProblem* problem)

Methods

build

public *DifferentialEvolution* **build** ()

getCrossoverOperator

public *DifferentialEvolutionCrossover* **getCrossoverOperator** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getPopulationSize

public int **getPopulationSize** ()

getProblem

public *DoubleProblem* **getProblem** ()

getSelectionOperator

public *DifferentialEvolutionSelection* **getSelectionOperator** ()

getSolutionListEvaluator

public *SolutionListEvaluator*<*DoubleSolution*> **getSolutionListEvaluator** ()

setCrossover

public *DifferentialEvolutionBuilder* **setCrossover** (*DifferentialEvolutionCrossover* crossover)

setMaxEvaluations

public *DifferentialEvolutionBuilder* **setMaxEvaluations** (int maxEvaluations)

setPopulationSize

public *DifferentialEvolutionBuilder* **setPopulationSize** (int populationSize)

setSelection

```
public DifferentialEvolutionBuilder setSelection (DifferentialEvolutionSelection selection)
```

setSolutionListEvaluator

```
public DifferentialEvolutionBuilder setSolutionListEvaluator (SolutionListEvaluator<DoubleSolution>  
                                                           evaluator)
```

2.39.3 DifferentialEvolutionBuilderTest

```
public class DifferentialEvolutionBuilderTest  
    Created by ajnebro on 25/11/14.
```

Methods

buildAlgorithm

```
public void buildAlgorithm ()
```

cleanup

```
public void cleanup ()
```

getProblem

```
public void getProblem ()
```

setNegativeMaxNumberOfEvaluations

```
public void setNegativeMaxNumberOfEvaluations ()
```

setNegativePopulationSize

```
public void setNegativePopulationSize ()
```

setNewCrossoverOperator

```
public void setNewCrossoverOperator ()
```

setNewEvaluator

```
public void setNewEvaluator ()
```

setNewSelectionOperator

```
public void setNewSelectionOperator ()
```

setPositiveMaxNumberOfEvaluations

```
public void setPositiveMaxNumberOfEvaluations ()
```

setValidPopulationSize

```
public void setValidPopulationSize ()
```

startup

```
public void startup ()
```

testDefaultConfiguration

```
public void testDefaultConfiguration ()
```

2.39.4 DifferentialEvolutionTestIT

```
public class DifferentialEvolutionTestIT  
    Created by Antonio J. Nebro on 25/11/14.
```

Methods**shouldCreateInitialPopulationWhenPopulationSizelsBiggerThanZero**

```
public void shouldCreateInitialPopulationWhenPopulationSizeIsBiggerThanZero ()
```

shouldCreateInitialPopulationWhenPopulationSizelsZero

```
public void shouldCreateInitialPopulationWhenPopulationSizeIsZero ()
```

shouldEvaluatePopulation

```
public void shouldEvaluatePopulation ()
```

shouldGetEvaluations

```
public void shouldGetEvaluations ()
```

shouldGetResultReturnsThenReturnTheBestIndividual

```
public void shouldGetResultReturnsThenReturnTheBestIndividual ()
```

shouldInitProgress

```
public void shouldInitProgress ()
```

shouldIsStoppingConditionReachedWhenEvaluationsBiggerThanMaxEvaluations

```
public void shouldIsStoppingConditionReachedWhenEvaluationsBiggerThanMaxEvaluations ()
```

shouldIsStoppingConditionReachedWhenEvaluationsEqualToMaxEvaluations

```
public void shouldIsStoppingConditionReachedWhenEvaluationsEqualToMaxEvaluations ()
```

shouldIsStoppingConditionReachedWhenEvaluationsLesserThanMaxEvaluations

```
public void shouldIsStoppingConditionReachedWhenEvaluationsLesserThanMaxEvaluations ()
```

shouldReplacemen2t

```
public void shouldReplacemen2t ()
```

shouldReproduction

```
public void shouldReproduction ()
```

shouldSelection

```
public void shouldSelection ()
```

shouldSetEvaluations

```
public void shouldSetEvaluations ()
```

shouldUpdateProgressWhenAnyIteration

```
public void shouldUpdateProgressWhenAnyIteration ()
```

shouldUpdateProgressWhenFirstIteration

```
public void shouldUpdateProgressWhenFirstIteration ()
```


startup

```
public void startup ()
```

2.40 org.uma.jmetal.algorithm.singleobjective.evolutionstrategy

2.40.1 CovarianceMatrixAdaptationEvolutionStrategy

```
public class CovarianceMatrixAdaptationEvolutionStrategy extends AbstractEvolutionStrategy<DoubleSolution, DoubleSolution>  
    Class implementing the CMA-ES algorithm
```

Methods

createInitialPopulation

```
protected List<DoubleSolution> createInitialPopulation ()
```

evaluatePopulation

```
protected List<DoubleSolution> evaluatePopulation (List<DoubleSolution> population)
```

getDescription

```
public String getDescription ()
```

getLambda

```
public int getLambda ()
```

getMaxEvaluations

```
public int getMaxEvaluations ()
```

getName

```
public String getName ()
```

getResult

```
public DoubleSolution getResult ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected List<*DoubleSolution*> **replacement** (List<*DoubleSolution*> *population*, List<*DoubleSolution*> *offspringPopulation*)

reproduction

protected List<*DoubleSolution*> **reproduction** (List<*DoubleSolution*> *population*)

selection

protected List<*DoubleSolution*> **selection** (List<*DoubleSolution*> *population*)

updateProgress

protected void **updateProgress** ()

2.40.2 CovarianceMatrixAdaptationEvolutionStrategy.Builder

public static class **Builder**
Builder class

Constructors

Builder

public **Builder** (*DoubleProblem* *problem*)

Methods

build

public *CovarianceMatrixAdaptationEvolutionStrategy* **build** ()

setLambda

public *Builder* **setLambda** (int *lambda*)

setMaxEvaluations

public *Builder* **setMaxEvaluations** (int *maxEvaluations*)

setSigma

```
public Builder setSigma (double sigma)
```

setTypicalX

```
public Builder setTypicalX (double[] typicalX)
```

2.40.3 ElitistEvolutionStrategy

```
public class ElitistEvolutionStrategy<S extends Solution<?>> extends AbstractEvolutionStrategy<S, S>
    Class implementing a (mu + lambda) Evolution Strategy (lambda must be divisible by mu)
```

Author Antonio J. Nebro

Constructors**ElitistEvolutionStrategy**

```
public ElitistEvolutionStrategy (Problem<S> problem, int mu, int lambda, int maxEvaluations, MutationOperator<S> mutation)
```

Constructor

Methods**createInitialPopulation**

```
protected List<S> createInitialPopulation ()
```

evaluatePopulation

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public S getResult ()
```

initProgress

protected void **initProgress** ()

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected [List<S>](#) **replacement** ([List<S>](#) *population*, [List<S>](#) *offspringPopulation*)

reproduction

protected [List<S>](#) **reproduction** ([List<S>](#) *population*)

selection

protected [List<S>](#) **selection** ([List<S>](#) *population*)

updateProgress

protected void **updateProgress** ()

2.40.4 EvolutionStrategyBuilder

public class **EvolutionStrategyBuilder**<S extends [Solution](#)<?>> implements [AlgorithmBuilder](#)<[Algorithm](#)<S>>
Class implementing a (mu , lambda) Evolution Strategy (lambda must be divisible by mu)

Author Antonio J. Nebro

Constructors

EvolutionStrategyBuilder

public **EvolutionStrategyBuilder** ([Problem](#)<S> *problem*, [MutationOperator](#)<S> *mutationOperator*,
[EvolutionStrategyVariant](#) *variant*)

Methods

build

public [Algorithm](#)<S> **build** ()

getLambda

```
public int getLambda ()
```

getMaxEvaluations

```
public int getMaxEvaluations ()
```

getMu

```
public int getMu ()
```

getMutation

```
public MutationOperator<S> getMutation ()
```

setLambda

```
public EvolutionStrategyBuilder<S> setLambda (int lambda)
```

setMaxEvaluations

```
public EvolutionStrategyBuilder<S> setMaxEvaluations (int maxEvaluations)
```

setMu

```
public EvolutionStrategyBuilder<S> setMu (int mu)
```

2.40.5 EvolutionStrategyBuilder.EvolutionStrategyVariant

```
public enum EvolutionStrategyVariant
```

Enum Constants**ELITIST**

```
public static final EvolutionStrategyBuilder.EvolutionStrategyVariant ELITIST
```

NON_ELITIST

```
public static final EvolutionStrategyBuilder.EvolutionStrategyVariant NON_ELITIST
```

2.40.6 NonElitistEvolutionStrategy

public class **NonElitistEvolutionStrategy**<S extends Solution<?>> extends *AbstractEvolutionStrategy*<S, S>
Class implementing a (mu + lambda) Evolution Strategy (lambda must be divisible by mu)

Author Antonio J. Nebro

Constructors

NonElitistEvolutionStrategy

public **NonElitistEvolutionStrategy** (*Problem*<S> *problem*, int *mu*, int *lambda*, int *maxEvaluations*,
MutationOperator<S> *mutation*)
Constructor

Methods

createInitialPopulation

protected *List*<S> **createInitialPopulation** ()

evaluatePopulation

protected *List*<S> **evaluatePopulation** (*List*<S> *population*)

getDescription

public *String* **getDescription** ()

getName

public *String* **getName** ()

getResult

public S **getResult** ()

initProgress

protected void **initProgress** ()

isStoppingConditionReached

protected boolean **isStoppingConditionReached** ()

replacement

protected `List<S> replacement (List<S> population, List<S> offspringPopulation)`

reproduction

protected `List<S> reproduction (List<S> population)`

selection

protected `List<S> selection (List<S> population)`

updateProgress

protected void `updateProgress ()`

2.41 org.uma.jmetal.algorithm.singleobjective.evolutionstrategy.util**2.41.1 CMAESUtils**

public class `CMAESUtils`

Methods**checkEigenSystem**

public static int `checkEigenSystem (int n, double[][] c, double[] diag, double[][] q)`

norm

public static double `norm (double[] vector)`

tql2

public static void `tql2 (int n, double[] d, double[] e, double[][] v)`

tred2

public static void `tred2 (int n, double[][] v, double[] d, double[] e)`

2.42 org.uma.jmetal.algorithm.singleobjective.geneticalgorithm

2.42.1 GenerationalGeneticAlgorithm

public class **GenerationalGeneticAlgorithm**<S extends Solution<?>> extends *AbstractGeneticAlgorithm*<S, S>

Author Antonio J. Nebro

Constructors

GenerationalGeneticAlgorithm

```
public GenerationalGeneticAlgorithm(Problem<S> problem, int maxEvaluations, int populationSize, CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator, SelectionOperator<List<S>, S> selectionOperator, SolutionListEvaluator<S> evaluator)
```

Constructor

Methods

evaluatePopulation

```
protected List<S> evaluatePopulation(List<S> population)
```

getDescription

```
public String getDescription()
```

getName

```
public String getName()
```

getResult

```
public S getResult()
```

initProgress

```
public void initProgress()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached()
```


replacement

protected `List<S>` **replacement** (`List<S>` *population*, `List<S>` *offspringPopulation*)

updateProgress

public void **updateProgress** ()

2.42.2 GenerationalGeneticAlgorithmTestIT

public class **GenerationalGeneticAlgorithmTestIT**
Created by ajnebro on 27/10/15.

Methods

shouldTheAlgorithmReturnTheCorrectSolutionWhenSolvingProblemOneMax

public void **shouldTheAlgorithmReturnTheCorrectSolutionWhenSolvingProblemOneMax** ()

2.42.3 GeneticAlgorithmBuilder

public class **GeneticAlgorithmBuilder**<S extends `Solution<?>`>
Created by ajnebro on 10/12/14.

Constructors

GeneticAlgorithmBuilder

public **GeneticAlgorithmBuilder** (`Problem<S>` *problem*, `CrossoverOperator<S>` *crossoverOperator*,
`MutationOperator<S>` *mutationOperator*)
Builder constructor

Methods

build

public `Algorithm<S>` **build** ()

getCrossoverOperator

public `CrossoverOperator<S>` **getCrossoverOperator** ()

getEvaluator

public `SolutionListEvaluator<S>` **getEvaluator** ()

getMaxEvaluations

public int **getMaxEvaluations** ()

getMutationOperator

public *MutationOperator*<S> **getMutationOperator** ()

getPopulationSize

public int **getPopulationSize** ()

getProblem

public *Problem*<S> **getProblem** ()

getSelectionOperator

public *SelectionOperator*<List<S>, S> **getSelectionOperator** ()

getVariant

public *GeneticAlgorithmVariant* **getVariant** ()

setMaxEvaluations

public *GeneticAlgorithmBuilder*<S> **setMaxEvaluations** (int *maxEvaluations*)

setPopulationSize

public *GeneticAlgorithmBuilder*<S> **setPopulationSize** (int *populationSize*)

setSelectionOperator

public *GeneticAlgorithmBuilder*<S> **setSelectionOperator** (*SelectionOperator*<List<S>, S> *selectionOperator*)

setSolutionListEvaluator

public *GeneticAlgorithmBuilder*<S> **setSolutionListEvaluator** (*SolutionListEvaluator*<S> *evaluator*)

setVariant

```
public GeneticAlgorithmBuilder<S> setVariant (GeneticAlgorithmVariant variant)
```

2.42.4 GeneticAlgorithmBuilder.GeneticAlgorithmVariant

```
public enum GeneticAlgorithmVariant
```

Enum Constants**GENERATIONAL**

```
public static final GeneticAlgorithmBuilder.GeneticAlgorithmVariant GENERATIONAL
```

STEADY_STATE

```
public static final GeneticAlgorithmBuilder.GeneticAlgorithmVariant STEADY_STATE
```

2.42.5 SteadyStateGeneticAlgorithm

```
public class SteadyStateGeneticAlgorithm<S extends Solution<?>> extends AbstractGeneticAlgorithm<S, S>
```

Author Antonio J. Nebro

Constructors**SteadyStateGeneticAlgorithm**

```
public SteadyStateGeneticAlgorithm (Problem<S> problem, int maxEvaluations, int populationSize,  
                                     CrossoverOperator<S> crossoverOperator, MutationOperator<S> mutationOperator,  
                                     SelectionOperator<List<S>, S> selectionOperator)
```

Constructor

Methods**evaluatePopulation**

```
protected List<S> evaluatePopulation (List<S> population)
```

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

getResult

```
public S getResult ()
```

initProgress

```
public void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

replacement

```
protected List<S> replacement (List<S> population, List<S> offspringPopulation)
```

reproduction

```
protected List<S> reproduction (List<S> matingPopulation)
```

selection

```
protected List<S> selection (List<S> population)
```

updateProgress

```
public void updateProgress ()
```

2.42.6 SteadyStateGeneticAlgorithmTestIT

```
public class SteadyStateGeneticAlgorithmTestIT  
    Created by ajnebro on 27/10/15.
```

Methods

shouldTheAlgorithmReturnTheCorrectSolutionWhenSolvingProblemOneMax

```
public void shouldTheAlgorithmReturnTheCorrectSolutionWhenSolvingProblemOneMax ()
```

2.43 org.uma.jmetal.algorithm.singleobjective.particleswarmoptimization

2.43.1 StandardPSO2007

public class **StandardPSO2007** extends *AbstractParticleSwarmOptimization*<*DoubleSolution*, *DoubleSolution*>
 Class implementing a Standard PSO 2007 algorithm.

Author Antonio J. Nebro

Constructors

StandardPSO2007

public **StandardPSO2007** (*DoubleProblem* problem, int objectiveId, int swarmSize, int maxIterations,
 int numberOfParticlesToInform, *SolutionListEvaluator*<*DoubleSolution*> evaluator)

Constructor

Parameters

- **problem** –
- **objectiveId** – This field indicates which objective, in the case of a multi-objective problem, is selected to be optimized.
- **swarmSize** –
- **maxIterations** –
- **numberOfParticlesToInform** –
- **evaluator** –

StandardPSO2007

public **StandardPSO2007** (*DoubleProblem* problem, int swarmSize, int maxIterations, int numberOfParti-
 clesToInform, *SolutionListEvaluator*<*DoubleSolution*> evaluator)

Constructor

Parameters

- **problem** –
- **swarmSize** –
- **maxIterations** –
- **numberOfParticlesToInform** –
- **evaluator** –

Methods

createInitialSwarm

public *List*<*DoubleSolution*> **createInitialSwarm**()

evaluateSwarm

```
public List<DoubleSolution> evaluateSwarm (List<DoubleSolution> swarm)
```

getDescription

```
public String getDescription ()
```

getLocalBest

```
public DoubleSolution[] getLocalBest ()
```

getName

```
public String getName ()
```

getResult

```
public DoubleSolution getResult ()
```

getSwarmSpeedMatrix

```
public double[][] getSwarmSpeedMatrix ()
```

initProgress

```
public void initProgress ()
```

initializeLeader

```
public void initializeLeader (List<DoubleSolution> swarm)
```

initializeParticlesMemory

```
public void initializeParticlesMemory (List<DoubleSolution> swarm)
```

initializeVelocity

```
public void initializeVelocity (List<DoubleSolution> swarm)
```

isStoppingConditionReached

```
public boolean isStoppingConditionReached ()
```

perturbation

```
public void perturbation (List<DoubleSolution> swarm)
```

updateLeaders

```
public void updateLeaders (List<DoubleSolution> swarm)
```

updateParticlesMemory

```
public void updateParticlesMemory (List<DoubleSolution> swarm)
```

updatePosition

```
public void updatePosition (List<DoubleSolution> swarm)
```

updateProgress

```
public void updateProgress ()
```

updateVelocity

```
public void updateVelocity (List<DoubleSolution> swarm)
```

2.43.2 StandardPSO2011

```
public class StandardPSO2011 extends AbstractParticleSwarmOptimization<DoubleSolution, DoubleSolution>
```

Class implementing a Standard PSO 2011 algorithm.

Author Antonio J. Nebro

Constructors**StandardPSO2011**

```
public StandardPSO2011 (DoubleProblem problem, int objectiveId, int swarmSize, int maxIterations,  
int numberOfParticlesToInform, SolutionListEvaluator<DoubleSolution> evaluator)
```

Constructor

Parameters

- **problem** –
- **objectiveId** – This field indicates which objective, in the case of a multi-objective problem, is selected to be optimized.
- **swarmSize** –
- **maxIterations** –

- `numberOfParticlesToInform` –
- `evaluator` –

StandardPSO2011

public **StandardPSO2011** (*DoubleProblem* problem, int *swarmSize*, int *maxIterations*, int *numberOfParticlesToInform*, *SolutionListEvaluator*<*DoubleSolution*> evaluator)

Constructor

Parameters

- `problem` –
- `swarmSize` –
- `maxIterations` –
- `numberOfParticlesToInform` –
- `evaluator` –

Methods

createInitialSwarm

public *List*<*DoubleSolution*> **createInitialSwarm** ()

evaluateSwarm

public *List*<*DoubleSolution*> **evaluateSwarm** (*List*<*DoubleSolution*> swarm)

getDescription

public *String* **getDescription** ()

getLocalBest

public *DoubleSolution*[] **getLocalBest** ()

getName

public *String* **getName** ()

getResult

public *DoubleSolution* **getResult** ()

getSwarmSpeedMatrix

```
public double[][] getSwarmSpeedMatrix ()
```

initProgress

```
public void initProgress ()
```

initializeLeader

```
public void initializeLeader (List<DoubleSolution> swarm)
```

initializeParticlesMemory

```
public void initializeParticlesMemory (List<DoubleSolution> swarm)
```

initializeVelocity

```
public void initializeVelocity (List<DoubleSolution> swarm)
```

isStoppingConditionReached

```
public boolean isStoppingConditionReached ()
```

perturbation

```
public void perturbation (List<DoubleSolution> swarm)
```

updateLeaders

```
public void updateLeaders (List<DoubleSolution> swarm)
```

updateParticlesMemory

```
public void updateParticlesMemory (List<DoubleSolution> swarm)
```

updatePosition

```
public void updatePosition (List<DoubleSolution> swarm)
```

updateProgress

```
public void updateProgress ()
```

updateVelocity

```
public void updateVelocity (List<DoubleSolution> swarm)
```

2.44 org.uma.jmetal.experiment

2.44.1 BinaryProblemsStudy

```
public class BinaryProblemsStudy
```

Example of experimental study based on solving two binary problems with four algorithms: NSGAI, SPEA2, MOCell, and MOCHC. This experiment assumes that the reference Pareto front are not known, so the must be produced. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Generate the reference Pareto fronts 4. Compute que quality indicators 5. Generate Latex tables reporting means and medians 6. Generate Latex tables with the result of applying the Wilcoxon Rank Sum Test 7. Generate Latex tables with the ranking obtained by applying the Friedman test 8. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<BinarySolution, List<BinarySolution>>> configureAlgorithmList (List<ExperimentProblem<prob-lem-List>
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*.

main

```
public static void main (String[] args)
```

2.44.2 ConstraintProblemsStudy

```
public class ConstraintProblemsStudy
```

Example of experimental study based on solving the unconstrained problems included in jMetal. This experiment assumes that the reference Pareto front are known and that, given a problem named P, there is a corresponding file called P.pf containing its corresponding Pareto front. If this is not the case, please refer to class *DTLZStudy* to see an example of how to explicitly indicate the name of those files. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Generate the reference Pareto fronts 4. Compute the quality indicators 5. Generate Latex tables reporting means and medians 6. Generate Latex tables with the result of applying the Wilcoxon Rank Sum Test 7. Generate Latex tables with the ranking obtained by applying the Friedman test 8. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-
                                                                    prob-
                                                                    lem-
                                                                    List>)
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*. The *ExperimentAlgorithm* has an optional tag component, that can be set as it is shown in this example, where four variants of a same algorithm are defined.

main

```
public static void main (String[] args)
```

2.44.3 DTLZStudy

```
public class DTLZStudy
```

Example of experimental study based on solving the problems (configured with 3 objectives) with the algorithms NSGAII, SPEA2, and SMPSO. This experiment assumes that the reference Pareto front are known and stored in files whose names are different from the default name expected for every problem. While the default would be “problem_name.pf” (e.g. DTLZ1.pf), the references are stored in files following the nomenclature “problem_name.3D.pf” (e.g. DTLZ1.3D.pf). This is indicated when creating the *ExperimentProblem* instance of each of the evaluated problems by using the method *changeReferenceFrontTo()*. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Compute the quality indicators 4. Generate Latex tables reporting means and medians 5. Generate R scripts to produce latex tables with the result of applying the Wilcoxon Rank Sum Test 6. Generate Latex tables with the ranking obtained by applying the Friedman test 7. Generate R scripts to obtain boxplots

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-
                                                                    prob-
                                                                    lem-
                                                                    List>)
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*.

main

```
public static void main (String[] args)
```

2.44.4 NSGAIIStudy

```
public class NSGAIIStudy
```

Example of experimental study based on solving the ZDT problems with four versions of NSGA-II, each of

them applying a different crossover probability (from 0.7 to 1.0). This experiment assumes that the reference Pareto front are known and that, given a problem named P, there is a corresponding file called P.pf containing its corresponding Pareto front. If this is not the case, please refer to class `DTLZStudy` to see an example of how to explicitly indicate the name of those files. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Compute the quality indicators 4. Generate Latex tables reporting means and medians 5. Generate Latex tables with the result of applying the Wilcoxon Rank Sum Test 6. Generate Latex tables with the ranking obtained by applying the Friedman test 7. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-  
                                                                    prob-  
                                                                    lem-  
                                                                    List>)
```

The algorithm list is composed of pairs `Algorithm` + `Problem` which form part of a `ExperimentAlgorithm`, which is a decorator for class `Algorithm`. The `ExperimentAlgorithm` has an optional tag component, that can be set as it is shown in this example, where four variants of a same algorithm are defined.

main

```
public static void main (String[] args)
```

2.44.5 NSGAIIStudy2

```
public class NSGAIIStudy2
```

Example of experimental study based on solving the ZDT problems with four versions of NSGA-II, each of them applying a different crossover probability (from 0.7 to 1.0). This experiment assumes that the reference Pareto front are not known, so the names of files containing them and the directory where they are located must be specified. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Generate the reference Pareto fronts 4. Compute the quality indicators 5. Generate Latex tables reporting means and medians 6. Generate Latex tables with the result of applying the Wilcoxon Rank Sum Test 7. Generate Latex tables with the ranking obtained by applying the Friedman test 8. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-  
                                                                    prob-  
                                                                    lem-  
                                                                    List>)
```

The algorithm list is composed of pairs `Algorithm` + `Problem` which form part of a `ExperimentAlgorithm`, which is a decorator for class `Algorithm`. The `ExperimentAlgorithm`

has an optional tag component, that can be set as it is shown in this example, where four variants of a same algorithm are defined.

main

```
public static void main (String[] args)
```

2.44.6 ZDTScalabilityStudy

```
public class ZDTScalabilityIStudy
```

Example of experimental study based on solving the ZDT1 problem but using five different number of variables. This can be interesting to study the behaviour of the algorithms when solving an scalable problem (in the number of variables). The used algorithms are NSGA-II, SPEA2 and SMPSO. This experiment assumes that the reference Pareto front of problem ZDT1 is known and that there is a file called ZDT1.pf containing it. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Generate the reference Pareto fronts 4. Compute the quality indicators 5. Generate Latex tables reporting means and medians 6. Generate Latex tables with the result of applying the Wilcoxon Rank Sum Test 7. Generate Latex tables with the ranking obtained by applying the Friedman test 8. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem<DoubleSolution>>> prob-lem-List)
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*. The *ExperimentAlgorithm* has an optional tag component, that can be set as it is shown in this example, where four variants of a same algorithm are defined.

main

```
public static void main (String[] args)
```

2.44.7 ZDTScalabilityStudy2

```
public class ZDTScalabilityIStudy2
```

Example of experimental study based on solving the ZDT1 problem but using five different number of variables. This can be interesting to study the behaviour of the algorithms when solving an scalable problem (in the number of variables). The used algorithms are NSGA-II, SPEA2 and SMPSO. This experiment assumes that the reference Pareto front of problem ZDT1 is not known, so a reference front must be obtained. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Generate the reference Pareto sets and Pareto fronts 4. Compute the quality indicators 5. Generate Latex tables reporting means and medians 6. Generate Latex tables with the result

of applying the Wilcoxon Rank Sum Test 7. Generate Latex tables with the ranking obtained by applying the Friedman test 8. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-  
                                                                    prob-  
                                                                    lem-  
                                                                    List>
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*. The *ExperimentAlgorithm* has an optional tag component, that can be set as it is shown in this example, where four variants of a same algorithm are defined.

main

```
public static void main (String[] args)
```

2.44.8 ZDTStudy

```
public class ZDTStudy
```

Example of experimental study based on solving the ZDT problems with the algorithms NSGAII, MOEA/D, and SMPSO This experiment assumes that the reference Pareto front are known and that, given a problem named P, there is a corresponding file called P.pf containing its corresponding Pareto front. If this is not the case, please refer to class *DTLZStudy* to see an example of how to explicitly indicate the name of those files. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Compute que quality indicators 4. Generate Latex tables reporting means and medians 5. Generate R scripts to produce latex tables with the result of applying the Wilcoxon Rank Sum Test 6. Generate Latex tables with the ranking obtained by applying the Friedman test 7. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-  
                                                                    prob-  
                                                                    lem-  
                                                                    List>
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*.

main

```
public static void main (String[] args)
```

2.44.9 ZDTStudy2

public class **ZDTStudy2**

Example of experimental study based on solving the ZDT problems with algorithms NSGAI, MOEA/D, and SMPSO. This experiment assumes that the reference Pareto front are not known, so the names of files containing them and the directory where they are located must be specified. Six quality indicators are used for performance assessment. The steps to carry out the experiment are: 1. Configure the experiment 2. Execute the algorithms 3. Generate the reference Pareto fronts 4. Compute the quality indicators 5. Generate LaTeX tables reporting means and medians 6. Generate LaTeX tables with the result of applying the Wilcoxon Rank Sum Test 7. Generate R scripts to obtain boxplots

Author Antonio J. Nebro

Methods

configureAlgorithmList

```
static List<ExperimentAlgorithm<DoubleSolution, List<DoubleSolution>>> configureAlgorithmList (List<ExperimentProblem-
                                                                    prob-
                                                                    lem-
                                                                    List>)
```

The algorithm list is composed of pairs *Algorithm* + *Problem* which form part of a *ExperimentAlgorithm*, which is a decorator for class *Algorithm*.

main

```
public static void main (String[] args)
```

2.45 org.uma.jmetal.measure

2.45.1 Measurable

public interface **Measurable**

A *Measurable* entity is an entity which provides one or several *Measures*. To keep it simple, these *Measures* are provided through a *MeasureManager*.

Author Created by Antonio J. Nebro on 21/10/14 based on the ideas of Matthieu Vergne

Methods

getMeasureManager

```
public MeasureManager getMeasureManager ()
```

Returns the *MeasureManager* which stores all the *Measures* supported by this *Measurable* entity

2.45.2 Measure

public interface **Measure**<Value> extends *DescribedEntity*, *Serializable*

A *Measure* aims at providing the Value of a specific property, typically of an *Algorithm*. In order to facilitate external uses, it implements the methods of *DescribedEntity*.

Author Created by Antonio J. Nebro on 21/10/14 based on the ideas of Matthieu Vergne

Parameters

- **<Value>** – the type of value the *Measure* can provide

2.45.3 MeasureListener

public interface **MeasureListener**<Value>

A *MeasureListener* allows to register a given behavior to *PushMeasure*. *register(MeasureListener)*. When the *PushMeasure* generate a Value, it is provided to *MeasureListener.measureGenerated(Object)* in order to execute the specified behavior.

Author Created by Antonio J. Nebro on 21/10/14 based on the ideas of Matthieu Vergne

Parameters

- **<Value>** –

Methods

measureGenerated

public void **measureGenerated** (Value value)

Parameters

- **value** – the Value generated by the *PushMeasure*

2.45.4 MeasureManager

public interface **MeasureManager**

A *MeasureManager* aims at managing a set of *Measures*. Typically, a *Measurable* entity would create a single *MeasureManager* to store all the *Measures* it would like to support, each of them being identified by a key. Because a *Measure* can be whether a *PullMeasure* or a *PushMeasure*, the two methods *getPullMeasure(Object)* and *getPushMeasure(Object)* are provided. It could be that a single *Measure* is also available through both (via a single instance implementing both interfaces or two different instances implementing each interface), leading to have both methods returning a non-null value for the same key.

Author Created by Antonio J. Nebro on 21/10/14 based on the ideas of Matthieu Vergne

Methods

getMeasureKeys

public *Collection*<*Object*> **getMeasureKeys** ()

This method should return all the keys identifying the *Measures* managed by this *MeasureManager*. More precisely, if *getPullMeasure(Object)* or *getPushMeasure(Object)* returns a non-null value for

a given key, then this key should be returned by `getMeasureKeys()`. However, it is not because a key is returned by `getMeasureKeys()` that it necessarily has a *PullMeasure* or a *PushMeasure* returned by this *MeasureManager*.

Returns the set of keys identifying the managed *Measures*

getPullMeasure

```
public <T> PullMeasure<T> getPullMeasure (Object key)
```

Parameters

- **key** – the key of the *Measure*

Returns the *PullMeasure* identified by this key

getPushMeasure

```
public <T> PushMeasure<T> getPushMeasure (Object key)
```

Parameters

- **key** – the key of the *Measure*

Returns the *PushMeasure* identified by this key

2.45.5 PullMeasure

```
public interface PullMeasure<Value> extends Measure<Value>
```

A *PullMeasure* is a *Measure* from which the *Value* can be accessed on demand through the `get()` method. As such, a *PullMeasure* should ensure that its current *Value* is always available or generated before to be returned by `get()`.

Author Created by Antonio J. Nebro on 21/10/14 based on the ideas of Matthieu Vergne

Parameters

- **<Value>** – the type of value the *PullMeasure* can provide

Methods

get

```
public Value get ()
```

Returns the current *Value* of the *Measure*

2.45.6 PushMeasure

```
public interface PushMeasure<Value> extends Measure<Value>
```

A *PushMeasure* is a *Measure* which provides its *Value* through notifications. As such, any observer on a *PushMeasure* should register a *MeasureListener* through `register(MeasureListener)` to specify what to do with the *Value* once it is received.

Author Created by Antonio J. Nebro on 21/10/14 based on the ideas of Matthieu Vergne

Parameters

- **<Value>** – the type of value the *PushMeasure* can provide

Methods**register**

```
public void register (MeasureListener<Value> listener)
```

Register a *MeasureListener* to use the Values of the *PushMeasure* when they are generated.

Parameters

- **listener** – the *MeasureListener* to register

unregister

```
public void unregister (MeasureListener<Value> listener)
```

Unregister a *MeasureListener* registered with *register*(*MeasureListener*) to stop receiving the notifications of the *PushMeasure*.

Parameters

- **listener** – the *MeasureListener* to unregister

2.46 org.uma.jmetal.measure.impl

2.46.1 BasicMeasure

```
public class BasicMeasure<T> extends SimplePushMeasure<T> implements PullMeasure<T>, PushMeasure<T>
```

A *BasicMeasure* provides a simple way to define a measure that merely stores a single value

Author Antonio J. Nebro

Constructors**BasicMeasure**

```
public BasicMeasure ()
```

Create a *BasicMeasure*

Methods**get**

```
public synchronized T get ()
```

Returns the current value

push

public void **push** (T *value*)

set

public synchronized void **set** (T *value*)

Parameters

- **value** – The value to be stored

2.46.2 CountingMeasure

public class **CountingMeasure** extends *SimplePushMeasure<Long>* implements *PullMeasure<Long>*, *PushMeasure<Long>*

A *CountingMeasure* provides a simple way to evaluate a number of occurrences. For instance, it can be used to count how many solutions have been generated within an algorithm, how many evaluations have been computed, how many rounds have been run, etc. If these occurrences are provided by some *PushMeasures*, you can use *link (PushMeasure)* to register the *CountingMeasure* to these *PushMeasures*. Otherwise, use *increment ()* when the *CountingMeasure* need to count one more occurrence. In order to get the count, one can access it immediately through *get ()* or when it is updated by registering a listener with *register (MeasureListener)*.

Author Matthieu Vergne

Fields**count**

long **count**

The current amount of occurrences counted.

Constructors**CountingMeasure**

public **CountingMeasure** (String *name*, String *description*, long *initialCount*)

Create a *CountingMeasure* which starts at a given value. The next value to be pushed to the registered observers will be this value + 1.

Parameters

- **name** – the name of the measure
- **description** – the description of the measure
- **initialCount** – the value to start from

CountingMeasure

public **CountingMeasure** (*String name*, *String description*)

Create a *CountingMeasure* starting from zero. The registered observers will receive their first notification when it will increment to 1.

Parameters

- **name** – the name of the measure
- **description** – the description of the measure

CountingMeasure

public **CountingMeasure** (long *initialCount*)

Create a *CountingMeasure* which starts at a given value. The next value to be pushed to the registered observers will be this value + 1. A default name and description are used.

Parameters

- **initialCount** – the value to start from

CountingMeasure

public **CountingMeasure** ()

Create a *CountingMeasure* starting from zero. The registered observers will receive their first notification when it will increment to 1. A default name and description are used.

Methods

finalize

protected void **finalize** ()

get

public synchronized Long **get** ()

Returns the current amount of occurrences counted

increment

public synchronized void **increment** ()

Add 1 to the current count and push its value to all the registered observers.

increment

public synchronized void **increment** (long *amount*)

Increment the current count in a given amount. If the amount is zero, no change occurs, thus no notification is sent.

Parameters

- **amount** – the amount to add

link

```
public <T> void link (PushMeasure<T> measure)
```

If this *CountingMeasure* is used to count the number of time a *PushMeasure* notifies its observers, you can use this method to link them. The *CountingMeasure* will automatically register a *MeasureListener* on the *PushMeasure* such that, every time the *PushMeasure* send a notification, *CountingMeasure.increment()* is called. You can link several *PushMeasures* at the same time, but each of their notifications will increment the counter, leading to summing their notifications. When a *PushMeasure* should not be considered anymore, use *unlink(PushMeasure)* to remove the link.

Parameters

- **measure** – the *PushMeasure* to link

reset

```
public synchronized void reset ()
```

Restart the counter to zero. Generate a notification if the value was not zero.

reset

```
public synchronized void reset (long value)
```

Restart the counter to a given value. Generate a notification if the value was different.

Parameters

- **value** – the value to restart from

unlink

```
public <T> void unlink (PushMeasure<T> measure)
```

If you have linked a *PushMeasure* through *link(PushMeasure)*, you can discard the link by using this method.

Parameters

- **measure** – the *PushMeasure* to unlink

2.46.3 CountingMeasureTest

```
public class CountingMeasureTest
```

Methods**testGetAlignedWithNotifications**

```
public void testGetAlignedWithNotifications ()
```

testIncrementAddOne

```
public void testIncrementAddOne ()
```

testIncrementNotificationsOccur

```
public void testIncrementNotificationsOccur ()
```

testIncrementNotificationsOccurIfNonZero

```
public void testIncrementNotificationsOccurIfNonZero ()
```

testLinkedMeasureCorrectlyCounted

```
public void testLinkedMeasureCorrectlyCounted ()
```

testMultipleLinkedMeasuresCorrectlyCounted

```
public void testMultipleLinkedMeasuresCorrectlyCounted ()
```

testMultipleLinksOnTheSameMeasureCountedOnce

```
public void testMultipleLinksOnTheSameMeasureCountedOnce ()
```

testReset

```
public void testReset ()
```

testResetNotificationsOccur

```
public void testResetNotificationsOccur ()
```

testResetToAGivenValue

```
public void testResetToAGivenValue ()
```

testUnlinkCorrectlyIgnored

```
public void testUnlinkCorrectlyIgnored ()
```

2.46.4 DurationMeasure

public class **DurationMeasure** extends *SimplePullMeasure*<Long>

This measure allows to have a simple way to compute the time spent in doing something. For instance, an algorithm can compute the time spent to run. In such a case, the algorithm would call *start()* at the beginning of the running and *stop()* at the end. Additional calls to these two methods can also be made during the running to exclude specific parts from the counting. At any time during (and after) the running, the *get()* method can be used to know how much time have been spent so far. If the algorithm is rerun, it will restart and the additional time will sum up to the time already spent before, but it can be avoided by resetting the measure with *reset()*.

Author Matthieu Vergne

Constructors

DurationMeasure

public **DurationMeasure** ()

Methods

get

public Long **get** ()

Returns the total time spent so far

reset

public void **reset** ()

Reset the total time to zero. If a round is currently running, it is restarted.

start

public void **start** ()

Start a round. If the round is already started, it has no effect.

stop

public void **stop** ()

Stop a round. If the round is already stopped, it has no effect.

2.46.5 DurationMeasureTest

public class **DurationMeasureTest**

Methods

testDoNotEvolveBetweenStopAndRestart

```
public void testDoNotEvolveBetweenStopAndRestart ()
```

testIncreasesBetweenStartAndStop

```
public void testIncreasesBetweenStartAndStop ()
```

testNoRunningRemainsAtZero

```
public void testNoRunningRemainsAtZero ()
```

testResetGoBackToZeroWhenStopped

```
public void testResetGoBackToZeroWhenStopped ()
```

testResetRestartFromZeroWhenRunning

```
public void testResetRestartFromZeroWhenRunning ()
```

2.46.6 LastEvaluationMeasure

public class **LastEvaluationMeasure**<Solution, Value> extends *SimplePushMeasure*<*Evaluation*<*Solution*, Value>>
LastEvaluationMeasure is a *PushMeasure* providing the last evaluation made in an algorithm. It extends *SimplePushMeasure* and add the method *push (Object, Object)* for simplicity.

Author Matthieu Vergne

Parameters

- **<Solution>** – the solution evaluated
- **<Value>** – the type of value used to evaluate the solution (Double, BigDecimal, enum, ...)

Constructors

LastEvaluationMeasure

```
public LastEvaluationMeasure ()
```

Methods

push

```
public void push (Solution solution, Value value)
```

This method is equivalent to *push (Object)* excepted that it automatically create the *Evaluation* instance.

Parameters

- **solution** – the solution evaluated
- **value** – the value of this solution

2.46.7 LastEvaluationMeasure.Evaluation

public static class **Evaluation**<Solution, Value>

This structure represent an atomic evaluation of a given solution.

Author Matthieu Vergne

Fields**solution**

Solution **solution**

The solution evaluated.

value

Value **value**

The evaluation of the solution.

2.46.8 LastEvaluationMeasureTest

public class **LastEvaluationMeasureTest**

Methods**testSpecializedPushActLikeParentPush**

public void **testSpecializedPushActLikeParentPush** ()

2.46.9 ListenerTimeMeasure

public class **ListenerTimeMeasure** extends *SimplePullMeasure*<Long> implements *PullMeasure*<Long>

This measure is a facility to evaluate the time spent in *MeasureListeners* registered in *PushMeasures*. In order to measure the time spent in a *MeasureListener*, you should wrap it by calling *wrapListener(MeasureListener)*. The wrapper returned should be used instead of the original *MeasureListener* to allow the *ListenerTimeMeasure* to account for its execution time. If you want to wrap automatically all the *MeasureListeners* registered to a given *PushMeasure*, you can wrap the *PushMeasure* through *wrapMeasure(PushMeasure)*: all the *MeasureListeners* registered to the wrapper will be wrapped too. You can restart the evaluation by calling *reset()*. Notice that the time accounted is not the physical time but the processing time: if several listeners run in parallel, their execution time is summed as if they were running sequentially, thus you can have a measured time which is superior to the physical time spent. If you want to measure the physical time spent in the execution of parallel runs, you should use another way.

Author Matthieu Vergne

Methods

get

public Long **get** ()

Returns the time spent in the wrapped *MeasureListeners*

reset

public void **reset** ()

This method reset the time measured to zero. Notice that *MeasureListeners* which are still running will be affected consequently: their execution time will be measured from the reset time, not from their own starting time.

wrapListener

public <Value> *MeasureListener*<Value> **wrapListener** (*MeasureListener*<Value> wrapped)

This method wrap a *MeasureListener* (the wrapped) into another one (the wrapper). Any notification made via the wrapper will allow to measure how much time has been spent by the wrapped to treat this notification. The wrapped listener is not changed, thus it can be reused in other *PushMeasures* that we don't want to consider. If a wrapper has already been made for the given wrapped, it will be returned and no new one will be instantiated (weak references are used to not keep in memory the unused wrappers).

Parameters

- **wrapped** – the *MeasureListener* to wrap

Returns the *MeasureListener* wrapper

wrapManager

public <Value> *MeasureManager* **wrapManager** (*MeasureManager* wrapped, Object measureKey)

This method wrap a *MeasureManager* (the wrapped) into another one (the wrapper) which provides the same measures, excepted that any *PushMeasure* returned by the wrapper will be automatically wrapped via *wrapMeasure* (*PushMeasure*). This allows to ensure that any *MeasureListener* registered to the *PushMeasures* provided by the wrapper will be considered, independently of who registers it or when it is registered. You can also provide an additional key to add this *ListenerTimeMeasure* to the wrapper. The wrapped manager is not changed, thus it can be reused to register *MeasureListeners* that we don't want to consider.

Parameters

- **wrapped** – the *MeasureManager* to wrap
- **measureKey** – the key that the wrapper should use for this *ListenerTimeMeasure*, null if it should not use it

Returns the *MeasureManager* wrapper

wrapMeasure

```
public <Value> PushMeasure<Value> wrapMeasure (PushMeasure<Value> wrapped)
```

This method wrap a *PushMeasure* (the wrapped) into another one (the wrapper). Any *MeasureListener* registered to the wrapper will be automatically wrapped via *wrapListener* (*MeasureListener*). This allows to ensure that any *MeasureListener* registered will be considered, independently of who registers it or when it is registered. The wrapped measure is not changed, thus it can be reused to register *MeasureListeners* that we don't want to consider. If a wrapper has already been made for the given wrapped, it will be returned and no new one will be instantiated (weak references are used to not keep in memory the unused wrappers).

Parameters

- **wrapped** – the *PushMeasure* to wrap

Returns the *PushMeasure* wrapper

2.46.10 ListenerTimeMeasureTest

```
public class ListenerTimeMeasureTest
```

Methods

testAdditionalKeyForWrappedManagerRejectAlreadyUsedKeys

```
public void testAdditionalKeyForWrappedManagerRejectAlreadyUsedKeys ()
```

testAdditionalKeyForWrappedManagerReturnCurrentMeasure

```
public void testAdditionalKeyForWrappedManagerReturnCurrentMeasure ()
```

testAdditionalKeyProvidedByManager

```
public void testAdditionalKeyProvidedByManager ()
```

testCountTimeInListeners

```
public void testCountTimeInListeners ()
```

testCountTimeInManager

```
public void testCountTimeInManager ()
```

testCountTimeInMeasures

```
public void testCountTimeInMeasures ()
```

testExceptionOnNullListener

```
public void testExceptionOnNullListener ()
```

testExceptionOnNullManager

```
public void testExceptionOnNullManager ()
```

testExceptionOnNullMeasure

```
public void testExceptionOnNullMeasure ()
```

testFakeListener

```
public void testFakeListener ()
```

testForgetListenerWrapperIfNotUsedAnymore

```
public void testForgetListenerWrapperIfNotUsedAnymore ()
```

testForgetMeasureWrapperIfNotUsedAnymore

```
public void testForgetMeasureWrapperIfNotUsedAnymore ()
```

testResetToCurrentTimeWhenListenerIsRunning

```
public void testResetToCurrentTimeWhenListenerIsRunning ()
```

testResetToZeroWhenNoListenerIsRunning

```
public void testResetToZeroWhenNoListenerIsRunning ()
```

testReturnSameWrapperForSameListener

```
public void testReturnSameWrapperForSameListener ()
```

testReturnSameWrapperForSameMeasure

```
public void testReturnSameWrapperForSameMeasure ()
```

testSameNameAndDescriptionThanOriginalMeasure

```
public void testSameNameAndDescriptionThanOriginalMeasure ()
```

2.46.11 MeasureFactory

public class **MeasureFactory**

The *MeasureFactory* provides some useful methods to build specific *Measures*.

Author Matthieu Vergne

Methods

createPullFromPush

public <Value> *PullMeasure*<Value> **createPullFromPush** (*PushMeasure*<Value> *push*, Value *initial-Value*)

Create a *PullMeasure* to backup the last Value of a *PushMeasure*. When the *PushMeasure* send a notification with a given Value, this Value is stored into a variable so that it can be retrieved at any time through the method *PullMeasure.get()*.

Parameters

- **push** – a *PushMeasure* to backup
- **initialValue** – the Value to return before the next notification of the *PushMeasure* is sent

Returns a *PullMeasure* allowing to retrieve the last value sent by the *PushMeasure*, or the initial value if it did not send any

createPullsFromFields

public Map<String, *PullMeasure*<?>> **createPullsFromFields** (Object *object*)

Create *PullMeasures* based on the fields available from an instance, whatever it is. The *Class* of the instance is analyzed to retrieve its public fields and a *PullMeasure* is built for each of them. The name of the field is further exploited to identify the measure, such that the map returned use the name of the field as a key which maps to the *PullMeasure* built from this field. The *PullMeasure* itself is named by using the name of the field.

Parameters

- **object** – the *Object* to cover

Returns the Map which contains the names of the getter methods and the corresponding *PullMeasure* built from them

createPullsFromGetters

public Map<String, *PullMeasure*<?>> **createPullsFromGetters** (Object *object*)

Create *PullMeasures* based on the getters available from an instance, whatever it is. The *Class* of the instance is analyzed to retrieve its public methods and a *PullMeasure* is built for each method which use a getter-like signature. The name of the method is further exploited to identify the measure, such that the map returned use the name of the method (without “get”) as a key which maps to the *PullMeasure* built from this method. The *PullMeasure* itself is named by using the name of the method.

Parameters

- **object** – the *Object* to cover

Returns the `Map` which contains the names of the getter methods and the corresponding `PullMeasure` built from them

createPushFromPull

```
public <Value> PushMeasure<Value> createPushFromPull (PullMeasure<Value> pull, long period)
```

Create a `PushMeasure` which checks at regular intervals the value of a `PullMeasure`. If the value have changed since the last check (or since the creation of the `PushMeasure`), a notification will be generated by the `PushMeasure` with the new `Value`. Notice that if the period is too small, the checking process could have a significant impact on performances, because a `Thread` is run in parallel to check regularly the `Value` modifications. If the period is too big, you could miss relevant notifications, especially if the `PullMeasure` change to a new `Value` and change back to its previous `Value` between two consecutive checks. In such a case, no notification will be sent because the `Value` during the two checks is equal.

Parameters

- **pull** – the `PullMeasure` to cover
- **period** – the number of milliseconds between each check

Returns a `PushMeasure` which will notify any change occurred on the `PullMeasure` at the given frequency

2.46.12 MeasureFactoryTest

```
public class MeasureFactoryTest
```

Methods

testCreatePullFromPush

```
public void testCreatePullFromPush ()
```

testCreatePullsFromFieldsRetrieveAllInheritedPublicFields

```
public void testCreatePullsFromFieldsRetrieveAllInheritedPublicFields ()
```

testCreatePullsFromFieldsRetrieveAllInstantiatedPublicFields

```
public void testCreatePullsFromFieldsRetrieveAllInstantiatedPublicFields ()
```

testCreatePullsFromFieldsRetrieveNoInheritedProtectedNorPrivateField

```
public void testCreatePullsFromFieldsRetrieveNoInheritedProtectedNorPrivateField ()
```

testCreatePullsFromFieldsRetrieveNoInstantiatedProtectedNorPrivateField

```
public void testCreatePullsFromFieldsRetrieveNoInstantiatedProtectedNorPrivateField ()
```

testCreatePullsFromFieldsRetrieveNothingFromEmptyObject

```
public void testCreatePullsFromFieldsRetrieveNothingFromEmptyObject ()
```

testCreatePullsFromGettersRetrieveAllInheritedPublicGetters

```
public void testCreatePullsFromGettersRetrieveAllInheritedPublicGetters ()
```

testCreatePullsFromGettersRetrieveAllInstantiatedPublicGetters

```
public void testCreatePullsFromGettersRetrieveAllInstantiatedPublicGetters ()
```

testCreatePullsFromGettersRetrieveNoInheritedProtectedNorPrivateGetter

```
public void testCreatePullsFromGettersRetrieveNoInheritedProtectedNorPrivateGetter ()
```

testCreatePullsFromGettersRetrieveNoInstantiatedProtectedNorPrivateGetter

```
public void testCreatePullsFromGettersRetrieveNoInstantiatedProtectedNorPrivateGetter ()
```

testCreatePullsFromGettersRetrieveNothingFromEmptyObject

```
public void testCreatePullsFromGettersRetrieveNothingFromEmptyObject ()
```

testCreatePushFromPullNotifiesOnlyWhenValueChanged

```
public void testCreatePushFromPullNotifiesOnlyWhenValueChanged ()
```

testCreatePushFromPullNotifiesWithTheCorrectFrequency

```
public void testCreatePushFromPullNotifiesWithTheCorrectFrequency ()
```

testCreatePushFromPullStopNotificationsWhenPullDestroyed

```
public void testCreatePushFromPullStopNotificationsWhenPullDestroyed ()
```

testCreatePushFromPullStopNotificationsWhenPushDestroyed

```
public void testCreatePushFromPullStopNotificationsWhenPushDestroyed ()
```

2.46.13 PullPushMeasure

public class **PullPushMeasure**<Value> implements *PullMeasure*<Value>, *PushMeasure*<Value>

A *PullPushMeasure* aims at providing both the *PushMeasure* and *PullMeasure* abilities into a single *Measure*. One could simply build a brand new *Measure* by calling *PullPushMeasure*(*String*, *String*), but in the case where some existing measures are available, he can wrap them into a *PullPushMeasure* by calling *PullPushMeasure*(*PushMeasure*, *Object*) or other constructors taking a *Measure* as argument.

Author Matthieu Vergne

Parameters

- <Value> –

Constructors

PullPushMeasure

public **PullPushMeasure** (*PullMeasure*<Value> *pull*, *PushMeasure*<Value> *push*, *DescribedEntity* *reference*)

Create a *PullPushMeasure* which wraps both a *PullMeasure* and a *PushMeasure*. The assumption is that both *Measures* already represent the same *Measure* (i.e. the same Value) but were implemented separately. Instantiating a *PullPushMeasure* this way allows to merge them easily without creating a completely new measure. Don't use this constructor to merge two different *Measures*. The last parameter is generally used to specify which of the two *Measures* should be used for *getName()* and *getDescription()*, but you can also provide a completely new instance to change them.

Parameters

- **pull** – the *PullMeasure* to wrap
- **push** – the *PushMeasure* to wrap
- **reference** – the reference to use for the name and the description of this *PullPushMeasure*

PullPushMeasure

public **PullPushMeasure** (*PullMeasure*<Value> *pull*, *PushMeasure*<Value> *push*, *String* *name*, *String* *description*)

Equivalent to *PullPushMeasure*(*PullMeasure*, *PushMeasure*, *DescribedEntity*) but the reference parameter is replaced by the specific name and description that you want to provide. This is a shortcut to the creation of the *DescribedEntity* instance followed by the call of the reference-based method.

Parameters

- **pull** – the *PullMeasure* to wrap
- **push** – the *PushMeasure* to wrap
- **name** – the name of the *PullPushMeasure*
- **description** – the description of the *PullPushMeasure*

PullPushMeasure

public **PullPushMeasure** (*PushMeasure*<Value> *push*, Value *initialValue*)

Create a *PullPushMeasure* which wraps a *PushMeasure*. The *PullMeasure* ability corresponds the storage of the Value pushed by the *PushMeasure* in order to retrieve it on demand through *PullMeasure.get()*. The name and the description of the *PullPushMeasure* are the ones provided by the wrapped *PushMeasure*.

Parameters

- **push** – the *PushMeasure* to wraps
- **initialValue** – the Value to return before the next notification of the *PushMeasure*

PullPushMeasure

public **PullPushMeasure** (String *name*, String *description*)

Create a *PullPushMeasure* from scratch.

Parameters

- **name** – the name of the *PullPushMeasure*
- **description** – the description of the *PullPushMeasure*

Methods

get

public Value **get** ()

getDescription

public String **getDescription** ()

getName

public String **getName** ()

register

public void **register** (*MeasureListener*<Value> *listener*)

unregister

public void **unregister** (*MeasureListener*<Value> *listener*)

2.46.14 SimpleMeasure

public class **SimpleMeasure**<Value> extends *SimpleDescribedEntity* implements *Measure*<Value>
SimpleMeasure is a basic implementation of *Measure*. It provides a basic support for the most generic properties required by this interface.

Author Matthieu Vergne

Parameters

- <Value> –

Constructors

SimpleMeasure

public **SimpleMeasure** (*String* name, *String* description)
Create a *SimpleMeasure* with a given name and a given description.

Parameters

- **name** – the name of the *Measure*
- **description** – the description of the *Measure*

SimpleMeasure

public **SimpleMeasure** (*String* name)
Create a *SimpleMeasure* with a given name and a null description.

Parameters

- **name** – the name of the *Measure*

SimpleMeasure

public **SimpleMeasure** ()
Create a *SimpleMeasure* with the class name as its name and a null description.

2.46.15 SimpleMeasureManager

public class **SimpleMeasureManager** implements *MeasureManager*
This *SimpleMeasureManager* provides a basic implementation to manage a collection of *Measures*. One can use the setXxxMeasure() methods to configure the *MeasureManager* with the finest granularity, or exploit the centralized *setMeasure(Object, Measure)* to register a *Measure* depending on the interfaces it implements, or even use the massive *setAllMeasures(Map)* to register a set of *Measures* at once. The corresponding removeXxx methods are also available for each case. However, the only way to access a *Measure* is through the finest granularity with *getPullMeasure(Object)* and *getPushMeasure(Object)*.

Author Matthieu Vergne

Methods

getMeasureKeys

public `Collection<Object>` **getMeasureKeys** ()

Provides the keys of all the *Measures* which are supported by this *SimpleMeasureManager*. If a key is provided, then at least one version is available through *getPullMeasure(Object)* or *getPushMeasure(Object)*.

getPullMeasure

public `<T> PullMeasure<T>` **getPullMeasure** (`Object key`)

getPushMeasure

public `<T> PushMeasure<T>` **getPushMeasure** (`Object key`)

removeAllMeasures

public void **removeAllMeasures** (`Iterable<? extends Object> keys`)

Massive equivalent to *removeMeasure(Object)*.

Parameters

- **keys** – the keys of the *Measures* to remove

removeMeasure

public void **removeMeasure** (`Object key`)

This method removes an entire *Measure*, meaning that if both a *PullMeasure* and a *PushMeasure* are registered for this key, then both are removed.

Parameters

- **key** – the key of the *Measure* to remove

removePullMeasure

public void **removePullMeasure** (`Object key`)

Parameters

- **key** – the key of the *PullMeasure* to remove

removePushMeasure

public void **removePushMeasure** (`Object key`)

Parameters

- **key** – the key of the *PushMeasure* to remove

setAllMeasures

public void **setAllMeasures** (Map<? extends *Object*, ? extends *Measure*<?>> *measures*)
Massive equivalent of *setMeasure (Object, Measure)*.

Parameters

- **measures** – the *Measures* to register with their corresponding keys

setMeasure

public void **setMeasure** (*Object* *key*, *Measure*<?> *measure*)

This method call *setPullMeasure (Object, PullMeasure)* or *setPushMeasure (Object, PushMeasure)* depending on the interfaces implemented by the *Measure* given in argument. If both interfaces are implemented, both methods are called, allowing to register all the aspects of the *Measure* in one call.

Parameters

- **key** – the key of the *Measure*
- **measure** – the *Measure* to register

setPullMeasure

public void **setPullMeasure** (*Object* *key*, *PullMeasure*<?> *measure*)

Parameters

- **key** – the key of the *Measure*
- **measure** – the *PullMeasure* to register

setPushMeasure

public void **setPushMeasure** (*Object* *key*, *PushMeasure*<?> *measure*)

Parameters

- **key** – the key of the *Measure*
- **measure** – the *PushMeasure* to register

2.46.16 SimpleMeasureManagerTest

public class **SimpleMeasureManagerTest**

Methods

testAddMeasureAddKey

public void **testAddMeasureAddKey** ()

testAddMultipleMeasures

```
public void testAddMultipleMeasures ()
```

testRemoveBothAtOnce

```
public void testRemoveBothAtOnce ()
```

testRemoveBothMeasuresRemoveKey

```
public void testRemoveBothMeasuresRemoveKey ()
```

testRemoveMultipleMeasures

```
public void testRemoveMultipleMeasures ()
```

testSetGetPullMeasure

```
public void testSetGetPullMeasure ()
```

testSetGetPushMeasure

```
public void testSetGetPushMeasure ()
```

testSetMeasureGetBoth

```
public void testSetMeasureGetBoth ()
```

testStartEmpty

```
public void testStartEmpty ()
```

2.46.17 SimplePullMeasure

public abstract class **SimplePullMeasure**<Value> extends *SimpleMeasure*<Value> implements *PullMeasure*<Value>
SimplePullMeasure is a basic implementation of *PullMeasure*. As a *PullMeasure*, it is intended to be used by external entities through its *get()* method. This method must be implemented by the algorithm to specify how the value can be retrieved.

Author Matthieu Vergne

Parameters

- <Value> –

Constructors

SimplePullMeasure

public **SimplePullMeasure** (String name, String description)

Create a *SimplePullMeasure* with a given name and a given description.

Parameters

- **name** – the name of the *Measure*
- **description** – the description of the *Measure*

SimplePullMeasure

public **SimplePullMeasure** (String name)

Create a *SimplePullMeasure* with a given name and a null description.

Parameters

- **name** – the name of the *Measure*

SimplePullMeasure

public **SimplePullMeasure** ()

Create a *SimplePullMeasure* with the class name as its name and a null description.

2.46.18 SimplePushMeasure

public class **SimplePushMeasure**<Value> extends *SimpleMeasure*<Value> implements *PushMeasure*<Value>

SimplePushMeasure is a basic implementation of *PushMeasure*. As a *PushMeasure*, it is intended to be fed by the algorithm while external entities should use *register (MeasureListener)* to be notified in real time. For the algorithm to feed it, it should provide a solution and its value to *push (Object, Object)*, leading to the notification of the registered observers.

Author Matthieu Vergne

Parameters

- **<Value>** –

Constructors

SimplePushMeasure

public **SimplePushMeasure** (String name, String description)

Create a *SimplePushMeasure* with a given name and a given description.

Parameters

- **name** – the name of the *Measure*
- **description** – the description of the *Measure*

SimplePushMeasure

public **SimplePushMeasure** (*String name*)

Create a *SimplePushMeasure* with a given name and a null description.

Parameters

- **name** – the name of the *Measure*

SimplePushMeasure

public **SimplePushMeasure** ()

Create a *SimplePushMeasure* with the class name as its name and a null description.

Methods

push

public void **push** (*Value value*)

Notify the observers which has registered a *MeasureListener* through *register (MeasureListener)* about a value.

Parameters

- **value** – the value to send to the observers

register

public void **register** (*MeasureListener<Value> listener*)

unregister

public void **unregister** (*MeasureListener<Value> listener*)

2.46.19 SimplePushMeasureTest

public class **SimplePushMeasureTest**

Methods

testNotNotifiedWhenUnregistered

public void **testNotNotifiedWhenUnregistered** ()

testNotifiedWhenRegistered

public void **testNotifiedWhenRegistered** ()

2.47 org.uma.jmetal.operator

2.47.1 CrossoverOperator

public interface **CrossoverOperator**<Source> extends *Operator*<List<Source>, List<Source>>

Interface representing crossover operators. They will receive a list of solutions and return another list of solutions

Author Antonio J. Nebro

Parameters

- <Source> – The class of the solutions

Methods

getNumberOfGeneratedChildren

int **getNumberOfGeneratedChildren** ()

getNumberOfRequiredParents

int **getNumberOfRequiredParents** ()

2.47.2 LocalSearchOperator

public interface **LocalSearchOperator**<Source> extends *Operator*<Source, Source>

Interface representing a local search operator Created by cbarba on 5/3/15.

Methods

getEvaluations

int **getEvaluations** ()

getNumberOfImprovements

int **getNumberOfImprovements** ()

getNumberOfNonComparableSolutions

int **getNumberOfNonComparableSolutions** ()

2.47.3 MutationOperator

public interface **MutationOperator**<Source> extends *Operator*<Source, Source>

Interface representing mutation operators

Author Antonio J. Nebro

Parameters

- **<Source>** – The solution class of the solution to be mutated

2.47.4 Operator

public interface **Operator**<Source, Result> extends [Serializable](#)

Interface representing an operator

Author Antonio J. Nebro

Parameters

- **<Source>** – Source Class of the object to be operated with
- **<Result>** – Result Class of the result obtained after applying the operator

Methods**execute**

Result **execute** (Source *source*)

Parameters

- **source** – The data to process

2.47.5 SelectionOperator

public interface **SelectionOperator**<Source, Result> extends [Operator](#)<Source, Result>

Interface representing selection operators

Author Antonio J. Nebro

Parameters

- **<Source>** – Class of the source object (typically, a list of solutions)
- **<Result>** – Class of the result of applying the operator

2.48 org.uma.jmetal.operator.impl.crossover

2.48.1 BLXAlphaCrossover

public class **BLXAlphaCrossover** implements [CrossoverOperator](#)<[DoubleSolution](#)>

This class allows to apply a BLX-alpha crossover operator to two parent solutions.

Author Antonio J. Nebro

Constructors

BLXAlphaCrossover

```
public BLXAlphaCrossover (double crossoverProbability)  
    Constructor
```

BLXAlphaCrossover

```
public BLXAlphaCrossover (double crossoverProbability, double alpha)  
    Constructor
```

BLXAlphaCrossover

```
public BLXAlphaCrossover (double crossoverProbability, double alpha, RepairDoubleSolution solution-  
                           Repair)  
    Constructor
```

BLXAlphaCrossover

```
public BLXAlphaCrossover (double crossoverProbability, double alpha, RepairDoubleSolution solution-  
                           Repair, RandomGenerator<Double> randomGenerator)  
    Constructor
```

Methods

doCrossover

```
public List<DoubleSolution> doCrossover (double probability, DoubleSolution parent1, DoubleSolution  
                                         parent2)  
    doCrossover method
```

execute

```
public List<DoubleSolution> execute (List<DoubleSolution> solutions)  
    Execute() method
```

getAlpha

```
public double getAlpha ()
```

getCrossoverProbability

```
public double getCrossoverProbability ()
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

setAlpha

```
public void setAlpha (double alpha)
```

setCrossoverProbability

```
public void setCrossoverProbability (double crossoverProbability)
```

2.48.2 BLXAlphaCrossoverTest

```
public class BLXAlphaCrossoverTest
```

Note: this class does check that the BLX-alpha crossover operator does not return invalid values, but not that it works properly (@see BLXAlphaCrossoverWorkingTest)

Author Antonio J. Nebro

Methods**shouldConstructorAssignTheCorrectDistributionIndex**

```
public void shouldConstructorAssignTheCorrectDistributionIndex ()
```

shouldConstructorAssignTheCorrectProbabilityValue

```
public void shouldConstructorAssignTheCorrectProbabilityValue ()
```

shouldConstructorFailWhenPassedANegativeAlphaValue

```
public void shouldConstructorFailWhenPassedANegativeAlphaValue ()
```

shouldConstructorFailWhenPassedANegativeProbabilityValue

```
public void shouldConstructorFailWhenPassedANegativeProbabilityValue ()
```

shouldCrossingTwoDoubleVariableSolutionsReturnValidSolutions

```
public void shouldCrossingTwoDoubleVariableSolutionsReturnValidSolutions ()
```

shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfNotCrossoverIsApplied

```
public void shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfNotCrossoverIsApplied()
```

shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfProbabilityIsZero

```
public void shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfProbabilityIsZero()
```

shouldCrossingTwoSingleVariableSolutionsReturnValidSolutions

```
public void shouldCrossingTwoSingleVariableSolutionsReturnValidSolutions()
```

shouldCrossingTwoSingleVariableSolutionsWithSimilarValueReturnTheSameVariables

```
public void shouldCrossingTwoSingleVariableSolutionsWithSimilarValueReturnTheSameVariables()
```

shouldExecuteWithInvalidSolutionListSizeThrowAnException

```
public void shouldExecuteWithInvalidSolutionListSizeThrowAnException()
```

shouldExecuteWithNullParameterThrowAnException

```
public void shouldExecuteWithNullParameterThrowAnException()
```

shouldGetAlphaReturnTheRightValue

```
public void shouldGetAlphaReturnTheRightValue()
```

shouldGetProbabilityReturnTheRightValue

```
public void shouldGetProbabilityReturnTheRightValue()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided()
```

2.48.3 DifferentialEvolutionCrossover

```
public class DifferentialEvolutionCrossover implements CrossoverOperator<DoubleSolution>
    Differential evolution crossover operator
```

Author Antonio J. Nebro Comments: - The operator receives two parameters: the current individual and an array of three parent individuals - The best and rand variants depends on the third parent, according whether it represents the current of the “best” individual or a random one. The implementation of both variants are the same, due to that the parent selection is external to the

crossover operator. - Implemented variants: - rand/1/bin (best/1/bin) - rand/1/exp (best/1/exp) - current-to-rand/1 (current-to-best/1) - current-to-rand/1/bin (current-to-best/1/bin) - current-to-rand/1/exp (current-to-best/1/exp)

Constructors

DifferentialEvolutionCrossover

```
public DifferentialEvolutionCrossover ()
    Constructor
```

DifferentialEvolutionCrossover

```
public DifferentialEvolutionCrossover (double cr, double f, String variant)
    Constructor
```

Parameters

- **cr** –
- **f** –
- **variant** –

DifferentialEvolutionCrossover

```
public DifferentialEvolutionCrossover (double cr, double f, String variant, RandomGenerator<Double> randomGenerator)
    Constructor
```

Parameters

- **cr** –
- **f** –
- **variant** –
- **jRandomGenerator** –
- **crRandomGenerator** –

DifferentialEvolutionCrossover

```
public DifferentialEvolutionCrossover (double cr, double f, String variant, BoundedRandomGenerator<Integer> jRandomGenerator, BoundedRandomGenerator<Double> crRandomGenerator)
    Constructor
```

Parameters

- **cr** –
- **f** –
- **variant** –
- **jRandomGenerator** –

- `crRandomGenerator` –

DifferentialEvolutionCrossover

public **DifferentialEvolutionCrossover** (double *cr*, double *f*, double *k*, *String* *variant*)
Constructor

Methods

execute

public *List*<*DoubleSolution*> **execute** (*List*<*DoubleSolution*> *parentSolutions*)
Execute() method

getCr

public double **getCr** ()

getF

public double **getF** ()

getK

public double **getK** ()

getNumberOfGeneratedChildren

public int **getNumberOfGeneratedChildren** ()

getNumberOfRequiredParents

public int **getNumberOfRequiredParents** ()

getVariant

public *String* **getVariant** ()

setCr

public void **setCr** (double *cr*)

setCurrentSolution

public void **setCurrentSolution** (*DoubleSolution* current)

setF

public void **setF** (double *f*)

setK

public void **setK** (double *k*)

2.48.4 DifferentialEvolutionCrossoverTest

public class **DifferentialEvolutionCrossoverTest**

Methods

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

public void **shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided** ()

2.48.5 HUXCrossover

public class **HUXCrossover** implements *CrossoverOperator<BinarySolution>*

This class allows to apply a HUX crossover operator using two parent solutions. NOTE: the operator is applied to the first encoding.variable of the solutions, and the type of the solutions must be Binary

Author Antonio J. Nebro, Juan J. Durillo

Constructors

HUXCrossover

public **HUXCrossover** (double *crossoverProbability*)
Constructor

HUXCrossover

public **HUXCrossover** (double *crossoverProbability*, *RandomGenerator<Double>* randomGenerator)
Constructor

Methods

doCrossover

```
public List<BinarySolution> doCrossover (double probability, BinarySolution parent1, BinarySolution parent2)
```

Perform the crossover operation

Parameters

- **probability** – Crossover setProbability
- **parent1** – The first parent
- **parent2** – The second parent

Throws

- *org.uma.jmetal.util.JMetalException* –

Returns An array containing the two offspring

execute

```
public List<BinarySolution> execute (List<BinarySolution> parents)
```

Execute() method

getCrossoverProbability

```
public double getCrossoverProbability ()
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

setCrossoverProbability

```
public void setCrossoverProbability (double crossoverProbability)
```

2.48.6 HUXCrossoverTest

```
public class HUXCrossoverTest
```


Methods

testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided

```
public void testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided ()
```

2.48.7 IntegerSBXCrossover

```
public class IntegerSBXCrossover implements CrossoverOperator<IntegerSolution>
```

This class allows to apply a SBX crossover operator using two parent solutions (Integer encoding)

Author Antonio J. Nebro

Constructors

IntegerSBXCrossover

```
public IntegerSBXCrossover (double crossoverProbability, double distributionIndex)
```

Constructor

IntegerSBXCrossover

```
public IntegerSBXCrossover (double crossoverProbability, double distributionIndex, RandomGenerator<Double> randomGenerator)
```

Constructor

Methods

doCrossover

```
public List<IntegerSolution> doCrossover (double probability, IntegerSolution parent1, IntegerSolution parent2)
```

doCrossover method

execute

```
public List<IntegerSolution> execute (List<IntegerSolution> solutions)
```

Execute() method

getCrossoverProbability

```
public double getCrossoverProbability ()
```

getDistributionIndex

```
public double getDistributionIndex ()
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

setCrossoverProbability

```
public void setCrossoverProbability (double crossoverProbability)
```

setDistributionIndex

```
public void setDistributionIndex (double distributionIndex)
```

2.48.8 IntegerSBXCrossoverTest

```
public class IntegerSBXCrossoverTest
```

Methods

testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided

```
public void testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided ()
```

2.48.9 NPointCrossover

```
public class NPointCrossover<T> implements CrossoverOperator<Solution<T>>  
    Created by FlapKap on 23-03-2017.
```

Constructors

NPointCrossover

```
public NPointCrossover (double probability, int crossovers)
```

NPointCrossover

```
public NPointCrossover (int crossovers)
```

Methods

execute

```
public List<Solution<T>> execute (List<Solution<T>> s)
```

getCrossoverProbability

```
public double getCrossoverProbability ()
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

2.48.10 NullCrossover

```
public class NullCrossover<S extends Solution<?>> implements CrossoverOperator<S>
```

This class defines a null crossover operator: the parent solutions are returned without any change. It can be useful when configuring a genetic algorithm and we want to use only mutation.

Author Antonio J. Nebro

Methods

execute

```
public List<S> execute (List<S> source)  
    Execute() method
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

2.48.11 NullCrossoverTest

```
public class NullCrossoverTest  
    Created by ajnebro on 10/6/15.
```

Methods

shouldExecuteReturnTwoDifferentObjectsWhichAreEquals

```
public void shouldExecuteReturnTwoDifferentObjectsWhichAreEquals ()
```

2.48.12 PMXCrossover

```
public class PMXCrossover implements CrossoverOperator<PermutationSolution<Integer>>
```

This class allows to apply a PMX crossover operator using two parent solutions.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

PMXCrossover

```
public PMXCrossover (double crossoverProbability)  
    Constructor
```

PMXCrossover

```
public PMXCrossover (double crossoverProbability, RandomGenerator<Double> randomGenerator)  
    Constructor
```

PMXCrossover

```
public PMXCrossover (double crossoverProbability, RandomGenerator<Double> crossoverRandomGenerator,  
    BoundedRandomGenerator<Integer> cuttingPointRandomGenerator)  
    Constructor
```

Methods

doCrossover

```
public List<PermutationSolution<Integer>> doCrossover (double probability,  
    List<PermutationSolution<Integer>> parents)  
    Perform the crossover operation
```

Parameters

- **probability** – Crossover probability
- **parents** – Parents

Returns An array containing the two offspring

execute

public List<*PermutationSolution*<Integer>> **execute** (List<*PermutationSolution*<Integer>> parents)
 Executes the operation

Parameters

- **parents** – An object containing an array of two solutions

getCrossoverProbability

public double **getCrossoverProbability** ()

getNumberOfGeneratedChildren

public int **getNumberOfGeneratedChildren** ()

getNumberOfRequiredParents

public int **getNumberOfRequiredParents** ()

setCrossoverProbability

public void **setCrossoverProbability** (double *crossoverProbability*)

2.48.13 PMXCrossoverTest

public class **PMXCrossoverTest**

Methods**shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided**

public void **shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided** ()

2.48.14 SBXCrossover

public class **SBXCrossover** implements *CrossoverOperator*<*DoubleSolution*>

This class allows to apply a SBX crossover operator using two parent solutions (Double encoding). A *RepairDoubleSolution* object is used to decide the strategy to apply when a value is out of range. The implementation is based on the NSGA-II code available in <http://www.iitk.ac.in/kangal/codes.shtml>

Author Antonio J. Nebro , Juan J. Durillo

Constructors

SBXCrossover

```
public SBXCrossover (double crossoverProbability, double distributionIndex)  
    Constructor
```

SBXCrossover

```
public SBXCrossover (double crossoverProbability, double distributionIndex, RandomGenerator<Double>  
    randomGenerator)  
    Constructor
```

SBXCrossover

```
public SBXCrossover (double crossoverProbability, double distributionIndex, RepairDoubleSolution solu-  
    tionRepair)  
    Constructor
```

SBXCrossover

```
public SBXCrossover (double crossoverProbability, double distributionIndex, RepairDoubleSolution solu-  
    tionRepair, RandomGenerator<Double> randomGenerator)  
    Constructor
```

Methods

doCrossover

```
public List<DoubleSolution> doCrossover (double probability, DoubleSolution parent1, DoubleSolution  
    parent2)  
    doCrossover method
```

execute

```
public List<DoubleSolution> execute (List<DoubleSolution> solutions)  
    Execute() method
```

getCrossoverProbability

```
public double getCrossoverProbability ()
```

getDistributionIndex

```
public double getDistributionIndex ()
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

setCrossoverProbability

```
public void setCrossoverProbability (double probability)
```

setDistributionIndex

```
public void setDistributionIndex (double distributionIndex)
```

2.48.15 SBXCrossoverTest

```
public class SBXCrossoverTest
```

Note: this class does check that the SBX crossover operator does not return invalid values, but not that it works properly (@see SBXCrossoverWorkingTest)

Author Antonio J. Nebro

Methods**shouldConstructorAssignTheCorrectDistributionIndex**

```
public void shouldConstructorAssignTheCorrectDistributionIndex ()
```

shouldConstructorAssignTheCorrectProbabilityValue

```
public void shouldConstructorAssignTheCorrectProbabilityValue ()
```

shouldConstructorFailWhenPassedANegativeDistributionIndex

```
public void shouldConstructorFailWhenPassedANegativeDistributionIndex ()
```

shouldConstructorFailWhenPassedANegativeProbabilityValue

```
public void shouldConstructorFailWhenPassedANegativeProbabilityValue ()
```

shouldCrossingTheSecondVariableReturnTheOtherVariablesUnchangedInTheOffspringSolutions

```
public void shouldCrossingTheSecondVariableReturnTheOtherVariablesUnchangedInTheOffspringSolutions ()
```

shouldCrossingTwoDoubleVariableSolutionsReturnValidSolutions

```
public void shouldCrossingTwoDoubleVariableSolutionsReturnValidSolutions ()
```

shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfNotCrossoverIsApplied

```
public void shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfNotCrossoverIsApplied ()
```

shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfProbabilityIsZero

```
public void shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutionsIfProbabilityIsZero ()
```

shouldCrossingTwoSingleVariableSolutionsReturnValidSolutions

```
public void shouldCrossingTwoSingleVariableSolutionsReturnValidSolutions ()
```

shouldCrossingTwoSingleVariableSolutionsWithSimilarValueReturnTheSameVariables

```
public void shouldCrossingTwoSingleVariableSolutionsWithSimilarValueReturnTheSameVariables ()
```

shouldExecuteWithInvalidSolutionListSizeThrowAnException

```
public void shouldExecuteWithInvalidSolutionListSizeThrowAnException ()
```

shouldExecuteWithNullParameterThrowAnException

```
public void shouldExecuteWithNullParameterThrowAnException ()
```

shouldGetDistributionIndexReturnTheRightValue

```
public void shouldGetDistributionIndexReturnTheRightValue ()
```

shouldGetProbabilityReturnTheRightValue

```
public void shouldGetProbabilityReturnTheRightValue ()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided ()
```


2.48.16 SinglePointCrossover

public class **SinglePointCrossover** implements *CrossoverOperator<BinarySolution>*

This class implements a single point crossover operator.

Author Antonio J. Nebro

Constructors

SinglePointCrossover

public **SinglePointCrossover** (double *crossoverProbability*)
 Constructor

SinglePointCrossover

public **SinglePointCrossover** (double *crossoverProbability*, *RandomGenerator<Double>* *randomGenerator*)
 Constructor

SinglePointCrossover

public **SinglePointCrossover** (double *crossoverProbability*, *RandomGenerator<Double>* *crossoverRandomGenerator*, *BoundedRandomGenerator<Integer>* *pointRandomGenerator*)
 Constructor

Methods

doCrossover

public *List<BinarySolution>* **doCrossover** (double *probability*, *BinarySolution* *parent1*, *BinarySolution* *parent2*)
 Perform the crossover operation.

Parameters

- **probability** – Crossover setProbability
- **parent1** – The first parent
- **parent2** – The second parent

Returns An array containing the two offspring

execute

public *List<BinarySolution>* **execute** (*List<BinarySolution>* *solutions*)

getCrossoverProbability

public double **getCrossoverProbability** ()

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

setCrossoverProbability

```
public void setCrossoverProbability (double crossoverProbability)
```

2.48.17 SinglePointCrossoverTest

```
public class SinglePointCrossoverTest
```

Methods

shouldConstructorAssignTheCorrectProbabilityValue

```
public void shouldConstructorAssignTheCorrectProbabilityValue ()
```

shouldConstructorFailWhenPassedANegativeProbabilityValue

```
public void shouldConstructorFailWhenPassedANegativeProbabilityValue ()
```

shouldCrossingTheBitInTheMiddleOfSecondVariableReturnTheCorrectCrossedSolutions

```
public void shouldCrossingTheBitInTheMiddleOfSecondVariableReturnTheCorrectCrossedSolutions ()
```

shouldCrossingTheBitInTheMiddleOfTwoSingleVariableSolutionsReturnTheCorrectCrossedSolutions

```
public void shouldCrossingTheBitInTheMiddleOfTwoSingleVariableSolutionsReturnTheCorrectCrossedSolutions ()
```

shouldCrossingTheFistBitOfSecondVariableReturnTheCorrectCrossedSolutions

```
public void shouldCrossingTheFistBitOfSecondVariableReturnTheCorrectCrossedSolutions ()
```

shouldCrossingTheFistBitOfTwoSingleVariableSolutionsReturnTheCorrectCrossedSolutions

```
public void shouldCrossingTheFistBitOfTwoSingleVariableSolutionsReturnTheCorrectCrossedSolutions ()
```

shouldCrossingTheLastBitOfTwoSingleVariableSolutionsReturnTheCorrectCrossedSolutions

```
public void shouldCrossingTheLastBitOfTwoSingleVariableSolutionsReturnTheCorrectCrossedSolutions()
```

shouldCrossingTwoVariableSolutionsReturnTheSameSolutionsIfNoBitsAreMutated

```
public void shouldCrossingTwoVariableSolutionsReturnTheSameSolutionsIfNoBitsAreMutated()
```

shouldExecuteFailIfTheListContainsMoreThanTwoSolutions

```
public void shouldExecuteFailIfTheListContainsMoreThanTwoSolutions()
```

shouldExecuteFailIfTheListContainsOnlyOneSolution

```
public void shouldExecuteFailIfTheListContainsOnlyOneSolution()
```

shouldExecuteWithNullParameterThrowAnException

```
public void shouldExecuteWithNullParameterThrowAnException()
```

shouldGetMutationProbabilityReturnTheRightValue

```
public void shouldGetMutationProbabilityReturnTheRightValue()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided()
```

2.48.18 TwoPointCrossover

```
public class TwoPointCrossover<T> implements CrossoverOperator<Solution<T>>
    Created by FlapKap on 27-05-2017.
```

Fields**operator**

```
NPointCrossover<T> operator
```

Constructors**TwoPointCrossover**

```
public TwoPointCrossover (double probability)
```

Methods

execute

```
public List<Solution<T>> execute (List<Solution<T>> solutions)
```

getCrossoverProbability

```
public double getCrossoverProbability ()
```

getNumberOfGeneratedChildren

```
public int getNumberOfGeneratedChildren ()
```

getNumberOfRequiredParents

```
public int getNumberOfRequiredParents ()
```

2.49 org.uma.jmetal.operator.impl.localsearch

2.49.1 ArchiveMutationLocalSearch

```
public class ArchiveMutationLocalSearch<S extends Solution<?>> implements LocalSearchOperator<S>
```

This class implements a local search operator based in the use of a mutation operator. An archive is used to store the non-dominated solutions found during the search.

Author Antonio J. Nebro

Constructors

ArchiveMutationLocalSearch

```
public ArchiveMutationLocalSearch (int improvementRounds, MutationOperator<S> mutationOperator, Archive<S> archive, Problem<S> problem)
```

Constructor. Creates a new local search object.

Parameters

- **improvementRounds** – number of iterations
- **mutationOperator** – mutation operator
- **archive** – archive to store non-dominated solution
- **problem** – problem to resolve

Methods

execute

public S **execute** (S *solution*)
 Executes the local search.

Parameters

- **solution** – The solution to improve

Returns The improved solution

getEvaluations

public int **getEvaluations** ()
 Returns the number of evaluations

getNumberOfImprovements

public int **getNumberOfImprovements** ()

getNumberOfNonComparableSolutions

public int **getNumberOfNonComparableSolutions** ()

2.49.2 BasicLocalSearch

public class **BasicLocalSearch**<S extends Solution<?>> implements *LocalSearchOperator*<S>
 This class implements a basic local search operator based in the use of a mutation operator.

Author Antonio J. Nebro

Constructors

BasicLocalSearch

public **BasicLocalSearch** (int *improvementRounds*, *MutationOperator*<S> *mutationOperator*, *Comparator*<S> *comparator*, *Problem*<S> *problem*)
 Constructor. Creates a new local search object.

Parameters

- **improvementRounds** – number of iterations
- **mutationOperator** – mutation operator
- **comparator** – comparator to determine which solution is the best
- **problem** – problem to resolve

BasicLocalSearch

```
public BasicLocalSearch (int improvementRounds, MutationOperator<S> mutationOperator, Comparator<S> comparator, Problem<S> problem, RandomGenerator<Double> randomGenerator)
```

Constructor. Creates a new local search object.

Parameters

- **improvementRounds** – number of iterations
- **mutationOperator** – mutation operator
- **comparator** – comparator to determine which solution is the best
- **problem** – problem to resolve
- **randomGenerator** – the *RandomGenerator* to use when we must choose between equivalent solutions

Methods

execute

```
public S execute (S solution)
```

Executes the local search.

Parameters

- **solution** – The solution to improve

Returns An improved solution

getEvaluations

```
public int getEvaluations ()
```

Returns the number of evaluations

getNumberOfImprovements

```
public int getNumberOfImprovements ()
```

getNumberOfNonComparableSolutions

```
public int getNumberOfNonComparableSolutions ()
```

2.49.3 BasicLocalSearchTest

```
public class BasicLocalSearchTest
```

Methods

testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided

```
public void testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided ()
```

2.50 org.uma.jmetal.operator.impl.mutation

2.50.1 BitFlipMutation

```
public class BitFlipMutation implements MutationOperator<BinarySolution>
```

Author Antonio J. Nebro

Constructors

BitFlipMutation

```
public BitFlipMutation (double mutationProbability)  
    Constructor
```

BitFlipMutation

```
public BitFlipMutation (double mutationProbability, RandomGenerator<Double> randomGenerator)  
    Constructor
```

Methods

doMutation

```
public void doMutation (double probability, BinarySolution solution)  
    Perform the mutation operation
```

Parameters

- **probability** – Mutation setProbability
- **solution** – The solution to mutate

execute

```
public BinarySolution execute (BinarySolution solution)  
    Execute() method
```

getMutationProbability

```
public double getMutationProbability ()
```

setMutationProbability

```
public void setMutationProbability (double mutationProbability)
```

2.50.2 BitFlipMutationTest

```
public class BitFlipMutationTest
```

Methods

shouldConstructorAssignTheCorrectProbabilityValue

```
public void shouldConstructorAssignTheCorrectProbabilityValue ()
```

shouldConstructorFailWhenPassedANegativeProbabilityValue

```
public void shouldConstructorFailWhenPassedANegativeProbabilityValue ()
```

shouldExecuteWithNullParameterThrowAnException

```
public void shouldExecuteWithNullParameterThrowAnException ()
```

shouldGetMutationProbabilityReturnTheRightValue

```
public void shouldGetMutationProbabilityReturnTheRightValue ()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided ()
```

shouldMutateASingleVariableSolutionReturnTheSameSolutionIfNoBitsAreMutated

```
public void shouldMutateASingleVariableSolutionReturnTheSameSolutionIfNoBitsAreMutated ()
```

shouldMutateASingleVariableSolutionWhenASingleBitIsMutated

```
public void shouldMutateASingleVariableSolutionWhenASingleBitIsMutated ()
```

shouldMutateATwoVariableSolutionReturnTheSameSolutionIfNoBitsAreMutated

```
public void shouldMutateATwoVariableSolutionReturnTheSameSolutionIfNoBitsAreMutated ()
```


shouldMutateATwoVariableSolutionWhenTwoBitsAreMutated

```
public void shouldMutateATwoVariableSolutionWhenTwoBitsAreMutated()
```

2.50.3 CDGMutation

```
public class CDGMutation implements MutationOperator<DoubleSolution>
```

This class implements a polynomial mutation operator. The implementation is based on the NSGA-II code available in <http://www.iitk.ac.in/kangal/codes.shtml>. If the lower and upper bounds of a variable are the same, no mutation is carried out and the bound value is returned.

Author Feng Zhang

Constructors**CDGMutation**

```
public CDGMutation()
```

Constructor

CDGMutation

```
public CDGMutation(DoubleProblem problem, double delta)
```

Constructor

CDGMutation

```
public CDGMutation(double mutationProbability, double delta)
```

Constructor

CDGMutation

```
public CDGMutation(double mutationProbability, double delta, RepairDoubleSolution solutionRepair)
```

Constructor

Methods**execute**

```
public DoubleSolution execute(DoubleSolution solution)
```

Execute() method

getDelta

```
public double getDelta()
```

getMutationProbability

```
public double getMutationProbability()
```

setDelta

```
public void setDelta(double delta)
```

setMutationProbability

```
public void setMutationProbability(double probability)
```

2.50.4 IntegerPolynomialMutation

```
public class IntegerPolynomialMutation implements MutationOperator<IntegerSolution>
```

This class implements a polynomial mutation operator to be applied to Integer solutions. If the lower and upper bounds of a variable are the same, no mutation is carried out and the bound value is returned. A *RepairDoubleSolution* object is used to decide the strategy to apply when a value is out of range.

Author Antonio J. Nebro

Constructors

IntegerPolynomialMutation

```
public IntegerPolynomialMutation()  
    Constructor
```

IntegerPolynomialMutation

```
public IntegerPolynomialMutation(IntegerProblem problem, double distributionIndex)  
    Constructor
```

IntegerPolynomialMutation

```
public IntegerPolynomialMutation(double mutationProbability, double distributionIndex)  
    Constructor
```

IntegerPolynomialMutation

```
public IntegerPolynomialMutation(double mutationProbability, double distributionIndex, Repair-DoubleSolution solutionRepair)  
    Constructor
```

IntegerPolynomialMutation

```
public IntegerPolynomialMutation(double mutationProbability, double distributionIndex, RepairDoubleSolution solutionRepair, RandomGenerator<Double> randomGenerator)
```

Constructor

Methods

execute

```
public IntegerSolution execute(IntegerSolution solution)
```

Execute() method

getDistributionIndex

```
public double getDistributionIndex()
```

getMutationProbability

```
public double getMutationProbability()
```

setDistributionIndex

```
public void setDistributionIndex(double distributionIndex)
```

setMutationProbability

```
public void setMutationProbability(double mutationProbability)
```

2.50.5 IntegerPolynomialMutationTest

```
public class IntegerPolynomialMutationTest
```

Methods

shouldConstructorAssignTheCorrectDistributionIndex

```
public void shouldConstructorAssignTheCorrectDistributionIndex()
```

shouldConstructorAssignTheCorrectProbabilityValue

```
public void shouldConstructorAssignTheCorrectProbabilityValue()
```

shouldConstructorFailWhenPassedANegativeDistributionIndex

```
public void shouldConstructorFailWhenPassedANegativeDistributionIndex ()
```

shouldConstructorFailWhenPassedANegativeProbabilityValue

```
public void shouldConstructorFailWhenPassedANegativeProbabilityValue ()
```

shouldConstructorWithProblemAndDistributionIndexParametersAssignTheCorrectValues

```
public void shouldConstructorWithProblemAndDistributionIndexParametersAssignTheCorrectValues ()
```

shouldConstructorWithoutParameterAssignTheDefaultValues

```
public void shouldConstructorWithoutParameterAssignTheDefaultValues ()
```

shouldExecuteWithNullParameterThrowAnException

```
public void shouldExecuteWithNullParameterThrowAnException ()
```

shouldGetDistributionIndexReturnTheRightValue

```
public void shouldGetDistributionIndexReturnTheRightValue ()
```

shouldGetMutationProbabilityReturnTheRightValue

```
public void shouldGetMutationProbabilityReturnTheRightValue ()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided ()
```

shouldMutateASingleVariableSolutionReturnAValidSolution

```
public void shouldMutateASingleVariableSolutionReturnAValidSolution ()
```

shouldMutateASingleVariableSolutionReturnAnotherValidSolution

```
public void shouldMutateASingleVariableSolutionReturnAnotherValidSolution ()
```

shouldMutateASingleVariableSolutionReturnTheSameSolutionIfItIsNotMutated

```
public void shouldMutateASingleVariableSolutionReturnTheSameSolutionIfItIsNotMutated ()
```

shouldMutateASingleVariableSolutionReturnTheSameSolutionIfProbabilityIsZero

```
public void shouldMutateASingleVariableSolutionReturnTheSameSolutionIfProbabilityIsZero ()
```

shouldMutateASingleVariableSolutionWithSameLowerAndUpperBoundsReturnTheBoundValue

```
public void shouldMutateASingleVariableSolutionWithSameLowerAndUpperBoundsReturnTheBoundValue ()
```

2.50.6 NonUniformMutation

```
public class NonUniformMutation implements MutationOperator<DoubleSolution>
```

This class implements a non-uniform mutation operator.

Author Antonio J. Nebro , Juan J. Durillo

Constructors**NonUniformMutation**

```
public NonUniformMutation (double mutationProbability, double perturbation, int maxIterations)
```

Constructor

NonUniformMutation

```
public NonUniformMutation (double mutationProbability, double perturbation, int maxIterations, RandomGenerator<Double> randomGenenerator)
```

Constructor

Methods**doMutation**

```
public void doMutation (double probability, DoubleSolution solution)
```

Perform the mutation operation

Parameters

- **probability** – Mutation setProbability
- **solution** – The solution to mutate

execute

```
public DoubleSolution execute (DoubleSolution solution)
```

Execute() method

getCurrentIteration

```
public int getCurrentIteration ()
```

getMaxIterations

```
public int getMaxIterations ()
```

getMutationProbability

```
public double getMutationProbability ()
```

getPerturbation

```
public double getPerturbation ()
```

setCurrentIteration

```
public void setCurrentIteration (int currentIteration)
```

setMaxIterations

```
public void setMaxIterations (int maxIterations)
```

setMutationProbability

```
public void setMutationProbability (double mutationProbability)
```

setPerturbation

```
public void setPerturbation (double perturbation)
```

2.50.7 NonUniformMutationTest

```
public class NonUniformMutationTest
```

Methods

testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided

```
public void testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided ()
```

2.50.8 NullMutation

```
public class NullMutation<S> implements MutationOperator<S>
```

This class is intended to perform no mutation. It can be useful when configuring a genetic algorithm and we want to use only crossover.

Author Antonio J. Nebro

Methods

execute

```
public S execute (S source)
    Execute() method
```

2.50.9 PermutationSwapMutation

public class **PermutationSwapMutation**<T> implements *MutationOperator*<*PermutationSolution*<T>>
 This class implements a swap mutation. The solution type of the solution must be Permutation.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

PermutationSwapMutation

```
public PermutationSwapMutation (double mutationProbability)
    Constructor
```

PermutationSwapMutation

```
public PermutationSwapMutation (double mutationProbability, RandomGenerator<Double> random-
    Constructor                                     Generator)
```

PermutationSwapMutation

```
public PermutationSwapMutation (double mutationProbability, RandomGenerator<Double> mutation-
    Constructor                                     RandomGenerator, BoundedRandomGenerator<Integer> position-
    RandomGenerator)
```

Methods

doMutation

```
public void doMutation (PermutationSolution<T> solution)
    Performs the operation
```

execute

```
public PermutationSolution<T> execute (PermutationSolution<T> solution)
```

getMutationProbability

```
public double getMutationProbability ()
```

setMutationProbability

```
public void setMutationProbability (double mutationProbability)
```

2.50.10 PermutationSwapMutationTest

```
public class PermutationSwapMutationTest
```

Methods

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided ()
```

2.50.11 PolynomialMutation

```
public class PolynomialMutation implements MutationOperator<DoubleSolution>
```

This class implements a polynomial mutation operator. The implementation is based on the NSGA-II code available in <http://www.iitk.ac.in/kangal/codes.shtml>. If the lower and upper bounds of a variable are the same, no mutation is carried out and the bound value is returned.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

PolynomialMutation

```
public PolynomialMutation ()  
    Constructor
```

PolynomialMutation

```
public PolynomialMutation (DoubleProblem problem, double distributionIndex)  
    Constructor
```

PolynomialMutation

```
public PolynomialMutation (DoubleProblem problem, double distributionIndex, RandomGenerator  
                           <Double> randomGenerator)  
    Constructor
```

PolynomialMutation

```
public PolynomialMutation (double mutationProbability, double distributionIndex)  
    Constructor
```


PolynomialMutation

```
public PolynomialMutation(double mutationProbability, double distributionIndex, RandomGenerator<Double> randomGenerator)
```

Constructor

PolynomialMutation

```
public PolynomialMutation(double mutationProbability, double distributionIndex, RepairDoubleSolution solutionRepair)
```

Constructor

PolynomialMutation

```
public PolynomialMutation(double mutationProbability, double distributionIndex, RepairDoubleSolution solutionRepair, RandomGenerator<Double> randomGenerator)
```

Constructor

Methods

execute

```
public DoubleSolution execute (DoubleSolution solution)
```

Execute() method

getDistributionIndex

```
public double getDistributionIndex ()
```

getMutationProbability

```
public double getMutationProbability ()
```

setDistributionIndex

```
public void setDistributionIndex (double distributionIndex)
```

setMutationProbability

```
public void setMutationProbability (double probability)
```

2.50.12 PolynomialMutationTest

```
public class PolynomialMutationTest
```

Note: this class does check that the polynomial mutation operator does not return invalid values, but not that it works properly (@see PolynomialMutationWorkingTest)

Author Antonio J. Nebro

Methods

shouldConstructorAssignTheCorrectDistributionIndex

```
public void shouldConstructorAssignTheCorrectDistributionIndex()
```

shouldConstructorAssignTheCorrectProbabilityValue

```
public void shouldConstructorAssignTheCorrectProbabilityValue()
```

shouldConstructorFailWhenPassedANegativeDistributionIndex

```
public void shouldConstructorFailWhenPassedANegativeDistributionIndex()
```

shouldConstructorFailWhenPassedANegativeProbabilityValue

```
public void shouldConstructorFailWhenPassedANegativeProbabilityValue()
```

shouldConstructorWithProblemAndDistributionIndexParametersAssignTheCorrectValues

```
public void shouldConstructorWithProblemAndDistributionIndexParametersAssignTheCorrectValues()
```

shouldConstructorWithoutParameterAssignTheDefaultValues

```
public void shouldConstructorWithoutParameterAssignTheDefaultValues()
```

shouldExecuteWithNullParameterThrowAnException

```
public void shouldExecuteWithNullParameterThrowAnException()
```

shouldGetDistributionIndexReturnTheRightValue

```
public void shouldGetDistributionIndexReturnTheRightValue()
```

shouldGetMutationProbabilityReturnTheRightValue

```
public void shouldGetMutationProbabilityReturnTheRightValue()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided()
```

shouldMutateASingleVariableSolutionReturnAValidSolution

```
public void shouldMutateASingleVariableSolutionReturnAValidSolution()
```

shouldMutateASingleVariableSolutionReturnAnotherValidSolution

```
public void shouldMutateASingleVariableSolutionReturnAnotherValidSolution()
```

shouldMutateASingleVariableSolutionReturnTheSameSolutionIfItIsNotMutated

```
public void shouldMutateASingleVariableSolutionReturnTheSameSolutionIfItIsNotMutated()
```

shouldMutateASingleVariableSolutionReturnTheSameSolutionIfProbabilityIsZero

```
public void shouldMutateASingleVariableSolutionReturnTheSameSolutionIfProbabilityIsZero()
```

shouldMutateASingleVariableSolutionWithSameLowerAndUpperBoundsReturnTheBoundValue

```
public void shouldMutateASingleVariableSolutionWithSameLowerAndUpperBoundsReturnTheBoundValue()
```

2.50.13 SimpleRandomMutation

```
public class SimpleRandomMutation implements MutationOperator<DoubleSolution>
```

This class implements a random mutation operator for double solutions

Author Antonio J. Nebro

Constructors**SimpleRandomMutation**

```
public SimpleRandomMutation (double probability)
```

Constructor

SimpleRandomMutation

```
public SimpleRandomMutation (double probability, RandomGenerator<Double> randomGenerator)
```

Constructor

Methods**execute**

```
public DoubleSolution execute (DoubleSolution solution)
```

Execute() method

getMutationProbability

public double **getMutationProbability**()

setMutationProbability

public void **setMutationProbability**(double *mutationProbability*)

2.50.14 SimpleRandomMutationTest

public class **SimpleRandomMutationTest**

Methods

testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided

public void **testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided**()

2.50.15 UniformMutation

public class **UniformMutation** implements *MutationOperator<DoubleSolution>*

This class implements a uniform mutation operator.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

UniformMutation

public **UniformMutation**(double *mutationProbability*, double *perturbation*)
Constructor

UniformMutation

public **UniformMutation**(double *mutationProbability*, double *perturbation*, *RandomGenerator<Double>*
randomGenenerator)
Constructor

Methods

doMutation

public void **doMutation**(double *probability*, *DoubleSolution* *solution*)
Perform the operation

Parameters

- **probability** – Mutation setProbability

- **solution** – The solution to mutate

execute

public *DoubleSolution* **execute** (*DoubleSolution* solution)
Execute() method

getMutationProbability

public *Double* **getMutationProbability** ()

getPerturbation

public double **getPerturbation** ()

setMutationProbability

public void **setMutationProbability** (*Double* mutationProbability)

setPerturbation

public void **setPerturbation** (*Double* perturbation)

2.50.16 UniformMutationTest

public class **UniformMutationTest**

Methods

testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided

public void **testJMetalRandomGeneratorNotUsedWhenCustomRandomGeneratorProvided** ()

2.51 org.uma.jmetal.operator.impl.selection

2.51.1 BestSolutionSelection

public class **BestSolutionSelection**<S> implements *SelectionOperator*<List<S>, S>

This class implements a selection operator used for selecting the best solution in a list according to a given comparator.

Author Antonio J. Nebro

Constructors

BestSolutionSelection

```
public BestSolutionSelection (Comparator<S> comparator)
```

Methods

execute

```
public S execute (List<S> solutionList)  
    Execute() method
```

2.51.2 BinaryTournamentSelection

```
public class BinaryTournamentSelection<S extends Solution<?>> extends TournamentSelection<S>  
    Applies a binary tournament selection to return the best solution between two that have been chosen at random  
    from a solution list. Modified by Juanjo in 13.03.2015. A binary tournament is now a TournamentSelection  
    with 2 tournaments
```

Author Antonio J. Nebro, Juan J. Durillo

Constructors

BinaryTournamentSelection

```
public BinaryTournamentSelection ()  
    Constructor
```

BinaryTournamentSelection

```
public BinaryTournamentSelection (Comparator<S> comparator)  
    Constructor
```

2.51.3 BinaryTournamentSelectionTest

```
public class BinaryTournamentSelectionTest
```

Author Antonio J. Nebro

Methods

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty ()
```

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull ()
```

shouldExecuteReturnAValidSolutionIsWithCorrectParameters

```
public void shouldExecuteReturnAValidSolutionIsWithCorrectParameters ()
```

shouldExecuteReturnTheSameSolutionIfTheListContainsOneSolution

```
public void shouldExecuteReturnTheSameSolutionIfTheListContainsOneSolution ()
```

shouldExecuteReturnTwoSolutionsIfTheListContainsTwoSolutions

```
public void shouldExecuteReturnTwoSolutionsIfTheListContainsTwoSolutions ()
```

shouldExecuteWorkProperlyIfTheTwoSolutionsInTheListAreNondominated

```
public void shouldExecuteWorkProperlyIfTheTwoSolutionsInTheListAreNondominated ()
```

tearDown

```
public void tearDown ()
```

2.51.4 DifferentialEvolutionSelection

```
public class DifferentialEvolutionSelection implements SelectionOperator<List<DoubleSolution>, List<DoubleSolution>
```

Class implementing the selection operator used in DE: three different solutions are returned from a population.

The three solutions must be also different from the one indicated by an index (its position in the list). As a consequence, the operator requires a solution list with at least for elements.

Author Antonio J. Nebro , Juan J. Durillo

Constructors**DifferentialEvolutionSelection**

```
public DifferentialEvolutionSelection ()
```

Constructor

DifferentialEvolutionSelection

```
public DifferentialEvolutionSelection (BoundedRandomGenerator<Integer> randomGenerator)
```

Constructor

Methods

execute

```
public List<DoubleSolution> execute (List<DoubleSolution> solutionSet)
    Execute() method
```

setIndex

```
public void setIndex (int index)
```

2.51.5 DifferentialEvolutionSelectionTest

```
public class DifferentialEvolutionSelectionTest
    Created by ajnebro on 3/5/15.
```

Fields

exception

```
public ExpectedException exception
```

Methods

shouldExecuteRaiseAnExceptionIfTheIndexIsHigherThanTheSolutionListLength

```
public void shouldExecuteRaiseAnExceptionIfTheIndexIsHigherThanTheSolutionListLength ()
```

shouldExecuteRaiseAnExceptionIfTheIndexIsNegative

```
public void shouldExecuteRaiseAnExceptionIfTheIndexIsNegative ()
```

shouldExecuteRaiseAnExceptionIfTheIndexIsNotIndicated

```
public void shouldExecuteRaiseAnExceptionIfTheIndexIsNotIndicated ()
```

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsHasOneSolution

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsHasOneSolution ()
```

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty ()
```


shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull ()
```

shouldExecuteReturnThreeDifferentSolutionsIfTheListHasFourElements

```
public void shouldExecuteReturnThreeDifferentSolutionsIfTheListHasFourElements ()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided ()
```

2.51.6 NaryRandomSelection

```
public class NaryRandomSelection<S> implements SelectionOperator<List<S>, List<S>>
```

This class implements a random selection operator used for selecting a N number of solutions from a list

Author Antonio J. Nebro

Constructors

NaryRandomSelection

```
public NaryRandomSelection ()  
    Constructor
```

NaryRandomSelection

```
public NaryRandomSelection (int numberOfSolutionsToBeReturned)  
    Constructor
```

Methods

execute

```
public List<S> execute (List<S> solutionList)  
    Execute() method
```

2.51.7 NaryRandomSelectionTest

```
public class NaryRandomSelectionTest
```

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldDefaultConstructorReturnASingleSolution

public void **shouldDefaultConstructorReturnASingleSolution**()

shouldExecuteRaiseAnExceptionIfTheListSizelsOneAndTwoSolutionsAreRequested

public void **shouldExecuteRaiseAnExceptionIfTheListSizeIsOneAndTwoSolutionsAreRequested**()

shouldExecuteRaiseAnExceptionIfTheListSizelsTwoAndFourSolutionsAreRequested

public void **shouldExecuteRaiseAnExceptionIfTheListSizeIsTwoAndFourSolutionsAreRequested**()

shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty

public void **shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty**()

shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull

public void **shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull**()

shouldExecuteReturnTheCorrectNumberOfSolutions

public void **shouldExecuteReturnTheCorrectNumberOfSolutions**()

shouldExecuteReturnTheSolutionInTheListIfTheListContainsASolution

public void **shouldExecuteReturnTheSolutionInTheListIfTheListContainsASolution**()

shouldExecuteReturnTheSolutionSInTheListIfTheListContainsTwoSolutions

public void **shouldExecuteReturnTheSolutionSInTheListIfTheListContainsTwoSolutions**()

shouldNonDefaultConstructorReturnTheCorrectNumberOfSolutions

public void **shouldNonDefaultConstructorReturnTheCorrectNumberOfSolutions**()

shouldSelectNRandomDifferentSolutionsReturnTheCorrectListOfSolutions

public void **shouldSelectNRandomDifferentSolutionsReturnTheCorrectListOfSolutions** ()
 If the list contains 4 solutions, the result list must return all of them

2.51.8 NaryTournamentSelection

public class **NaryTournamentSelection**<S extends Solution<?>> implements *SelectionOperator*<List<S>, S>
 Applies a N-ary tournament selection to return the best solution between N that have been chosen at random from a solution list.

Author Antonio J. Nebro

Constructors

NaryTournamentSelection

public **NaryTournamentSelection** ()
 Constructor

NaryTournamentSelection

public **NaryTournamentSelection** (int *numberOfSolutionsToBeReturned*, *Comparator*<S> *comparator*)
 Constructor

Methods

execute

public S **execute** (*List*<S> *solutionList*)

2.51.9 NaryTournamentSelectionTest

public class **NaryTournamentSelectionTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldDefaultConstructorSetTheNumberOfSolutionsToBeReturnedEqualsToTwo

public void **shouldDefaultConstructorSetTheNumberOfSolutionsToBeReturnedEqualsToTwo** ()

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty()
```

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull()
```

shouldExecuteRaiseAnExceptionIfTheListSizelsOneAndTwoSolutionsAreRequested

```
public void shouldExecuteRaiseAnExceptionIfTheListSizeIsOneAndTwoSolutionsAreRequested()
```

shouldExecuteReturnAValidSolutionIsWithCorrectParameters

```
public void shouldExecuteReturnAValidSolutionIsWithCorrectParameters()
```

shouldExecuteReturnTheSameSolutionIfTheListContainsOneSolution

```
public void shouldExecuteReturnTheSameSolutionIfTheListContainsOneSolution()
```

shouldExecuteReturnTwoSolutionsIfTheListContainsTwoSolutions

```
public void shouldExecuteReturnTwoSolutionsIfTheListContainsTwoSolutions()
```

2.51.10 RandomSelection

```
public class RandomSelection<S> implements SelectionOperator<List<S>, S>
```

This class implements a random selection operator used for selecting a N number of solutions from a list

Author Antonio J. Nebro

Methods

execute

```
public S execute (List<S> solutionList)  
    Execute() method
```

2.51.11 RandomSelectionTest

```
public class RandomSelectionTest  
    Created by ajnebro on 4/5/15.
```

2.51.12 RankingAndCrowdingSelection

public class **RankingAndCrowdingSelection**<S extends Solution<?>> implements *SelectionOperator*<List<S>, List<S>>

This class implements a selection for selecting a number of solutions from a solution list. The solutions are taken by mean of its ranking and crowding distance values.

Author Antonio J. Nebro, Juan J. Durillo

Constructors

RankingAndCrowdingSelection

public **RankingAndCrowdingSelection** (int *solutionsToSelect*, *Comparator*<S> *dominanceComparator*)

Constructor

RankingAndCrowdingSelection

public **RankingAndCrowdingSelection** (int *solutionsToSelect*)

Constructor

Methods

addLastRankedSolutionsToPopulation

protected void **addLastRankedSolutionsToPopulation** (*Ranking*<S> *ranking*, int *rank*, List<S> *population*)

addRankedSolutionsToPopulation

protected void **addRankedSolutionsToPopulation** (*Ranking*<S> *ranking*, int *rank*, List<S> *population*)

crowdingDistanceSelection

protected List<S> **crowdingDistanceSelection** (*Ranking*<S> *ranking*)

execute

public List<S> **execute** (List<S> *solutionList*)

Execute() method

getNumberOfSolutionsToSelect

public int **getNumberOfSolutionsToSelect** ()

subfrontFillsIntoThePopulation

protected boolean **subfrontFillsIntoThePopulation** (*Ranking*<S> ranking, int rank, *List*<S> population)

2.51.13 RankingAndCrowdingSelectionTest

public class **RankingAndCrowdingSelectionTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldDefaultConstructorReturnASingleSolution

public void **shouldDefaultConstructorReturnASingleSolution** ()

shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty

public void **shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty** ()

shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull

public void **shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull** ()

shouldNonDefaultConstructorReturnTheCorrectNumberOfSolutions

public void **shouldNonDefaultConstructorReturnTheCorrectNumberOfSolutions** ()

2.51.14 RankingAndPreferenceSelection

public class **RankingAndPreferenceSelection**<S extends Solution<?>> implements *SelectionOperator*<*List*<S>, *List*<S>>>

Constructors

RankingAndPreferenceSelection

public **RankingAndPreferenceSelection** (int solutionsToSelect, *List*<*Double*> interestPoint, double epsilon)

Constructor

Methods

addLastRankedSolutionsToPopulation

protected void **addLastRankedSolutionsToPopulation** (*Ranking*<S> ranking, int rank, *List*<S> population)

addRankedSolutionsToPopulation

protected void **addRankedSolutionsToPopulation** (*Ranking*<S> ranking, int rank, *List*<S> population)

execute

public *List*<S> **execute** (*List*<S> solutionList)

getNumberOfSolutionsToSelect

public int **getNumberOfSolutionsToSelect** ()

preferenceDistanceSelection

protected *List*<S> **preferenceDistanceSelection** (*Ranking*<S> ranking, int numberOfObjectives)

subfrontFillsIntoThePopulation

protected boolean **subfrontFillsIntoThePopulation** (*Ranking*<S> ranking, int rank, *List*<S> population)

2.51.15 TournamentSelection

public class **TournamentSelection**<S extends *Solution*<?>> implements *SelectionOperator*<*List*<S>, S>

Author Juanjo

Constructors

TournamentSelection

public **TournamentSelection** (int numberOfTournaments)
Constructor

TournamentSelection

public **TournamentSelection** (*Comparator*<S> comparator, int numberOfTournaments)
Constructor

Methods

execute

```
public S execute (List<S> solutionList)
```

2.51.16 TournamentSelectionTest

```
public class TournamentSelectionTest  
    Created by ajnebro on 3/5/15.
```

Fields

exception

```
public ExpectedException exception
```

Methods

shouldConstructorAssignTheCorrectValueSToTheNumberOfTournamentsAndTheComparator

```
public void shouldConstructorAssignTheCorrectValueSToTheNumberOfTournamentsAndTheComparator ()
```

shouldConstructorAssignTheCorrectValueToTheNumberOfTournaments

```
public void shouldConstructorAssignTheCorrectValueToTheNumberOfTournaments ()
```

shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty

```
public void shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty ()
```

shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull ()
```

shouldExecuteReturnAnElementIfTheListHasOneElement

```
public void shouldExecuteReturnAnElementIfTheListHasOneElement ()
```

2.52 org.uma.jmetal.problem

2.52.1 BinaryProblem

```
public interface BinaryProblem extends Problem<BinarySolution>  
    Interface representing binary problems
```


Author Antonio J. Nebro

Methods

getNumberOfBits

public int **getNumberOfBits** (int *index*)

getTotalNumberOfBits

public int **getTotalNumberOfBits** ()

2.52.2 ConstrainedProblem

public interface **ConstrainedProblem**<S> extends *Problem*<S>

Interface representing problems having constraints

Author Antonio J. Nebro

Methods

evaluateConstraints

public void **evaluateConstraints** (S *solution*)

getNumberOfConstraints

public int **getNumberOfConstraints** ()

2.52.3 DoubleBinaryProblem

public interface **DoubleBinaryProblem**<S> extends *Problem*<S>

Interface representing problems having integer and double variables

Author Antonio J. Nebro

Methods

getLowerBound

public *Number* **getLowerBound** (int *index*)

getNumberOfBits

public int **getNumberOfBits** ()

getNumberOfDoubleVariables

```
public int getNumberOfDoubleVariables ()
```

getUpperBound

```
public Number getUpperBound (int index)
```

2.52.4 DoubleProblem

```
public interface DoubleProblem extends Problem<DoubleSolution>
```

Interface representing continuous problems

Author Antonio J. Nebro

Methods

getLowerBound

```
Double getLowerBound (int index)
```

getUpperBound

```
Double getUpperBound (int index)
```

2.52.5 IntegerDoubleProblem

```
public interface IntegerDoubleProblem<S> extends Problem<S>
```

Interface representing problems having integer and double variables

Author Antonio J. Nebro

Methods

getLowerBound

```
public Number getLowerBound (int index)
```

getNumberOfDoubleVariables

```
public int getNumberOfDoubleVariables ()
```

getNumberOfIntegerVariables

```
public int getNumberOfIntegerVariables ()
```

getUpperBound

public [Number](#) **getUpperBound** (int *index*)

2.52.6 IntegerProblem

public interface **IntegerProblem** extends [Problem](#)<[IntegerSolution](#)>

Interface representing integer problems

Author Antonio J. Nebro

Methods

getLowerBound

public [Integer](#) **getLowerBound** (int *index*)

getUpperBound

public [Integer](#) **getUpperBound** (int *index*)

2.52.7 PermutationProblem

public interface **PermutationProblem**<S> extends [PermutationSolution](#)<?>> extends [Problem](#)<S>

Interface representing permutation problems

Author Antonio J. Nebro

Methods

getPermutationLength

public int **getPermutationLength** ()

2.52.8 Problem

public interface **Problem**<S> extends [Serializable](#)

Interface representing a multi-objective optimization problem

Author Antonio J. Nebro

Parameters

- <S> – Encoding

Methods

createSolution

S **createSolution** ()

evaluate

void **evaluate** (*S solution*)

getName

String **getName** ()

getNumberOfConstraints

int **getNumberOfConstraints** ()

getNumberOfObjectives

int **getNumberOfObjectives** ()

getNumberOfVariables

int **getNumberOfVariables** ()

2.53 org.uma.jmetal.problem.impl

2.53.1 AbstractBinaryProblem

public abstract class **AbstractBinaryProblem** extends *AbstractGenericProblem<BinarySolution>* implements *BinaryProblem*

Methods

createSolution

public *BinarySolution* **createSolution** ()

getBitsPerVariable

protected abstract int **getBitsPerVariable** (int *index*)

getNumberOfBits

public int **getNumberOfBits** (int *index*)

getTotalNumberOfBits

public int **getTotalNumberOfBits** ()

2.53.2 AbstractDoubleProblem

public abstract class **AbstractDoubleProblem** extends *AbstractGenericProblem<DoubleSolution>* implements *DoubleProblem*

Methods

createSolution

public *DoubleSolution* **createSolution** ()

getLowerBound

public *Double* **getLowerBound** (int *index*)

getUpperBound

public *Double* **getUpperBound** (int *index*)

setLowerLimit

protected void **setLowerLimit** (*List<Double>* *lowerLimit*)

setUpperLimit

protected void **setUpperLimit** (*List<Double>* *upperLimit*)

2.53.3 AbstractGenericProblem

public abstract class **AbstractGenericProblem<S>** implements *Problem<S>*

Methods

getName

public *String* **getName** ()

getNumberOfConstraints

public int **getNumberOfConstraints** ()

getNumberOfObjectives

public int **getNumberOfObjectives** ()

getNumberOfVariables

```
public int getNumberOfVariables ()
```

setName

```
protected void setName (String name)
```

setNumberOfConstraints

```
protected void setNumberOfConstraints (int numberOfConstraints)
```

setNumberOfObjectives

```
protected void setNumberOfObjectives (int numberOfObjectives)
```

setNumberOfVariables

```
protected void setNumberOfVariables (int numberOfVariables)
```

2.53.4 AbstractIntegerDoubleProblem

```
public abstract class AbstractIntegerDoubleProblem<S> extends AbstractGenericProblem<S> implements IntegerDoublePro
```

Methods

getLowerBound

```
public Number getLowerBound (int index)
```

getNumberOfDoubleVariables

```
public int getNumberOfDoubleVariables ()
```

getNumberOfIntegerVariables

```
public int getNumberOfIntegerVariables ()
```

getUpperBound

```
public Number getUpperBound (int index)
```

setLowerLimit

protected void **setLowerLimit** (*List<Number> lowerLimit*)

setNumberOfDoubleVariables

protected void **setNumberOfDoubleVariables** (int *numberOfDoubleVariables*)

setNumberOfIntegerVariables

protected void **setNumberOfIntegerVariables** (int *numberOfIntegerVariables*)

setUpperLimit

protected void **setUpperLimit** (*List<Number> upperLimit*)

2.53.5 AbstractIntegerPermutationProblem

public abstract class **AbstractIntegerPermutationProblem** extends *AbstractGenericProblem<PermutationSolution<Integer>*

Methods**createSolution**

public *PermutationSolution<Integer>* **createSolution** ()

2.53.6 AbstractIntegerProblem

public abstract class **AbstractIntegerProblem** extends *AbstractGenericProblem<IntegerSolution>* implements *IntegerProblem*

Methods**createSolution**

public *IntegerSolution* **createSolution** ()

getLowerBound

public *Integer* **getLowerBound** (int *index*)

getUpperBound

public *Integer* **getUpperBound** (int *index*)

setLowerLimit

protected void **setLowerLimit** (*List<Integer> lowerLimit*)

setUpperLimit

protected void **setUpperLimit** (*List<Integer> upperLimit*)

2.54 org.uma.jmetal.problem.multiobjective

2.54.1 Binh2

public class **Binh2** extends *AbstractDoubleProblem* implements *ConstrainedProblem<DoubleSolution>*
Class representing problem Binh2

Fields**numberOfViolatedConstraints**

public *NumberOfViolatedConstraints<DoubleSolution>* **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation<DoubleSolution>* **overallConstraintViolationDegree**

Constructors**Binh2**

public **Binh2** ()
Constructor Creates a default instance of the Binh2 problem

Methods**evaluate**

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution solution*)
EvaluateConstraints() method

2.54.2 ConstrEx

public class **ConstrEx** extends *AbstractDoubleProblem* implements *ConstrainedProblem<DoubleSolution>*
Class representing problem ConstrEx

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints<DoubleSolution>* **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation<DoubleSolution>* **overallConstraintViolationDegree**

Constructors

ConstrEx

public **ConstrEx** ()
Constructor Creates a default instance of the ConstrEx problem

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)
EvaluateConstraints() method

2.54.3 Fonseca

public class **Fonseca** extends *AbstractDoubleProblem*
Class representing problem Fonseca

Constructors

Fonseca

public **Fonseca** ()
Constructor

Methods

evaluate

```
public void evaluate (DoubleSolution solution)
    Evaluate() method
```

2.54.4 FourBarTruss

```
public class FourBarTruss extends AbstractDoubleProblem
    Class representing problem FourBarTruss Measures: f = 10kN e = 200000 kN/cm2 l = 200 cm sigma = 10kN/cm2
```

Constructors

FourBarTruss

```
public FourBarTruss ()
    Constructor Creates a default instance of the FourBarTruss problem
```

Methods

evaluate

```
public void evaluate (DoubleSolution solution)
    Evaluate() method
```

2.54.5 Golinski

```
public class Golinski extends AbstractDoubleProblem implements ConstrainedProblem<DoubleSolution>
    Class representing problem Golinski.
```

Fields

numberOfViolatedConstraints

```
public NumberOfViolatedConstraints<DoubleSolution> numberOfViolatedConstraints
```

overallConstraintViolationDegree

```
public OverallConstraintViolation<DoubleSolution> overallConstraintViolationDegree
```

Constructors

Golinski

public **Golinski** ()
Constructor. Creates a default instance of the Golinski problem.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)
EvaluateConstraints() method

2.54.6 Kursawe

public class **Kursawe** extends *AbstractDoubleProblem*
Class representing problem Kursawe

Constructors

Kursawe

public **Kursawe** ()
Constructor. Creates a default instance (3 variables) of the Kursawe problem

Kursawe

public **Kursawe** (*Integer* numberOfVariables)
Constructor. Creates a new instance of the Kursawe problem.

Parameters

- **numberOfVariables** – Number of variables of the problem

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.54.7 MultiobjectiveTSP

public class **MultiobjectiveTSP** extends *AbstractIntegerPermutationProblem*

Class representing a bi-objective TSP (Traveling Salesman Problem) problem. It accepts data files from TSPLIB: <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/>

Fields

costMatrix

protected double[][] **costMatrix**

distanceMatrix

protected double[][] **distanceMatrix**

numberOfCities

protected int **numberOfCities**

Constructors

MultiobjectiveTSP

public **MultiobjectiveTSP** (*String distanceFile*, *String costFile*)
Creates a new MultiobjectiveTSP problem instance

Methods

evaluate

public void **evaluate** (*PermutationSolution*<*Integer*> *solution*)
Evaluate() method

getPermutationLength

public int **getPermutationLength** ()

2.54.8 NMMin

public class **NMMin** extends *AbstractIntegerProblem*

Created by Antonio J. Nebro on 03/07/14. Bi-objective problem for testing integer encoding. Objective 1: minimizing the distance to value N Objective 2: minimizing the distance to value M

Constructors

NMMin

public **NMMin** ()

NMMin

public **NMMin** (int *numberOfVariables*, int *n*, int *m*, int *lowerBound*, int *upperBound*)
Constructor

Methods

evaluate

public void **evaluate** (*IntegerSolution* *solution*)
Evaluate() method

2.54.9 NMMin2

public class **NMMin2** extends *AbstractIntegerDoubleProblem*<*IntegerDoubleSolution*>
Created by Antonio J. Nebro on 18/09/14. Bi-objective problem for testing integer/double encoding. Objective 1: minimizing the distance to value N Objective 2: minimizing the distance to value M

Constructors

NMMin2

public **NMMin2** ()

NMMin2

public **NMMin2** (int *numberOfIntegerVariables*, int *numberOfDoubleVariables*, int *n*, int *m*, int *lowerBound*, int *upperBound*)
Constructor

Methods

createSolution

public *IntegerDoubleSolution* **createSolution** ()

evaluate

public void **evaluate** (*IntegerDoubleSolution* *solution*)
Evaluate() method

2.54.10 NMMinTest

public class **NMMinTest**

Created by Antonio J. Nebro on 17/09/14.

Fields

problem

Problem<IntegerSolution> **problem**

Methods

evaluateSimpleSolutions

public void **evaluateSimpleSolutions** ()

2.54.11 OneZeroMax

public class **OneZeroMax** extends *AbstractBinaryProblem*

Class representing problem OneZeroMax. The problem consist of maximizing the number of '1's and '0's in a binary string.

Constructors

OneZeroMax

public **OneZeroMax** ()

Constructor

OneZeroMax

public **OneZeroMax** (*Integer numberOfBits*)

Constructor

Methods

createSolution

public *BinarySolution* **createSolution** ()

evaluate

public void **evaluate** (*BinarySolution solution*)

Evaluate() method

getBitsPerVariable

protected int **getBitsPerVariable** (int *index*)

2.54.12 Osyczka2

public class **Osyczka2** extends *AbstractDoubleProblem* implements *ConstrainedProblem<DoubleSolution>*
 Class representing problem Oyczka2

Fields**numberOfViolatedConstraints**

public *NumberOfViolatedConstraints<DoubleSolution>* **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation<DoubleSolution>* **overallConstraintViolationDegree**

Constructors**Osyczka2**

public **Osyczka2** ()
 Constructor. Creates a default instance of the Osyczka2 problem.

Methods**evaluate**

public void **evaluate** (*DoubleSolution solution*)
 Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution solution*)
 EvaluateConstraints() method

2.54.13 Schaffer

public class **Schaffer** extends *AbstractDoubleProblem*
 Class representing problem Schaffer

Constructors

Schaffer

public **Schaffer** ()
Constructor. Creates a default instance of problem Schaffer

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.54.14 Srinivas

public class **Srinivas** extends *AbstractDoubleProblem* implements *ConstrainedProblem<DoubleSolution>*
Class representing problem Srinivas

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints<DoubleSolution>* **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation<DoubleSolution>* **overallConstraintViolationDegree**

Constructors

Srinivas

public **Srinivas** ()
Constructor

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)
EvaluateConstraints() method

2.54.15 Tanaka

public class **Tanaka** extends *AbstractDoubleProblem* implements *ConstrainedProblem<DoubleSolution>*
Class representing problem Tanaka

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints<DoubleSolution>* **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation<DoubleSolution>* **overallConstraintViolationDegree**

Constructors

Tanaka

public **Tanaka** ()
Constructor. Creates a default instance of the problem Tanaka

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)
EvaluateConstraints() method

2.54.16 Viennet2

public class **Viennet2** extends *AbstractDoubleProblem*
Class representing problem Viennet2

Constructors

Viennet2

public **Viennet2** ()
Constructor. Creates a default instance of the Viennet2 problem

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.54.17 Viennet3

public class **Viennet3** extends *AbstractDoubleProblem*
Class representing problem Viennet3

Constructors

Viennet3

public **Viennet3** ()
Constructor. Creates a default instance of the Viennet3 problem.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.54.18 Viennet4

public class **Viennet4** extends *AbstractDoubleProblem* implements *ConstrainedProblem*<*DoubleSolution*>
Class representing problem Viennet4

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints*<*DoubleSolution*> **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation*<*DoubleSolution*> **overallConstraintViolationDegree**

Constructors

Viennet4

public **Viennet4** ()
Constructor. Creates a default instance of the Viennet4 problem.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)
EvaluateConstraints() method

2.54.19 Water

public class **Water** extends *AbstractDoubleProblem* implements *ConstrainedProblem*<*DoubleSolution*>
Class representing problem Water

Fields

LOWERLIMIT

public static final *Double*[] **LOWERLIMIT**

UPPERLIMIT

public static final *Double*[] **UPPERLIMIT**

numberOfViolatedConstraints

public *NumberOfViolatedConstraints*<*DoubleSolution*> **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation*<*DoubleSolution*> **overallConstraintViolationDegree**

Constructors

Water

public **Water** ()
Constructor. Creates a default instance of the Water problem.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)
EvaluateConstraints() method

2.55 org.uma.jmetal.problem.multiobjective.UF

2.55.1 UF1

public class **UF1** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF1

Constructors

UF1

public **UF1** ()
Constructor. Creates a default instance of problem CEC2009_UF1 (30 decision variables)

UF1

public **UF1** (int *numberOfVariables*)
Creates a new instance of problem CEC2009_UF1.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.55.2 UF10

public class **UF10** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF10

Constructors

UF10

public **UF10** ()
Constructor. Creates a default instance of problem CEC2009_UF10 (30 decision variables)

UF10

public **UF10** (int *numberOfVariables*)
Creates a new instance of problem CEC2009_UF10.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.55.3 UF2

public class **UF2** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF2

Constructors

UF2

public **UF2** ()
Constructor. Creates a default instance of problem CEC2009_UF2 (30 decision variables)

UF2

public **UF2** (int *numberOfVariables*)
Creates a new instance of problem CEC2009_UF2.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.55.4 UF3

public class **UF3** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF3

Constructors

UF3

public **UF3** ()
Constructor. Creates a default instance of problem CEC2009_UF3 (30 decision variables)

UF3

public **UF3** (int *numberOfVariables*)
Creates a new instance of problem CEC2009_UF3.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* *solution*)
Evaluate() method

2.55.5 UF4

public class **UF4** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF4

Constructors

UF4

public **UF4** ()
Constructor. Creates a default instance of problem CEC2009_UF4 (30 decision variables)

UF4

public **UF4** (*Integer* *numberOfVariables*)
Creates a new instance of problem CEC2009_UF4.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.55.6 UF5

public class **UF5** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF5

Fields

epsilon

double **epsilon**

n

int **n**

Constructors

UF5

public **UF5** ()
Constructor. Creates a default instance of problem CEC2009_UF5 (30 decision variables)

UF5

public **UF5** (int *numberOfVariables*, int *N*, double *epsilon*)
Creates a new instance of problem CEC2009_UF5.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.55.7 UF6

public class **UF6** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF5

Fields

epsilon

double **epsilon**

n

int **n**

Constructors

UF6

public **UF6** ()
Constructor. Creates a default instance of problem CEC2009_UF6 (30 decision variables, N =10, epsilon = 0.1)

UF6

public **UF6** (*Integer numberOfVariables*, int *N*, double *epsilon*)
Creates a new instance of problem CEC2009_UF6.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.55.8 UF7

public class **UF7** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF7

Constructors

UF7

public **UF7** ()
 Constructor. Creates a default instance of problem CEC2009_UF7 (30 decision variables)

UF7

public **UF7** (int *numberOfVariables*)
 Creates a new instance of problem CEC2009_UF7.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
 Evaluate() method

2.55.9 UF8

public class **UF8** extends *AbstractDoubleProblem*
 Class representing problem CEC2009_UF8

Constructors

UF8

public **UF8** ()
 Constructor. Creates a default instance of problem CEC2009_UF8 (30 decision variables)

UF8

public **UF8** (int *numberOfVariables*)
 Creates a new instance of problem CEC2009_UF8.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
 Evaluate() method

2.55.10 UF9

public class **UF9** extends *AbstractDoubleProblem*
Class representing problem CEC2009_UF9

Fields

epsilon

double **epsilon**

Constructors

UF9

public **UF9** ()
Constructor. Creates a default instance of problem CEC2009_UF9 (30 decision variables, epsilon = 0.1)

UF9

public **UF9** (int *numberOfVariables*, double *epsilon*)
Creates a new instance of problem CEC2009_UF9.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* *solution*)
Evaluate() method

2.56 org.uma.jmetal.problem.multiobjective.cdtlz

2.56.1 C1_DTLZ1

public class **C1_DTLZ1** extends *DTLZ1* implements *ConstrainedProblem*<*DoubleSolution*>
Problem C1-DTLZ1, defined in: Jain, H. and K. Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach.” *EEE Transactions on Evolutionary Computation*, 18(4):602-622, 2014.

Author Antonio J. Nebro

Fields

numberOfViolatedConstraints

```
public NumberOfViolatedConstraints<DoubleSolution> numberOfViolatedConstraints
```

overallConstraintViolationDegree

```
public OverallConstraintViolation<DoubleSolution> overallConstraintViolationDegree
```

Constructors

C1_DTLZ1

```
public C1_DTLZ1 (int numberOfVariables, int numberOfObjectives)
    Constructor
```

Parameters

- **numberOfVariables** –
- **numberOfObjectives** –

Methods

evaluateConstraints

```
public void evaluateConstraints (DoubleSolution solution)
```

2.56.2 C1_DTLZ3

```
public class C1_DTLZ3 extends DTLZ3 implements ConstrainedProblem<DoubleSolution>
    Problem C1-DTLZ3, defined in: Jain, H. and K. Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach.” EEE Transactions on Evolutionary Computation, 18(4):602-622, 2014.
```

Author Antonio J. Nebro

Fields

numberOfViolatedConstraints

```
public NumberOfViolatedConstraints<DoubleSolution> numberOfViolatedConstraints
```

overallConstraintViolationDegree

```
public OverallConstraintViolation<DoubleSolution> overallConstraintViolationDegree
```

Constructors

C1_DTLZ3

public **C1_DTLZ3** (int *numberOfVariables*, int *numberOfObjectives*)
Constructor

Parameters

- **numberOfVariables** –
- **numberOfObjectives** –

Methods

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)

2.56.3 C2_DTLZ2

public class **C2_DTLZ2** extends *DTLZ2* implements *ConstrainedProblem*<*DoubleSolution*>

Problem C2-DTLZ2, defined in: Jain, H. and K. Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach.” *EEE Transactions on Evolutionary Computation*, 18(4):602-622, 2014.

Author Antonio J. Nebro

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints*<*DoubleSolution*> **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation*<*DoubleSolution*> **overallConstraintViolationDegree**

Constructors

C2_DTLZ2

public **C2_DTLZ2** (int *numberOfVariables*, int *numberOfObjectives*)
Constructor

Parameters

- **numberOfVariables** –
- **numberOfObjectives** –

Methods

evaluateConstraints

```
public void evaluateConstraints (DoubleSolution solution)
```

2.56.4 C3_DTLZ1

```
public class C3_DTLZ1 extends DTLZ1 implements ConstrainedProblem<DoubleSolution>
```

Problem C3-DTLZ1, defined in: Jain, H. and K. Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach.” *EEE Transactions on Evolutionary Computation*, 18(4):602-622, 2014.

Author Antonio J. Nebro

Fields

numberOfViolatedConstraints

```
public NumberOfViolatedConstraints<DoubleSolution> numberOfViolatedConstraints
```

overallConstraintViolationDegree

```
public OverallConstraintViolation<DoubleSolution> overallConstraintViolationDegree
```

Constructors

C3_DTLZ1

```
public C3_DTLZ1 (int numberOfVariables, int numberOfObjectives, int numberOfConstraints)
```

Constructor

Parameters

- **numberOfVariables** –
- **numberOfObjectives** –

Methods

evaluateConstraints

```
public void evaluateConstraints (DoubleSolution solution)
```

2.56.5 C3_DTLZ4

```
public class C3_DTLZ4 extends DTLZ4 implements ConstrainedProblem<DoubleSolution>
```

Problem C3-DTLZ4, defined in: Jain, H. and K. Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach.” *EEE Transactions on Evolutionary Computation*, 18(4):602-622, 2014.

Author Antonio J. Nebro

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints*<*DoubleSolution*> **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation*<*DoubleSolution*> **overallConstraintViolationDegree**

Constructors

C3_DTLZ4

public **C3_DTLZ4** (int *numberOfVariables*, int *numberOfObjectives*, int *numberOfConstraints*)
Constructor

Parameters

- **numberOfVariables** –
- **numberOfObjectives** –

Methods

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* *solution*)

2.56.6 ConvexC2_DTLZ2

public class **ConvexC2_DTLZ2** extends *DTLZ2* implements *ConstrainedProblem*<*DoubleSolution*>
Problem ConvexC2-DTLZ2, defined in: Jain, H. and K. Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach.” *EEE Transactions on Evolutionary Computation*, 18(4):602-622, 2014.

Author Antonio J. Nebro

Fields

numberOfViolatedConstraints

public *NumberOfViolatedConstraints*<*DoubleSolution*> **numberOfViolatedConstraints**

overallConstraintViolationDegree

public *OverallConstraintViolation*<*DoubleSolution*> **overallConstraintViolationDegree**

Constructors

ConvexC2_DTLZ2

public **ConvexC2_DTLZ2** (int *numberOfVariables*, int *numberOfObjectives*)
 Constructor

Parameters

- *numberOfVariables* –
- *numberOfObjectives* –

Methods

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* *solution*)

2.57 org.uma.jmetal.problem.multiobjective.cec2015OptBigDataCompetition

2.57.1 BigOpt2015

public class **BigOpt2015** extends *AbstractDoubleProblem*
 Created by ajnebro on 14/1/15.

Fields

dTypeG

int **dTypeG**

f1max

double **f1max**

f1min

double **f1min**

f2max

double **f2max**

f2min

double **f2min**

scaling

boolean **scaling**

Constructors

BigOpt2015

public **BigOpt2015** (*String instanceName*)
Constructor

Methods

correlation

List<List<Double>> **correlation** (*List<List<Double>> list1*, *List<List<Double>> list2*)

diagonal1

double **diagonal1** (*List<List<Double>> list*)

diagonal2

double **diagonal2** (*List<List<Double>> list*)

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

multiplyWithoutAMP

List<List<Double>> **multiplyWithoutAMP** (*List<List<Double>> list1*, *List<List<Double>> list2*)

newMeanStandardDeviation

List<Double> **newMeanStandardDeviation** (*List<Double> list*)

vectorCorrelation

double **vectorCorrelation** (*List<Double> list1*, *List<Double> list2*)

2.58 org.uma.jmetal.problem.multiobjective.dtlz

2.58.1 DTLZ1

public class **DTLZ1** extends *AbstractDoubleProblem*
Class representing problem DTLZ1

Constructors

DTLZ1

public **DTLZ1** ()
Creates a default DTLZ1 problem (7 variables and 3 objectives)

DTLZ1

public **DTLZ1** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a DTLZ1 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.58.2 DTLZ2

public class **DTLZ2** extends *AbstractDoubleProblem*
Class representing problem DTLZ1

Constructors

DTLZ2

public **DTLZ2** ()
Creates a default DTLZ2 problem (12 variables and 3 objectives)

DTLZ2

public **DTLZ2** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a DTLZ2 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.58.3 DTLZ3

public class **DTLZ3** extends *AbstractDoubleProblem*
Class representing problem DTLZ3

Constructors

DTLZ3

public **DTLZ3** ()
Creates a default DTLZ3 problem (12 variables and 3 objectives)

DTLZ3

public **DTLZ3** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a DTLZ3 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.58.4 DTLZ4

public class **DTLZ4** extends *AbstractDoubleProblem*
 Class representing problem DTLZ4

Constructors

DTLZ4

public **DTLZ4** ()
 Creates a default DTLZ4 problem (12 variables and 3 objectives)

DTLZ4

public **DTLZ4** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
 Creates a DTLZ4 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
 Evaluate() method

2.58.5 DTLZ5

public class **DTLZ5** extends *AbstractDoubleProblem*
 Class representing problem DTLZ5

Constructors

DTLZ5

public **DTLZ5** ()
 Creates a default DTLZ5 problem (12 variables and 3 objectives)

DTLZ5

public **DTLZ5** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
 Creates a DTLZ5 problem instance

Parameters

- **numberOfVariables** – Number of variables

- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.58.6 DTLZ6

public class **DTLZ6** extends *AbstractDoubleProblem*
Class representing problem DTLZ6

Constructors

DTLZ6

public **DTLZ6** ()
Creates a default DTLZ6 problem (12 variables and 3 objectives)

DTLZ6

public **DTLZ6** (*Integer* numberOfVariables, *Integer* numberOfObjectives)
Creates a DTLZ6 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.58.7 DTLZ7

public class **DTLZ7** extends *AbstractDoubleProblem*
Class representing problem DTLZ7

Constructors

DTLZ7

public **DTLZ7** ()
 Creates a default DTLZ7 problem (22 variables and 3 objectives)

DTLZ7

public **DTLZ7** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
 Creates a DTLZ7 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
 Evaluate() method

2.59 org.uma.jmetal.problem.multiobjective.ebes

2.59.1 Ebes

public class **Ebes** extends *AbstractDoubleProblem* implements *ConstrainedProblem<DoubleSolution>*
 Class representing problem Ebes Spatial Bars Structure (Estructuras de Barras Espaciales)

Fields

AREA

int **AREA**

ART_ART

int **ART_ART**

ART_RIG

int **ART_RIG**

AxialForcei_

protected double[] **AxialForcei_**
Stores the Axial force in node i

AxialForcej_

protected double[] **AxialForcej_**
Stores the Axial force in node j

Ay_

int **Ay_**

Az_

int **Az_**

BETA

int **BETA**

BLijY

int **BLijY_**

BLijZ

int **BLijZ_**

CARGA_MOMENTO_DISTRIBUIDO

int **CARGA_MOMENTO_DISTRIBUIDO**

CARGA_MOMENTO_PUNTUAL

int **CARGA_MOMENTO_PUNTUAL**

CARGA_PARABOLICA

int **CARGA_PARABOLICA**

CARGA_PUNTUAL

int **CARGA_PUNTUAL**

CARGA_TEMPERATURA

int **CARGA_TEMPERATURA**

CARGA_TRIANGULAR_I

int **CARGA_TRIANGULAR_I**

CARGA_UNIFORME_PARCIAL

int **CARGA_UNIFORME_PARCIAL**

CARGA_UNIFORME_TOTAL

int **CARGA_UNIFORME_TOTAL**

CARGA__TRIANGULAR_J

int **CARGA__TRIANGULAR_J**

CIRCLE

public static final int **CIRCLE**

COMPRESSION

int **COMPRESSION**

CONSTRAINT

int **CONSTRAINT**

DESCRIPTION

int **DESCRIPTION**

DisplacementNodes_

protected double[][] **DisplacementNodes_**
Stores the k displacement

ELONGATION_NEG

int **ELONGATION_NEG**

ELONGATION_POS

int **ELONGATION_POS**

E

int **E_**

Efforti_

protected double[][] **Efforti_**
Stores the Effort in node i

Effortj_

protected double[][] **Effortj_**
Stores the Effort in node j

Ei

int **Ei_**

Ej

int **Ej_**

Element

protected double[] **Element_**
Stores the Element

Fyz

int **Fyz_**

GROUP

int **GROUP_**

G_

int **G_**

GravitationalAxis

String **GravitationalAxis_**

Groups

protected double[][] **Groups_**
Stores the Groups

HOLE_CIRCLE

public static final int **HOLE_CIRCLE**

HOLE_RECTANGLE

public static final int **HOLE_RECTANGLE**

H_DOUBLE

public static final int **H_DOUBLE**

H_SINGLE

public static final int **H_SINGLE**

INDEX

int **INDEX_**

I_DOUBLE

public static final int **I_DOUBLE**

I_SINGLE

public static final int **I_SINGLE**

It

int **It_**

lw

int **Iw_**

ly

int **Iy_**

lz

int **Iz_**

KGii

double[][] **KGii**

KGij

double[][] **KGij**

KGji

double[][] **KGji**

KGjj

double[][] **KGjj**

Kii

double[][] **Kii**

KiiSOG

double[][] **KiiSOG**

Kij

double[][] **Kij**

KijSOG

double[][] **KijSOG**

Kji

```
double[][] Kji
```

KjiSOG

```
double[][] KjiSOG
```

Kjj

```
double[][] Kjj
```

KjjSOG

```
double[][] KjjSOG
```

L

```
int L_
```

L_DOUBLE

```
public static final int L_DOUBLE
```

L_SINGLE

```
public static final int L_SINGLE
```

Li

```
int Li_
```

Lj

```
int Lj_
```

MAX_COLUMN

```
int MAX_COLUMN
```

MatrixStiffness_

```
protected double[] MatrixStiffness_  
    Stores the k
```

MxyMax

protected double[][] **MxyMax_**
Stores the max Mxy for groups

MxyMin

protected double[][] **MxyMin_**
Stores the min Mxy for groups

MxzMax

protected double[][] **MxzMax_**
Stores the max Mxz for groups

MxzMin

protected double[][] **MxzMin_**
Stores the max Mxz for groups

NodeRestrict

protected double[][] **NodeRestrict_**
Stores the NodeRestrict

Node

protected double[][] **Node_**
Stores the Node

NxxMax

protected double[][] **NxxMax_**
Stores the max Nxx for groups

NxxMin

protected double[][] **NxxMin_**
Stores the min Nxx for groups

OF

String[] **OF_**

OldStrainMax

protected double[][] **OldStrainMax_**
Stores the max Strain for elements

OldStrainMin

protected double[][] **OldStrainMin_**

OverloadInElement

protected double[][] **OverloadInElement_**
Stores the OverLoad on Elements

PQ

double[][] **PQ**

QAx

int **QAx_**

QAy

int **QAy_**

QAz

int **QAz_**

QE

int **QE_**

QH

int **QH_**

QT

int **QT_**

Qa

int **Qa_**

Qb

int **Qb_**

Qi

double[] **Qi**

Qj

double[] **Qj**

RATIO_YZ

int **RATIO_YZ**

RECTANGLE

public static final int **RECTANGLE**

RIG_ART

int **RIG_ART**

RIG_RIG

int **RIG_RIG**

RTij

double[][] **RTij**

RTji

double[][] **RTji**

Reaction

double **Reaction_**

Rij

double[][] **Ri***j*

Rji

double[][] **Rj***i*

RpTij

double[][] **RpT***i**j*

RpTji

double[][] **RpT***j**i*

Rpij

double[][] **Rp***i**j*

Rpji

double[][] **Rp***j**i*

SHAPE

int **SHAPE**

SPECIFIC_WEIGHT

int **SPECIFIC_WEIGHT**

STRAIN_COMPRESS

int **STRAIN_COMPRESS**

STRAIN_CUT

int **STRAIN_CUT**

STRAIN_TRACTION

int **STRAIN_TRACTION**

STRESS

int **STRESS**

STRESS_CUT

int **STRESS_CUT**

StrainCutMax

protected double[][] **StrainCutMax_**
Stores the max Strain for elements

StrainMax

protected double[][] **StrainMax_**
Stores the max Strain for elements

StrainMin

protected double[][] **StrainMin_**

StrainMxyMax

protected double[][] **StrainMxyMax_**
Stores the max Mxz Strain for groups

StrainMxyMin

protected double[][] **StrainMxyMin_**
Stores the max Mxz Strain for groups

StrainMxzMax

protected double[][] **StrainMxzMax_**
Stores the max Mxz Strain for groups

StrainMxzMin

protected double[][] **StrainMxzMin_**
Stores the max Mxz Strain for groups

StrainNxxMax

protected double[][] **StrainNxxMax_**
 Stores the max Nxx Strain for groups

StrainNxxMin

protected double[][] **StrainNxxMin_**

StrainResidualCut

protected double[] **StrainResidualCut_**
 Stores the Cut Strain Residual for elements

StrainResidualMax

protected double[] **StrainResidualMax_**
 Stores the max Strain for elements

StrainResidualMin

protected double[] **StrainResidualMin_**
 Stores the min Strain for elements

Straini_

protected double[][][] **Straini_**
 Stores the Strain in node i

Strainj

protected double[][][] **Strainj_**
 Stores the Strain in node j

T_DOUBLE

public static final int **T_DOUBLE**

T_SINGLE

public static final int **T_SINGLE**

TypeMaterial

int **TypeMaterial_**

VARIABLES

int **VARIABLES**

VAR_POSITION

int **VAR_POSITION**

VAR_Y_LOWER_LIMIT

int **VAR_Y_LOWER_LIMIT**

VAR_Y_UPPER_LIMIT

int **VAR_Y_UPPER_LIMIT**

VAR_Z_LOWER_LIMIT

int **VAR_Z_LOWER_LIMIT**

VAR_Z_UPPER_LIMIT

int **VAR_Z_UPPER_LIMIT**

VAR_eY_LOWER_LIMIT

int **VAR_eY_LOWER_LIMIT**

VAR_eY_UPPER_LIMIT

int **VAR_eY_UPPER_LIMIT**

VAR_eZ_LOWER_LIMIT

int **VAR_eZ_LOWER_LIMIT**

VAR_eZ_UPPER_LIMIT

int **VAR_eZ_UPPER_LIMIT**

Vij

int **Vij_**

WeightElement

protected double[][] **WeightElement_**
Stores the Load on Elements Itself

WeightNode

protected double[][] **WeightNode_**
Stores the Load on Nodes

Y

int **Y_**

Z

int **Z_**

aX

int **aX_**

aY_

int **aY_**

aZ_

int **aZ_**

cbi

double[][][] **cbi**

cbj

double[][][] **cbj**

dY

int **dY_**

eY

int **eY_**

eZ

int **eZ_**

elementsBetweenDiffGreat

protected int **elementsBetweenDiffGreat_**
Stores the Elements Between Difference Greatest

gX

int **gX_**

gY

int **gY_**

gZ

int **gZ_**

g_

double **g_**

geometryCheck_

protected int[][] **geometryCheck_**

i

int **i_**

j

int **j_**

IBuckling

public boolean **lBuckling**

ILoadsOwnWeight

public boolean **lLoadsOwnWeight**

ISecondOrderGeometric

public boolean **lSecondOrderGeometric**

lZ

int **lZ_**

matrixWidthBand

protected int **matrixWidthBand_**
Stores the number a wide the diagonal matrix

nodeCheck_

protected double[][] **nodeCheck_**
Stores the number of Nodes of the problem

numberOfConstraintsGeometric

public int **numberOfConstraintsGeometric_**

numberOfConstraintsNodes

protected int **numberOfConstraintsNodes_**

numberOfElements

protected int **numberOfElements_**
Stores the number of Bar of the problem

numberOfEval

protected int **numberOfEval_**
Stores the number of Bar Groups

numberOfGroupElements

protected int **numberOfGroupElements_**
Stores the number of Bar Groups

numberOfGroupsToCheckGeometry

protected int **numberOfGroupsToCheckGeometry_**

numberOfLibertyDegree

protected int **numberOfLibertyDegree_**
Stores the number of Nodes of the problem

numberOfNodes

protected int **numberOfNodes**

numberOfNodesRestricts_

protected int **numberOfNodesRestricts_**

numberOfWeigthHypothesis

protected int **numberOfWeigthHypothesis_**

numberOfWeightsElements

protected int **numberOfWeightsElements_**
Stores the number of Load in ElementsNodes of the problem

numberOfWeightsNodes

protected int **numberOfWeightsNodes_**
Stores the number of Load in Nodes of the problem

omegaMax

protected double[][] **omegaMax_**
Stores the max omega for groups

overallConstraintViolationDegree

public *OverallConstraintViolation*<*DoubleSolution*> **overallConstraintViolationDegree**

pi

double[] **pi**

pj

double[] **pj**

rZ

int **rZ_**

selectedOF

int **selectedOF**

strainAdmissibleCut

protected int **strainAdmissibleCut_**

uY

int **uY_**

Constructors

Ebes

public **Ebes** ()

Ebes

public **Ebes** (String *ebesFileName*, String[] *objectiveList*)
 Constructor

Throws

- **FileNotFoundException** –

Methods

AxialForcei_

public double **AxialForcei_** (int *element*)

AxialForcej_

public double **AxialForcej_** (int *element*)

BucklingOmega

public double **BucklingOmega** (double *Nxx*, double[] *G*, double[] *B*)

DisplacementNodes

public double **DisplacementNodes** (int *node*, int *hi*)

EBEsAssignAxialForces

public void **EBEsAssignAxialForces** (int *hi*)

EBEsCalculus

public void **EBEsCalculus** ()

EBEsEcuationSolution

public void **EBEsEcuationSolution** (int *hi*)

EBEsEffortsElements3D

public void **EBEsEffortsElements3D** (int *hi*, int *countIter*, double[][] *Slip*)

EBEsEffortsTotal3D

public void **EBEsEffortsTotal3D** (int *hi*)

EBEsElementsTopology

public void **EBEsElementsTopology** (*DoubleSolution* *solution*)

EBEsInitialize

public void **EBEsInitialize** (*String* *file*)

EBEsMat3DG

public void **EBEsMat3DG** (int *e*)

EBEsMat3DGij

public void **EBEsMat3DGij** ()

EBEsMat3DL_SOG

```
public void EBEsMat3DL_SOG (int e)
```

EBEsMat3DL_iArt_jArt

```
public void EBEsMat3DL_iArt_jArt (int e)
```

EBEsMat3DL_iArt_jRig

```
public void EBEsMat3DL_iArt_jRig (int e)
```

EBEsMat3DL_iRig_jArt

```
public void EBEsMat3DL_iRig_jArt (int e)
```

EBEsMat3DL_iRig_jRig

```
public void EBEsMat3DL_iRig_jRig (int e)
```

EBEsMatRot3DLaG

```
public void EBEsMatRot3DLaG (int e)
```

EBEsMatRot3DLpSaL

```
public void EBEsMatRot3DLpSaL (int e)
```

EBEsMatrixAdd

```
public double[][] EBEsMatrixAdd (double[][] s, double[][] t)
```

EBEsMatrixGlobalFactory

```
public void EBEsMatrixGlobalFactory (int countIter)
```

EBEsMatrixGlobalPenalization

```
public void EBEsMatrixGlobalPenalization ()
```

EBEsMatrixSubtractions

```
public double[][] EBEsMatrixSubtractions (double[][] s, double[][] t)
```

EBEsMatrixWeight

```
public void EBEsMatrixWeight (int hi)
```

EBEsMatrizMultiplicar

```
public double[][] EBEsMatrizMultiplicar (double[][] s, double[][] t)
```

EBEsMatrizTraspuesta

```
public double[][] EBEsMatrizTraspuesta (double[][] m)
```

EBEsMatrizVectorMultiplicar

```
public double[] EBEsMatrizVectorMultiplicar (double[][] s, double[] t)
```

EBEsNodesEquilibrium3D

```
public void EBEsNodesEquilibrium3D (int hi)
```

EBEsOverloadWeightElement

```
public void EBEsOverloadWeightElement ()
```

EBEsPrintArchTxtDesp

```
public void EBEsPrintArchTxtDesp (int hi)
```

EBEsPrintArchTxtEfforts

```
public void EBEsPrintArchTxtEfforts (int hi)
```

EBEsPrintArchTxtElements

```
public void EBEsPrintArchTxtElements ()
```

EBEsPrintArchTxtMKG

```
public void EBEsPrintArchTxtMKG (String s, int hi)
```

EBEsPrintArchTxtMKLB

```
public void EBEsPrintArchTxtMKLB (int e)
```

EBEsPrintArchTxtReaction

```
public void EBEsPrintArchTxtReaction (int hi)
```

EBEsPrintArchTxtStrain

```
public void EBEsPrintArchTxtStrain ()
```

EBEsReactions3D

```
public void EBEsReactions3D (int hi)
```

EBEsReadDataFile

```
public final void EBEsReadDataFile (String fileName)
```

EBEsReadProblems

```
public String EBEsReadProblems ()
```

EBEsSteelingResults

```
public void EBEsSteelingResults (int hi)
```

EBEsStrainMaxWhitElement

```
public void EBEsStrainMaxWhitElement ()
```

EBEsStrainMaxWhitGroup

```
public void EBEsStrainMaxWhitGroup ()
```

EBEsStrainMinWhitElement

```
public void EBEsStrainMinWhitElement ()
```

EBEsStrainMinWhitGroup

```
public void EBEsStrainMinWhitGroup ()
```

EBEsStrainNode

```
public double[][][] EBEsStrainNode (double[][][] E)
```

EBEsStrainResidualVerication

```
public void EBEsStrainResidualVerication()
```

EBEsTransversalSectionCircular

```
public void EBEsTransversalSectionCircular(int gr, double d)
```

EBEsTransversalSectionHoleCircular

```
public void EBEsTransversalSectionHoleCircular(int gr, double D, double e)
```

EBEsTransversalSectionHoleRectangle

```
public void EBEsTransversalSectionHoleRectangle(int gr, double y, double z, double ey, double  
                                                    ez)
```

EBEsTransversalSectionRectangle

```
public void EBEsTransversalSectionRectangle(int gr, double y, double z)
```

EBEsTransversalSection_H_Double

```
public void EBEsTransversalSection_H_Double(int gr, double y, double z, double ey, double ez)
```

EBEsTransversalSection_H_Single

```
public void EBEsTransversalSection_H_Single(int gr, double y, double z, double ey, double ez)
```

EBEsTransversalSection_I_Double

```
public void EBEsTransversalSection_I_Double(int gr, double y, double z, double ey, double ez)
```

EBEsTransversalSection_I_Single

```
public void EBEsTransversalSection_I_Single(int gr, double y, double z, double ey, double ez)
```

EBEsTransversalSection_L_Double

```
public void EBEsTransversalSection_L_Double(int gr, double y, double z, double ey, double ez)
```

EBEsTransversalSection_L_Single

```
public void EBEsTransversalSection_L_Single(int gr, double y, double z, double ey, double ez)
```

EBEsTransversalSection_T_Double

```
public void EBEsTransversalSection_T_Double (int ba, double y, double z, double ey, double ez)
```

EBEsTransversalSection_T_Single

```
public void EBEsTransversalSection_T_Single (int ba, double y, double z, double ey, double ez)
```

EBEsWeightDistributedUniformly

```
public void EBEsWeightDistributedUniformly (int el, double[] LoadInElement_)
```

EBEsWeightNodes

```
public void EBEsWeightNodes ()
```

EBEsWeigthElement

```
public void EBEsWeigthElement ()
```

Efforti

```
public double Efforti (int i, int element, int hypothesis)
```

Effortj

```
public double Effortj (int i, int element, int hypothesis)
```

FunctionENS

```
public double FunctionENS (int hi)
```

FunctionsMahalanobis_Distance_With_Variance

```
public double FunctionsMahalanobis_Distance_With_Variance (int hi)
```

Interpolation_I_Single_Y_func_Area

```
public double Interpolation_I_Single_Y_func_Area_ (double A)
```

Interpolation_I_Single_Y_func_Wxy

```
public double Interpolation_I_Single_Y_func_Wxy_ (double Wxy)
```

Interpolation_I_Single_Y_func_Wxz

public double **Interpolation_I_Single_Y_func_Wxz_** (double *Wxz*)

Interpolation_I_Single_Z_func_Y

public double **Interpolation_I_Single_Z_func_Y_** (double *Y*)

Interpolation_I_Single_ey_func_Y

public double **Interpolation_I_Single_ey_func_Y_** (double *Y*)

Interpolation_I_Single_ez_func_Y

public double **Interpolation_I_Single_ez_func_Y_** (double *Y*)

MatrixStiffness

public double **MatrixStiffness** (int *i*)

Straini

public double **Straini** (int *i*, int *element*, int *hypothesis*)

Variable_Position

public int **Variable_Position** ()

createSolution

public *DoubleSolution* **createSolution** ()

evaluate

public void **evaluate** (*DoubleSolution* *solution*)

Evaluates a solution

Parameters

- **solution** – The solution to evaluate

evaluateConstraints

public void **evaluateConstraints** (*DoubleSolution* solution)

Evaluates the constraint overhead of a solution

Parameters

- **solution** – The solution

Throws

- *JMetalException* –

geometryCheck

public int **geometryCheck** (int *i*, int *j*)

getElement

public double **getElement** (int *i*, int *j*)

getElementsBetweenDiffGreat

public int **getElementsBetweenDiffGreat** ()

getGroupShape

public int **getGroupShape** (int *groupId*)

getGroups

public double **getGroups** (int *i*)

getMatrixWidthBand

public int **getMatrixWidthBand** ()

getMxyMax

public double **getMxyMax** (int *group*, int *hypothesis*)

getMxyMin

public double **getMxyMin** (int *group*, int *hypothesis*)

getMxzMax

public double **getMxzMax** (int *group*, int *hypothesis*)

getMxzMin

public double **getMxzMin** (int *group*, int *hypothesis*)

getNode

public double **getNode** (int *i*, int *j*)

getNodeRestrict

public double **getNodeRestrict** (int *i*, int *j*)

getNumberOfConstraintsNodes

public int **getNumberOfConstraintsNodes** ()

getNumberOfElements

public int **getNumberOfElements** ()

getNumberOfNodes

public int **getNumberOfNodes** ()

getNumberOfNodesRestricts

public int **getNumberOfNodesRestricts** ()

getNumberOfWeigthHypothesis

public int **getNumberOfWeigthHypothesis** ()

getNumberOfWeightsElements

public int **getNumberOfWeightsElements** ()

getNumberOfWeightsNodes

public int **getNumberOfWeightsNodes** ()

getNxxMax

public double **getNxxMax** (int *group*, int *hypothesis*)

getNxxMin

public double **getNxxMin** (int *group*, int *hypothesis*)

getOldStrainMax

public double **getOldStrainMax** (int *group*, int *hypothesis*)

getOldStrainMin

public double **getOldStrainMin** (int *group*, int *hypothesis*)

getOmegaMax

public double **getOmegaMax** (int *group*, int *hypothesis*)

getStrainAdmissibleCut

public int **getStrainAdmissibleCut** ()

getStrainCutMax

public double **getStrainCutMax** (int *group*, int *hypothesis*)

getStrainMax

public double **getStrainMax** (int *group*, int *hypothesis*)

getStrainMin

public double **getStrainMin** (int *group*, int *hypothesis*)

getStrainMxyMax

public double **getStrainMxyMax** (int *group*, int *hypothesis*)

getStrainMxyMin

public double **getStrainMxyMin** (int *group*, int *hypothesis*)

getStrainMxzMax

public double **getStrainMxzMax** (int *group*, int *hypothesis*)

getStrainMxzMin

public double **getStrainMxzMin** (int *group*, int *hypothesis*)

getStrainNxxMax

public double **getStrainNxxMax** (int *group*, int *hypothesis*)

getStrainNxxMin

public double **getStrainNxxMin** (int *group*, int *hypothesis*)

getStrainResidualCut

public double **getStrainResidualCut** (int *hypothesis*)

getStrainResidualMax

public double **getStrainResidualMax** (int *hypothesis*)

getStrainResidualMin

public double **getStrainResidualMin** (int *hypothesis*)

getStrainj

public double **getStrainj** (int *i*, int *element*, int *hypothesis*)

getVariablePosition

public int **getVariablePosition** (int *groupId*)

getWeightElement

public double **getWeightElement** (int *i*, int *j*)

getWeightElementItself

public double **getWeightElementItself** (int *i*, int *j*)

getWeightNode

public double **getWeightNode** (int *i*, int *j*)

getnumberOfConstraintsGeometric

public int **getnumberOfConstraintsGeometric** ()

getnumberOfGroupElements

public int **getnumberOfGroupElements** ()

nodeCheck

public double **nodeCheck** (int *i*, int *j*)

numberOfNodesRestricts

public void **numberOfNodesRestricts** (int *numberOfNodesRestricts*)

setElementsBetweenDiffGreat

public void **setElementsBetweenDiffGreat** (int *elementsBetweenDiffGreat*)

setMatrixWidthBand

public void **setMatrixWidthBand** (int *matrixWidthBand*)

setNumberOfConstraintsNodes

public void **setNumberOfConstraintsNodes** (int *numberOfConstraintsNodes*)

setNumberOfElements

public void **setNumberOfElements** (int *numberOfElements*)

setNumberOfNodes

public void **setNumberOfNodes** (int *numberOfNodes*)

setNumberOfWeigthHypothesis

public void **setNumberOfWeigthHypothesis** (int *numberOfWeigthHypothesis*)

setNumberOfWeightsElements

```
public void setNumberOfWeightsElements (int numberOfWeightsElements)
```

setNumberOfWeightsNodes

```
public void setNumberOfWeightsNodes (int numberOfWeightsNodes)
```

setStrainAdmissibleCut

```
public void setStrainAdmissibleCut (int strainAdmissibleCut)
```

setnumberOfConstraintsGeometric

```
public void setnumberOfConstraintsGeometric (int i)
```

setnumberOfGroupElements

```
public void setnumberOfGroupElements (int i)
```

2.60 org.uma.jmetal.problem.multiobjective.glt

2.60.1 GLT1

```
public class GLT1 extends AbstractDoubleProblem
```

Problem GLT1. Defined in F. Gu, H.-L. Liu, and K. C. Tan, “A multiobjective evolutionary algorithm using dynamic weight design method,” International Journal of Innovative Computing, Information and Control, vol. 8, no. 5B, pp. 3677–3688, 2012.

Author Antonio J. Nebro

Constructors

GLT1

```
public GLT1 ()  
    Default constructor
```

GLT1

```
public GLT1 (int numberOfVariables)  
    Constructor
```

Parameters

- **numberOfVariables** –

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

2.60.2 GLT2

public class **GLT2** extends *AbstractDoubleProblem*

Problem GLT2. Defined in F. Gu, H.-L. Liu, and K. C. Tan, “A multiobjective evolutionary algorithm using dynamic weight design method,” International Journal of Innovative Computing, Information and Control, vol. 8, no. 5B, pp. 3677–3688, 2012.

Author Antonio J. Nebro

Constructors

GLT2

public **GLT2** ()
Default constructor

GLT2

public **GLT2** (int *numberOfVariables*)
Constructor

Parameters

- **numberOfVariables** –

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

2.60.3 GLT3

public class **GLT3** extends *AbstractDoubleProblem*

Problem GLT3. Defined in F. Gu, H.-L. Liu, and K. C. Tan, “A multiobjective evolutionary algorithm using dynamic weight design method,” International Journal of Innovative Computing, Information and Control, vol. 8, no. 5B, pp. 3677–3688, 2012.

Author Antonio J. Nebro

Constructors

GLT3

public **GLT3** ()
Default constructor

GLT3

public **GLT3** (int *numberOfVariables*)
Constructor

Parameters

- **numberOfVariables** –

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

2.60.4 GLT4

public class **GLT4** extends *AbstractDoubleProblem*

Problem GLT4. Defined in F. Gu, H.-L. Liu, and K. C. Tan, “A multiobjective evolutionary algorithm using dynamic weight design method,” International Journal of Innovative Computing, Information and Control, vol. 8, no. 5B, pp. 3677–3688, 2012.

Author Antonio J. Nebro

Constructors

GLT4

public **GLT4** ()
Default constructor

GLT4

public **GLT4** (int *numberOfVariables*)
Constructor

Parameters

- **numberOfVariables** –

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

2.60.5 GLT5

public class **GLT5** extends *AbstractDoubleProblem*

Problem GLT5. Defined in F. Gu, H.-L. Liu, and K. C. Tan, “A multiobjective evolutionary algorithm using dynamic weight design method,” International Journal of Innovative Computing, Information and Control, vol. 8, no. 5B, pp. 3677–3688, 2012.

Author Antonio J. Nebro

Constructors

GLT5

public **GLT5** ()
Default constructor

GLT5

public **GLT5** (int *numberOfVariables*)
Constructor

Parameters

- **numberOfVariables** –

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

2.60.6 GLT6

public class **GLT6** extends *AbstractDoubleProblem*

Problem GLT6. Defined in F. Gu, H.-L. Liu, and K. C. Tan, “A multiobjective evolutionary algorithm using dynamic weight design method,” International Journal of Innovative Computing, Information and Control, vol. 8, no. 5B, pp. 3677–3688, 2012.

Author Antonio J. Nebro

Constructors

GLT6

public **GLT6** ()
Default constructor

GLT6

public **GLT6** (int *numberOfVariables*)
Constructor

Parameters

- **numberOfVariables** –

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

2.61 org.uma.jmetal.problem.multiobjective.lz09

2.61.1 LZ09

public class **LZ09**
Base class to implement the problem of the lz09 benchmark, which is defined in: H. Li and Q. Zhang. Multiobjective optimization problem with complicated pareto sets, MOEA/D and NSGA-II. IEEE Transactions on Evolutionary Computation, 12(2):284-302, April 2009.

Fields

dtype

int **dtype**

ltype

int **ltype**

nobj

int **nobj**

nvar

int **nvar**

ptype

int **ptype**

Constructors

LZ09

public **LZ09** (int *nvar*, int *nobj*, int *ptype*, int *dtype*, int *ltype*)

Methods

alphaFunction

void **alphaFunction** (double[] *alpha*, List<Double> *x*, int *dim*, int *type*)

betaFunction

double **betaFunction** (List<Double> *x*, int *type*)

objective

void **objective** (List<Double> *xVar*, List<Double> *yObj*)

psfunc2

double **psfunc2** (double *x*, double *t1*, int *dim*, int *type*, int *css*)

psfunc3

double **psfunc3** (double *x*, double *t1*, double *t2*, int *dim*, int *type*)

2.61.2 LZ09F1

public class **LZ09F1** extends *AbstractDoubleProblem*
Class representing problem LZ09F1

Constructors

LZ09F1

public **LZ09F1** ()
Creates a default LZ09F1 problem (10 variables and 2 objectives)

LZ09F1

public **LZ09F1** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F1 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.3 LZ09F2

public class **LZ09F2** extends *AbstractDoubleProblem*
Class representing problem LZ09F2

Constructors

LZ09F2

public **LZ09F2** ()
Creates a default LZ09F2 problem (30 variables and 3 objectives)

LZ09F2

public **LZ09F2** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F2 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.4 LZ09F3

public class **LZ09F3** extends *AbstractDoubleProblem*
Class representing problem LZ09F3

Constructors

LZ09F3

public **LZ09F3** ()
Creates a default LZ09F3 problem (30 variables and 2 objectives)

LZ09F3

public **LZ09F3** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F3 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.5 LZ09F4

public class **LZ09F4** extends *AbstractDoubleProblem*
Class representing problem LZ09F4

Constructors

LZ09F4

public **LZ09F4** ()
Creates a default LZ09F4 problem (30 variables and 2 objectives)

LZ09F4

public **LZ09F4** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F4 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.6 LZ09F5

public class **LZ09F5** extends *AbstractDoubleProblem*
Class representing problem LZ09F5

Constructors

LZ09F5

public **LZ09F5** ()
Creates a default LZ09F5 problem (30 variables and 2 objectives)

LZ09F5

public **LZ09F5** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F5 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.7 LZ09F6

public class **LZ09F6** extends *AbstractDoubleProblem*
Class representing problem LZ09F6

Constructors

LZ09F6

public **LZ09F6** ()
Creates a default LZ09F6 problem (30 variables and 2 objectives)

LZ09F6

public **LZ09F6** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F6 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.8 LZ09F7

public class **LZ09F7** extends *AbstractDoubleProblem*
Class representing problem LZ09F7

Constructors

LZ09F7

public **LZ09F7** ()
Creates a default LZ09F7 problem (10 variables and 2 objectives)

LZ09F7

public **LZ09F7** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F7 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.9 LZ09F8

public class **LZ09F8** extends *AbstractDoubleProblem*
Class representing problem LZ09F8

Constructors

LZ09F8

public **LZ09F8** ()
Creates a default LZ09F8 problem (10 variables and 2 objectives)

LZ09F8

public **LZ09F8** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F8 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.61.10 LZ09F9

public class **LZ09F9** extends *AbstractDoubleProblem*
Class representing problem LZ09F9

Constructors

LZ09F9

public **LZ09F9** ()
Creates a default LZ09F9 problem (30 variables and 2 objectives)

LZ09F9

public **LZ09F9** (*Integer ptype*, *Integer dtype*, *Integer ltype*)
Creates a LZ09F9 problem instance

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.62 org.uma.jmetal.problem.multiobjective.maf

2.62.1 MaF01

public class **MaF01** extends *AbstractDoubleProblem*
Class representing problem MaF01

Constructors

MaF01

public **MaF01** ()
Default constructor

MaF01

public **MaF01** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF01 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.2 MaF02

public class **MaF02** extends *AbstractDoubleProblem*
Class representing problem MaF02, DTLZ2BZ

Fields

const2

public static int **const2**

Constructors

MaF02

public **MaF02** ()
Default constructor

MaF02

public **MaF02** (*Integer* numberOfVariables, *Integer* numberOfObjectives)
Creates a MaF02 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.3 MaF03

public class **MaF03** extends *AbstractDoubleProblem*
Class representing problem MaF03, convex DTLZ3

Constructors

MaF03

public **MaF03** ()
Default constructor

MaF03

public **MaF03** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF03 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.4 MaF04

public class **MaF04** extends *AbstractDoubleProblem*
Class representing problem MaF04

Fields

const4

public static double **const4**

Constructors

MaF04

public **MaF04** ()
Default constructor

MaF04

public **MaF04** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF04 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.5 MaF05

public class **MaF05** extends *AbstractDoubleProblem*
Class representing problem MaF05

Fields

const5

public static double **const5**

Constructors

MaF05

public **MaF05** ()
Default constructor

MaF05

public **MaF05** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF05 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.6 MaF06

public class **MaF06** extends *AbstractDoubleProblem*
Class representing problem MaF06

Constructors

MaF06

public **MaF06** ()
Default constructor

MaF06

public **MaF06** (*Integer* numberOfVariables, *Integer* numberOfObjectives)
Creates a MaF06 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.7 MaF07

public class **MaF07** extends *AbstractDoubleProblem*
Class representing problem MaF07

Constructors

MaF07

public **MaF07** ()
Default constructor

MaF07

public **MaF07** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF07 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.8 MaF08

public class **MaF08** extends *AbstractDoubleProblem*
Class representing problem MaF08

Fields

const8

public static double **const8**

Constructors

MaF08

public **MaF08** ()
Default constructor

MaF08

public **MaF08** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF03 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

nextPoint

public static double[] **nextPoint** (double *arc*, double[] *startp*, double *r*)

polygonpoints

public static double[][] **polygonpoints** (int *m*, double *r*)

2.62.9 MaF09

public class **MaF09** extends *AbstractDoubleProblem*
Class representing problem MaF05

Fields

M9

public static int **M9**

maxinter9

public static int **maxinter9**

pindex9

public static int **pindex9**

points9

public static double **points9**

Constructors

MaF09

public **MaF09** ()
Default constructor

MaF09

public **MaF09** (*Integer* numberOfVariables, *Integer* numberOfObjectives)
Creates a MaF09 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

checkWithJdkGeneralPath

public static boolean **checkWithJdkGeneralPath** (*Point2D.Double* point, *List<Point2D.Double>* polygon)

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

generV

public static double **generV** (double lb, double ub)

if_infeasible

public static boolean **if_infeasible** (double[] x)

if_inside_polygon

public static boolean **if_inside_polygon** (double[] p1, double[][] points)

intersection

```
public static double[] intersection (double[] kb1, double[] kb2)
```

line_of_twoP

```
public static double[] line_of_twoP (double[] p1, double[] p2)
```

lines_of_polygon

```
public double[][] lines_of_polygon (double[][] p)
```

polygonpoints

```
public static double[][] polygonpoints (int m, double r)
```

2.62.10 MaF10

```
public class MaF10 extends AbstractDoubleProblem  
    Class representing problem MaF10
```

Fields

K10

```
public static int K10
```

Constructors

MaF10

```
public MaF10 ()  
    Default constructor
```

MaF10

```
public MaF10 (Integer numberOfVariables, Integer numberOfObjectives)  
    Creates a MaF10 problem instance
```

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.11 MaF11

public class **MaF11** extends *AbstractDoubleProblem*
Class representing problem MaF11

Fields

K11

public static int **K11**

Constructors

MaF11

public **MaF11** ()
Default constructor

MaF11

public **MaF11** (*Integer* numberOfVariables, *Integer* numberOfObjectives)
Creates a MaF11 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.12 MaF12

public class **MaF12** extends *AbstractDoubleProblem*
Class representing problem MaF12

Fields

K12

public static int **K12**

Constructors

MaF12

public **MaF12** ()
Default constructor

MaF12

public **MaF12** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
Creates a MaF12 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.13 MaF13

public class **MaF13** extends *AbstractDoubleProblem*
Class representing problem MaF13

Constructors

MaF13

public **MaF13** ()
Default constructor

MaF13

public **MaF13** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
 Creates a MaF13 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
 Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.62.14 MaF14

public class **MaF14** extends *AbstractDoubleProblem*
 Class representing problem MaF14

Fields

nk14

public static int **nk14**

sublen14

public static int **sublen14**

Constructors

MaF14

public **MaF14** ()
 Default constructor

MaF14

public **MaF14** (*Integer numberOfVariables*, *Integer numberOfObjectives*)
 Creates a MaF14 problem instance

Parameters

- **numberOfVariables** – Number of variables
- **numberOfObjectives** – Number of objective functions

Methods

Rastrigin

public static double **Rastrigin** (double[] *x*)

Rosenbrock

public static double **Rosenbrock** (double[] *x*)

evaluate

public void **evaluate** (*DoubleSolution* *solution*)

2.62.15 MaF15

public class **MaF15** extends *AbstractDoubleProblem*
Class representing problem MaF15

Fields

nk15

public static int **nk15**

sublen15

public static int **sublen15**

Constructors

MaF15

public **MaF15** ()
Default constructor

MaF15

public **MaF15** (*Integer* *numberOfVariables*, *Integer* *numberOfObjectives*)
Creates a MaF15 problem instance

Parameters

- **numberOfVariables** – Number of variables

- **numberOfObjectives** – Number of objective functions

Methods

Griewank

public static double **Griewank** (double[] *x*)

Sphere

public static double **Sphere** (double[] *x*)

evaluate

public void **evaluate** (*DoubleSolution* *solution*)

Evaluates a solution

Parameters

- **solution** – The solution to evaluate

2.63 org.uma.jmetal.problem.multiobjective.mop

2.63.1 MOP1

public class **MOP1** extends *AbstractDoubleProblem*

Problem MOP1. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP1

public **MOP1** ()

Constructor. Creates default instance of problem MOP1 (10 decision variables)

MOP1

public **MOP1** (*Integer* *numberOfVariables*)

Creates a new instance of problem MOP1.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.63.2 MOP2

public class **MOP2** extends *AbstractDoubleProblem*

Problem MOP2. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP2

public **MOP2** ()
Constructor. Creates default instance of problem MOP2 (10 decision variables)

MOP2

public **MOP2** (*Integer* numberOfVariables)
Creates a new instance of problem MOP2.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.63.3 MOP3

public class **MOP3** extends *AbstractDoubleProblem*

Problem MOP3. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP3

public **MOP3** ()
 Constructor. Creates default instance of problem MOP3 (10 decision variables)

MOP3

public **MOP3** (*Integer numberOfVariables*)
 Creates a new instance of problem MOP3.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
 Evaluate() method

2.63.4 MOP4

public class **MOP4** extends *AbstractDoubleProblem*
 Problem MOP4. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP4

public **MOP4** ()
 Constructor. Creates default instance of problem MOP4 (10 decision variables)

MOP4

public **MOP4** (*Integer numberOfVariables*)
 Creates a new instance of problem MOP4.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.63.5 MOP5

public class **MOP5** extends *AbstractDoubleProblem*

Problem MOP5. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP5

public **MOP5** ()
Constructor. Creates default instance of problem MOP5 (10 decision variables)

MOP5

public **MOP5** (*Integer* numberOfVariables)
Creates a new instance of problem MOP5.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.63.6 MOP6

public class **MOP6** extends *AbstractDoubleProblem*

Problem MOP6. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP6

public **MOP6** ()
Constructor. Creates default instance of problem MOP6 (10 decision variables)

MOP6

public **MOP6** (*Integer numberOfVariables*)
Creates a new instance of problem MOP6.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.63.7 MOP7

public class **MOP7** extends *AbstractDoubleProblem*
Problem MOP7. Defined in H. L. Liu, F. Gu and Q. Zhang, “Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems,” in IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450-455, June 2014.

Author Mastermay

Constructors

MOP7

public **MOP7** ()
Constructor. Creates default instance of problem MOP7 (10 decision variables)

MOP7

public **MOP7** (*Integer numberOfVariables*)
Creates a new instance of problem MOP7.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.64 org.uma.jmetal.problem.multiobjective.wfg

2.64.1 Shapes

public class **Shapes**

Class implementing shape functions for wfg benchmark Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Methods

concave

public float **concave** (float[] *x*, int *m*)
Calculate a concave shape

convex

public float **convex** (float[] *x*, int *m*)
Calculate a convex shape

disc

public float **disc** (float[] *x*, int *A*, float *alpha*, float *beta*)
Calculate a disc shape

linear

public float **linear** (float[] *x*, int *m*)
Calculate a linear shape

mixed

public float **mixed** (float[] *x*, int *A*, float *alpha*)
Calculate a mixed shape

2.64.2 Transformations

public class **Transformations**

Class implementing the basics transformations for wfg

Methods

bFlat

public float **bFlat** (float *y*, float *A*, float *B*, float *C*)
bFlat transformation

bParam

public float **bParam** (float *y*, float *u*, float *A*, float *B*, float *C*)
bParam transformation

bPoly

public float **bPoly** (float *y*, float *alpha*)
bPoly transformation

Throws

- *org.uma.jmetal.util.JMetalException* –

correctTo01

float **correctTo01** (float *a*)

rNonsep

public float **rNonsep** (float[] *y*, int *A*)
rNonsep transformation

rSum

public float **rSum** (float[] *y*, float[] *w*)
rSum transformation

sDecept

public float **sDecept** (float *y*, float *A*, float *B*, float *C*)
sDecept transformation

sLinear

public float **sLinear** (float *y*, float *A*)
sLinear transformation

sMulti

public float **sMulti** (float *y*, int *A*, int *B*, float *C*)
sMulti transformation

2.64.3 WFG

public abstract class **WFG** extends *AbstractDoubleProblem*

Implements a reference abstract class for all wfg org.uma.test problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Fields

a

protected int[] **a**

d

protected int **d**

k

protected int **k**

l

protected int **l**

m

protected int **m**

random

protected *Random* **random**

s

protected int[] **s**

Constructors**WFG**

public **WFG** (*Integer k*, *Integer l*, *Integer M*)
 Constructor Creates a wfg problem

Parameters

- **k** – position-related parameters
- **l** – distance-related parameters
- **M** – Number of objectives

Methods**calculateX**

public float[] **calculateX** (float[] *t*)
 Gets the x vector

correctTo01

public float **correctTo01** (float *a*)

createSolution

public *DoubleSolution* **createSolution** ()

evaluate

public abstract float[] **evaluate** (float[] *variables*)
 Evaluates a solution

Parameters

- **variables** – The solution to evaluate

Returns a double [] with the evaluation results

normalise

public float[] **normalise** (float[] *z*)
 Normalizes a vector (consulte wfg toolkit reference)

subVector

public float[] **subVector** (float[] *z*, int *head*, int *tail*)

Gets a subvector of a given vector (Head inclusive and tail inclusive)

Parameters

- **z** – the vector

Returns the subvector

2.64.4 WFG1

public class **WFG1** extends *WFG*

This class implements the WFG1 problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Constructors

WFG1

public **WFG1** ()

Constructor Creates a default WFG1 instance with 2 position-related parameters 4 distance-related parameters and 2 objectives

WFG1

public **WFG1** (*Integer k*, *Integer l*, *Integer m*)

Creates a WFG1 problem instance

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods

evaluate

public float[] **evaluate** (float[] *z*)

Evaluate

evaluate

public void **evaluate** (*DoubleSolution solution*)

Evaluates a solution

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

public float[] **t1** (float[] *z*, int *k*)
WFG1 t1 transformation

t2

public float[] **t2** (float[] *z*, int *k*)
WFG1 t2 transformation

t3

public float[] **t3** (float[] *z*)
WFG1 t3 transformation

Throws

- *org.uma.jmetal.util.JMetalException* –

t4

public float[] **t4** (float[] *z*, int *k*, int *M*)
WFG1 t4 transformation

2.64.5 WFG2

public class **WFG2** extends *WFG*

This class implements the WFG2 problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Constructors

WFG2

public **WFG2** ()
Creates a default WFG2 instance with 2 position-related parameters 4 distance-related parameters and 2 objectives

WFG2

public **WFG2** (*Integer k*, *Integer l*, *Integer m*)
Creates a WFG2 problem instance

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods

evaluate

public float[] **evaluate** (float[] *z*)

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to runAlgorithm

t1

public float[] **t1** (float[] *z*, int *k*)
WFG2 t1 transformation

t2

public float[] **t2** (float[] *z*, int *k*)
WFG2 t2 transformation

t3

public float[] **t3** (float[] *z*, int *k*, int *M*)
WFG2 t3 transformation

2.64.6 WFG3

public class **WFG3** extends *WFG*

This class implements the WFG3 problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Constructors

WFG3

public **WFG3** ()

Creates a default WFG3 instances with 2 position-related parameters 4 distance-related parameters and 2 objectives

WFG3

public **WFG3** (*Integer k*, *Integer l*, *Integer m*)

Creates a WFG3 problem instance

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods

evaluate

public float[] **evaluate** (float[] *z*)

evaluate

public void **evaluate** (*DoubleSolution solution*)

Evaluates a solution

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

public float[] **t1** (float[] *z*, int *k*)

WFG3 t1 transformation

t2

public float[] **t2** (float[] *z*, int *k*)

WFG3 t2 transformation

t3

```
public float[] t3 (float[] z, int k, int M)
    WFG3 t3 transformation
```

2.64.7 WFG4

```
public class WFG4 extends WFG
```

This class implements the WFG4 problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Constructors**WFG4**

```
public WFG4 ()
    Creates a default WFG4 with 2 position-related parameter, 4 distance-related parameter and 2 objectives
```

WFG4

```
public WFG4 (Integer k, Integer l, Integer m)
    Creates a WFG4 problem instance
```

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods**evaluate**

```
public float[] evaluate (float[] z)
    Evaluate() method
```

evaluate

```
public void evaluate (DoubleSolution solution)
    Evaluates a solution
```

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

```
public float[] t1 (float[] z, int k)
    WFG4 t1 transformation
```

t2

```
public float[] t2 (float[] z, int k, int M)
    WFG4 t2 transformation
```

2.64.8 WFG5

```
public class WFG5 extends WFG
```

This class implements the WFG5 problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Constructors**WFG5**

```
public WFG5 ()
    Creates a default WFG5 instance with 2 position-related parameters 4 distance-related parameters and 2 objectives
```

WFG5

```
public WFG5 (Integer k, Integer l, Integer m)
    Creates a WFG5 problem instance
```

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods**evaluate**

```
public float[] evaluate (float[] z)
    Evaluate() method
```

evaluate

```
public void evaluate (DoubleSolution solution)
    Evaluates a solution
```

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

```
public float[] t1 (float[] z, int k)  
    WFG5 t1 transformation
```

t2

```
public float[] t2 (float[] z, int k, int M)  
    WFG5 t2 transformation
```

2.64.9 WFG6

```
public class WFG6 extends WFG
```

This class implements the WFG6 problem Reference: Simon Huband, Luigi Barone, Lyndon While, Phil Hingston A Scalable Multi-objective Test Problem Toolkit. Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005. Proceedings, volume 3410 of Lecture Notes in Computer Science

Constructors**WFG6**

```
public WFG6 ()  
    Creates a default WFG6 with 2 position-related parameters, 4 distance-related parameters, and 2 objectives
```

WFG6

```
public WFG6 (Integer k, Integer l, Integer m)  
    Creates a WFG6 problem instance
```

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods**evaluate**

```
public float[] evaluate (float[] z)  
    Evaluate() method
```

evaluate

public void **evaluate** (*DoubleSolution* solution)

Evaluates a solution

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

public float[] **t1** (float[] z, int *k*)

WFG6 t1 transformation

t2

public float[] **t2** (float[] z, int *k*, int *M*)

WFG6 t2 transformation

2.64.10 WFG7

public class **WFG7** extends *WFG*

Constructors**WFG7**

public **WFG7** ()

Creates a default WFG7 problem with 2 position-related parameters, 4 distance-related parameters, and 2 objectives

WFG7

public **WFG7** (*Integer* *k*, *Integer* *l*, *Integer* *m*)

Creates a WFG7 problem instance

Parameters

- **k** – Number of position parameters
- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods

evaluate

```
public float[] evaluate (float[] z)
    Evaluate() method
```

evaluate

```
public void evaluate (DoubleSolution solution)
    Evaluate() method
```

t1

```
public float[] t1 (float[] z, int k)
    WFG7 t1 transformation
```

t2

```
public float[] t2 (float[] z, int k)
    WFG7 t2 transformation
```

t3

```
public float[] t3 (float[] z, int k, int M)
    WFG7 t3 transformation
```

2.64.11 WFG8

```
public class WFG8 extends WFG
    Creates a default WFG8 problem with 2 position-related parameters, 4 distance-related parameters, and 2 objectives
```

Constructors

WFG8

```
public WFG8 ()
    Creates a default WFG8 with 2 position-related parameters, 4 distance-related parameters, and 2 objectives
```

WFG8

```
public WFG8 (Integer k, Integer l, Integer m)
    Creates a WFG8 problem instance
```

Parameters

- **k** – Number of position parameters

- **l** – Number of distance parameters
- **m** – Number of objective functions

Methods

evaluate

public float[] **evaluate** (float[] *z*)
Evaluate() method

evaluate

public void **evaluate** (*DoubleSolution* *solution*)
Evaluates a solution

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

public float[] **t1** (float[] *z*, int *k*)
WFG8 t1 transformation

t2

public float[] **t2** (float[] *z*, int *k*)
WFG8 t2 transformation

t3

public float[] **t3** (float[] *z*, int *k*, int *M*)
WFG8 t3 transformation

2.64.12 WFG9

public class **WFG9** extends *WFG*

Creates a default WFG9 problem with 2 position-related parameters, 4 distance-related parameters, and 2 objectives

Constructors

WFG9

public **WFG9** ()
Creates a default WFG9 with 2 position-related parameters, 4 distance-related parameters, and 2 objectives

WFG9

public **WFG9** (*Integer k*, *Integer l*, *Integer m*)
Creates a WFG9 problem instance

Parameters

- **k** – Number of position variables
- **l** – Number of distance variables
- **m** – Number of objective functions

Methods

evaluate

public float[] **evaluate** (float[] z)
Evaluate() method

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluates a solution

Parameters

- **solution** – The solution to runAlgorithm

Throws

- *org.uma.jmetal.util.JMetalException* –

t1

public float[] **t1** (float[] z, int k)
WFG9 t1 transformation

t2

public float[] **t2** (float[] z, int k)
WFG9 t2 transformation

t3

```
public float[] t3 (float[] z, int k, int M)
    WFG9 t3 transformation
```

2.65 org.uma.jmetal.problem.multiobjective.zdt

2.65.1 ZDT1

```
public class ZDT1 extends AbstractDoubleProblem
    Class representing problem ZDT1
```

Constructors

ZDT1

```
public ZDT1 ()
    Constructor. Creates default instance of problem ZDT1 (30 decision variables)
```

ZDT1

```
public ZDT1 (Integer numberOfVariables)
    Creates a new instance of problem ZDT1.
```

Parameters

- **numberOfVariables** – Number of variables.

Methods

evalH

```
public double evalH (double f, double g)
    Returns the value of the ZDT1 function H.
```

Parameters

- **f** – First argument of the function H.
- **g** – Second argument of the function H.

evaluate

```
public void evaluate (DoubleSolution solution)
    Evaluate() method
```

2.65.2 ZDT2

public class **ZDT2** extends *AbstractDoubleProblem*
Class representing problem ZDT2

Constructors

ZDT2

public **ZDT2** ()
Constructor. Creates default instance of problem ZDT2 (30 decision variables)

ZDT2

public **ZDT2** (*Integer numberOfVariables*)
Constructor. Creates a new ZDT2 problem instance.

Parameters

- **numberOfVariables** – Number of variables

Methods

evalH

public double **evalH** (double *f*, double *g*)
Returns the value of the ZDT2 function H.

Parameters

- **f** – First argument of the function H.
- **g** – Second argument of the function H.

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.65.3 ZDT3

public class **ZDT3** extends *AbstractDoubleProblem*
Class representing problem ZDT3

Constructors

ZDT3

public **ZDT3** ()
Constructor. Creates default instance of problem ZDT3 (30 decision variables)

ZDT3

public **ZDT3** (*Integer numberOfVariables*)
 Constructor. Creates a instance of ZDT3 problem.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evalH

public double **evalH** (double *f*, double *g*)
 Returns the value of the ZDT3 function H.

Parameters

- **f** – First argument of the function H.
- **g** – Second argument of the function H.

evaluate

public void **evaluate** (*DoubleSolution solution*)
 Evaluate() method

2.65.4 ZDT4

public class **ZDT4** extends *AbstractDoubleProblem*
 Class representing problem ZDT4

Constructors

ZDT4

public **ZDT4** ()
 Constructor. Creates a default instance of problem ZDT4 (10 decision variables)

ZDT4

public **ZDT4** (*Integer numberOfVariables*)
 Creates a instance of problem ZDT4.

Parameters

- **numberOfVariables** – Number of variables.

Methods

evalG

public double **evalG** (*DoubleSolution* solution)
Returns the value of the ZDT4 function G.

Parameters

- **solution** – Solution

evalH

public double **evalH** (double *f*, double *g*)
Returns the value of the ZDT4 function H.

Parameters

- **f** – First argument of the function H.
- **g** – Second argument of the function H.

evaluate

public void **evaluate** (*DoubleSolution* solution)
Evaluate() method

2.65.5 ZDT5

public class **ZDT5** extends *AbstractBinaryProblem*
Class representing problem ZDT5

Constructors

ZDT5

public **ZDT5** ()
Creates a default instance of problem ZDT5 (11 decision variables)

ZDT5

public **ZDT5** (*Integer* numberOfVariables)
Creates a instance of problem ZDT5

Parameters

- **numberOfVariables** – Number of variables.

Methods

evalG

public double **evalG** (*BinarySolution* solution)
Returns the value of the ZDT5 function G.

Parameters

- **solution** – The solution.

evalH

public double **evalH** (double *f*, double *g*)
Returns the value of the ZDT5 function H.

Parameters

- **f** – First argument of the function H.
- **g** – Second argument of the function H.

evalV

public double **evalV** (double *value*)
Returns the value of the ZDT5 function V.

Parameters

- **value** – The parameter of V function.

evaluate

public void **evaluate** (*BinarySolution* solution)
Evaluate() method

getBitsPerVariable

protected int **getBitsPerVariable** (int *index*)

2.65.6 ZDT6

public class **ZDT6** extends *AbstractDoubleProblem*
Class representing problem ZDT6

Constructors

ZDT6

public **ZDT6** ()
Constructor. Creates a default instance of problem ZDT6 (10 decision variables)

ZDT6

public **ZDT6** (*Integer numberOfVariables*)
Creates a instance of problem ZDT6

Parameters

- **numberOfVariables** – Number of variables

Methods

evalG

public double **evalG** (*DoubleSolution solution*)
Returns the value of the ZDT6 function G.

Parameters

- **solution** – Solution

evalH

public double **evalH** (double *f*, double *g*)
Returns the value of the ZDT6 function H.

Parameters

- **f** – First argument of the function H.
- **g** – Second argument of the function H.

evaluate

public void **evaluate** (*DoubleSolution solution*)
Evaluate() method

2.66 org.uma.jmetal.problem.singleobjective

2.66.1 CEC2005Problem

public class **CEC2005Problem** extends *AbstractDoubleProblem*
Class representing for solving the CEC2005 competition problems.

Fields

testFunction

TestFunc **testFunction**

Constructors

CEC2005Problem

```
public CEC2005Problem (int problemID, int numberOfVariables)  
    Constructor
```

Methods

evaluate

```
public void evaluate (DoubleSolution solution)  
    Evaluate() method
```

2.66.2 Griewank

```
public class Griewank extends AbstractDoubleProblem  
    Class representing problem Griewank
```

Constructors

Griewank

```
public Griewank (Integer numberOfVariables)  
    Constructor Creates a default instance of the Griewank problem
```

Parameters

- **numberOfVariables** – Number of variables of the problem

Methods

evaluate

```
public void evaluate (DoubleSolution solution)  
    Evaluate() method
```

2.66.3 NIntegerMin

```
public class NIntegerMin extends AbstractIntegerProblem  
    Created by Antonio J. Nebro on 03/07/14. Single objective problem for testing integer encoding. Objective:  
    minimizing the distance to value N
```

Constructors

NIntegerMin

```
public NIntegerMin ()
```

NIntegerMin

public **NIntegerMin** (int *numberOfVariables*, int *n*, int *lowerBound*, int *upperBound*)
Constructor

Methods

evaluate

public void **evaluate** (*IntegerSolution* *solution*)
Evaluate() method

2.66.4 OneMax

public class **OneMax** extends *AbstractBinaryProblem*
Class representing problem OneMax. The problem consist of maximizing the number of ‘1’s in a binary string.

Constructors

OneMax

public **OneMax** ()
Constructor

OneMax

public **OneMax** (*Integer* *numberOfBits*)
Constructor

Methods

createSolution

public *BinarySolution* **createSolution** ()

evaluate

public void **evaluate** (*BinarySolution* *solution*)
Evaluate() method

getBitsPerVariable

protected int **getBitsPerVariable** (int *index*)

2.66.5 Rastrigin

public class **Rastrigin** extends *AbstractDoubleProblem*

Constructors

Rastrigin

public **Rastrigin** (*Integer* numberOfVariables)

Constructor Creates a default instance of the Rastrigin problem

Parameters

- **numberOfVariables** – Number of variables of the problem

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

Evaluate() method

2.66.6 Rosenbrock

public class **Rosenbrock** extends *AbstractDoubleProblem*

Constructors

Rosenbrock

public **Rosenbrock** (*Integer* numberOfVariables)

Constructor Creates a default instance of the Rosenbrock problem

Parameters

- **numberOfVariables** – Number of variables of the problem

Methods

evaluate

public void **evaluate** (*DoubleSolution* solution)

Evaluate() method

2.66.7 Sphere

public class **Sphere** extends *AbstractDoubleProblem*

Class representing a Sphere problem.

Constructors

Sphere

```
public Sphere ()  
    Constructor
```

Sphere

```
public Sphere (Integer numberOfVariables)  
    Constructor
```

Methods

evaluate

```
public void evaluate (DoubleSolution solution)  
    Evaluate() method
```

2.66.8 TSP

```
public class TSP extends AbstractIntegerPermutationProblem  
    Class representing a single-objective TSP (Traveling Salesman Problem) problem. It accepts data files from  
    TSPLIB: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/
```

Constructors

TSP

```
public TSP (String distanceFile)  
    Creates a new TSP problem instance
```

Methods

evaluate

```
public void evaluate (PermutationSolution<Integer> solution)  
    Evaluate() method
```

getPermutationLength

```
public int getPermutationLength ()
```


2.67 org.uma.jmetal.problem.singleobjective.cec2005competitioncode

2.67.1 Benchmark

public class **Benchmark**

Fields

CEC2005SUPPORTDATADIRECTORY

public static final `String` **CEC2005SUPPORTDATADIRECTORY**

DEFAULT_FILE_BIAS

public static final `String` **DEFAULT_FILE_BIAS**

MAX_SUPPORT_DIM

public static final int **MAX_SUPPORT_DIM**

NUM_TEST_FUNC

public static final int **NUM_TEST_FUNC**

PIx2

public static final double **PIx2**

loader

public static final `ClassLoader` **loader**

numberFormatter

public static final `DecimalFormat` **numberFormatter**

percentageFormatter

public static final `DecimalFormat` **percentageFormatter**

random

public static final `Random` **random**

scientificFormatter

```
public static final DecimalFormat scientificFormatter
```

test_func_arg_types

```
static final Class<?>[] test_func_arg_types
```

test_func_class_names

```
public static final String[] test_func_class_names
```

Constructors

Benchmark

```
public Benchmark ()
```

Benchmark

```
public Benchmark (String file_bias)
```

Methods

Ax

```
public static void Ax (double[] result, double[][] A, double[] x)
```

EScafferF6

```
public static double EScafferF6 (double[] x)
```

EScafferF6NonCont

```
public static double EScafferF6NonCont (double[] x)
```

F2

```
public static double F2 (double x, double y)
```

F8

```
public static double F8 (double x)
```

F8F2

```
public static double F8F2 (double[] x)
```

ScafferF6

```
public static double ScafferF6 (double x, double y)
```

ackley

```
public static double ackley (double[] x)
```

elliptic

```
public static double elliptic (double[] x)
```

griewank

```
public static double griewank (double[] x)
```

hybrid_composition

```
public static double hybrid_composition (double[] x, HCJob job)
```

loadColumnVector

```
public static void loadColumnVector (BufferedReader brSrc, int rows, double[] column)
```

loadColumnVectorFromFile

```
public static void loadColumnVectorFromFile (String file, int rows, double[] column)
```

loadMatrix

```
public static void loadMatrix (BufferedReader brSrc, int rows, int columns, double[][] matrix)
```

loadMatrixFromFile

```
public static void loadMatrixFromFile (String file, int rows, int columns, double[][] matrix)
```

loadNMatrixFromFile

```
public static void loadNMatrixFromFile (String file, int N, int rows, int columns, double[][][] matrix)
```

loadRowVector

```
public static void loadRowVector (BufferedReader brSrc, int columns, double[] row)
```

loadRowVectorFromFile

```
public static void loadRowVectorFromFile (String file, int columns, double[] row)
```

loadTestDataFromFile

```
public static void loadTestDataFromFile (String file, int num_test_points, int test_dimension, double[][]  
x, double[] f)
```

main

```
public static void main (String[] args)
```

myRound

```
public static double myRound (double x)
```

myXRound

```
public static double myXRound (double x, double o)
```

myXRound

```
public static double myXRound (double x)
```

rastrigin

```
public static double rastrigin (double[] x)
```

rastriginNonCont

```
public static double rastriginNonCont (double[] x)
```

rosenbrock

```
public static double rosenbrock (double[] x)
```

rotate

```
public static void rotate (double[] results, double[] x, double[][] matrix)
```

runTest

```
public void runTest ()
```

runTest

```
public void runTest (int func_num)
```

schwefel_102

```
public static double schwefel_102 (double[] x)
```

shift

```
public static void shift (double[] results, double[] x, double[] o)
```

sphere

```
public static double sphere (double[] x)
```

sphere_noise

```
public static double sphere_noise (double[] x)
```

testFunctionFactory

```
public TestFunc testFunctionFactory (int func_num, int dimension)
```

weierstrass

```
public static double weierstrass (double[] x)
```

weierstrass

```
public static double weierstrass (double[] x, double a, double b, int Kmax)
```

xA

```
public static void xA (double[] result, double[] x, double[][] A)
```

xy

```
public static double xy (double[] x, double[] y)
```

2.67.2 F01ShiftedSphere

public class **F01ShiftedSphere** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final *String* **DEFAULT_FILE_DATA**

FUNCTION_NAME

public static final *String* **FUNCTION_NAME**

Constructors

F01ShiftedSphere

public **F01ShiftedSphere** (int *dimension*, double *bias*)

F01ShiftedSphere

public **F01ShiftedSphere** (int *dimension*, double *bias*, *String* *file_data*)

Methods

f

public double **f** (double[] *x*)

2.67.3 F02ShiftedSchwefel

public class **F02ShiftedSchwefel** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final *String* **DEFAULT_FILE_DATA**

FUNCTION_NAME

public static final *String* **FUNCTION_NAME**

Constructors

F02ShiftedSchwefel

```
public F02ShiftedSchwefel (int dimension, double bias)
```

F02ShiftedSchwefel

```
public F02ShiftedSchwefel (int dimension, double bias, String file_data)
```

Methods

f

```
public double f (double[] x)
```

2.67.4 F03ShiftedRotatedHighCondElliptic

```
public class F03ShiftedRotatedHighCondElliptic extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Constructors

F03ShiftedRotatedHighCondElliptic

```
public F03ShiftedRotatedHighCondElliptic (int dimension, double bias)
```

F03ShiftedRotatedHighCondElliptic

```
public F03ShiftedRotatedHighCondElliptic (int dimension, double bias, String file_data, String  
                                         file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.5 F04ShiftedSchwefelNoise

```
public class F04ShiftedSchwefelNoise extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Constructors

F04ShiftedSchwefelNoise

```
public F04ShiftedSchwefelNoise (int dimension, double bias)
```

F04ShiftedSchwefelNoise

```
public F04ShiftedSchwefelNoise (int dimension, double bias, String file_data)
```

Methods

f

```
public double f (double[] x)
```

2.67.6 F05SchwefelGlobalOptBound

```
public class F05SchwefelGlobalOptBound extends TestFunc
```


Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Constructors

F05SchwefelGlobalOptBound

```
public F05SchwefelGlobalOptBound (int dimension, double bias)
```

F05SchwefelGlobalOptBound

```
public F05SchwefelGlobalOptBound (int dimension, double bias, String file_data)
```

Methods

f

```
public double f (double[] x)
```

2.67.7 F06ShiftedRosenbrock

```
public class F06ShiftedRosenbrock extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Constructors

F06ShiftedRosenbrock

```
public F06ShiftedRosenbrock (int dimension, double bias)
```

F06ShiftedRosenbrock

```
public F06ShiftedRosenbrock (int dimension, double bias, String file_data)
```

Methods

f

```
public double f (double[] x)
```

2.67.8 F07ShiftedRotatedGriewank

```
public class F07ShiftedRotatedGriewank extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Constructors

F07ShiftedRotatedGriewank

```
public F07ShiftedRotatedGriewank (int dimension, double bias)
```

F07ShiftedRotatedGriewank

```
public F07ShiftedRotatedGriewank (int dimension, double bias, String file_data, String file_m)
```

Methods

f

public double **f** (double[] *x*)

2.67.9 F08ShiftedRotatedAckleyGlobalOptBound

public class **F08ShiftedRotatedAckleyGlobalOptBound** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final String **DEFAULT_FILE_DATA**

DEFAULT_FILE_MX_PREFIX

public static final String **DEFAULT_FILE_MX_PREFIX**

DEFAULT_FILE_MX_SUFFIX

public static final String **DEFAULT_FILE_MX_SUFFIX**

FUNCTION_NAME

public static final String **FUNCTION_NAME**

Constructors

F08ShiftedRotatedAckleyGlobalOptBound

public **F08ShiftedRotatedAckleyGlobalOptBound** (int *dimension*, double *bias*)

F08ShiftedRotatedAckleyGlobalOptBound

public **F08ShiftedRotatedAckleyGlobalOptBound** (int *dimension*, double *bias*, String *file_data*,
String *file_m*)

Methods

f

public double **f** (double[] *x*)

2.67.10 F09ShiftedRastrigin

public class **F09ShiftedRastrigin** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final *String* **DEFAULT_FILE_DATA**

FUNCTION_NAME

public static final *String* **FUNCTION_NAME**

Constructors

F09ShiftedRastrigin

public **F09ShiftedRastrigin** (int *dimension*, double *bias*)

F09ShiftedRastrigin

public **F09ShiftedRastrigin** (int *dimension*, double *bias*, *String* *file_data*)

Methods

f

public double **f** (double[] *x*)

2.67.11 F10ShiftedRotatedRastrigin

public class **F10ShiftedRotatedRastrigin** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final *String* **DEFAULT_FILE_DATA**

DEFAULT_FILE_MX_PREFIX

public static final *String* **DEFAULT_FILE_MX_PREFIX**

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Constructors**F10ShiftedRotatedRastrigin**

```
public F10ShiftedRotatedRastrigin (int dimension, double bias)
```

F10ShiftedRotatedRastrigin

```
public F10ShiftedRotatedRastrigin (int dimension, double bias, String file_data, String file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.12 F11ShiftedRotatedWeierstrass

```
public class F11ShiftedRotatedWeierstrass extends TestFunc
```

Fields**DEFAULT_FILE_DATA**

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

Kmax

public static final int **Kmax**

PIx2

public static final double **PIx2**

a

public static final double **a**

b

public static final double **b**

Constructors

F11ShiftedRotatedWeierstrass

public **F11ShiftedRotatedWeierstrass** (int *dimension*, double *bias*)

F11ShiftedRotatedWeierstrass

public **F11ShiftedRotatedWeierstrass** (int *dimension*, double *bias*, *String* *file_data*, *String* *file_m*)

Methods

f

public double **f** (double[] *x*)

2.67.13 F12Schwefel

public class **F12Schwefel** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final *String* **DEFAULT_FILE_DATA**

FUNCTION_NAME

public static final *String* **FUNCTION_NAME**

Constructors

F12Schwefel

public **F12Schwefel** (int *dimension*, double *bias*)

F12Schwefel

public **F12Schwefel** (int *dimension*, double *bias*, [String](#) *file_data*)

Methods

f

public double **f** (double[] *x*)

2.67.14 F13ShiftedExpandedGriewankRosenbrock

public class **F13ShiftedExpandedGriewankRosenbrock** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final [String](#) **DEFAULT_FILE_DATA**

FUNCTION_NAME

public static final [String](#) **FUNCTION_NAME**

Constructors

F13ShiftedExpandedGriewankRosenbrock

public **F13ShiftedExpandedGriewankRosenbrock** (int *dimension*, double *bias*)

F13ShiftedExpandedGriewankRosenbrock

public **F13ShiftedExpandedGriewankRosenbrock** (int *dimension*, double *bias*, [String](#) *file_data*)

Methods

f

public double **f** (double[] *x*)

2.67.15 F14ShiftedRotatedExpandedScaffer

public class **F14ShiftedRotatedExpandedScaffer** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final *String* **DEFAULT_FILE_DATA**

DEFAULT_FILE_MX_PREFIX

public static final *String* **DEFAULT_FILE_MX_PREFIX**

DEFAULT_FILE_MX_SUFFIX

public static final *String* **DEFAULT_FILE_MX_SUFFIX**

FUNCTION_NAME

public static final *String* **FUNCTION_NAME**

Constructors

F14ShiftedRotatedExpandedScaffer

public **F14ShiftedRotatedExpandedScaffer** (int *dimension*, double *bias*)

F14ShiftedRotatedExpandedScaffer

public **F14ShiftedRotatedExpandedScaffer** (int *dimension*, double *bias*, *String* *file_data*, *String* *file_m*)

Methods

f

public double **f** (double[] *x*)

2.67.16 F15HybridComposition1

public class **F15HybridComposition1** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors

F15HybridComposition1

```
public F15HybridComposition1 (int dimension, double bias)
```

F15HybridComposition1

```
public F15HybridComposition1 (int dimension, double bias, String file_data)
```

Methods

f

```
public double f (double[] x)
```

2.67.17 F16RotatedHybridComposition1

```
public class F16RotatedHybridComposition1 extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors

F16RotatedHybridComposition1

```
public F16RotatedHybridComposition1 (int dimension, double bias)
```

F16RotatedHybridComposition1

```
public F16RotatedHybridComposition1 (int dimension, double bias, String file_data, String file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.18 F17RotatedHybridComposition1Noise

```
public class F17RotatedHybridComposition1Noise extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors**F17RotatedHybridComposition1Noise**

```
public F17RotatedHybridComposition1Noise (int dimension, double bias)
```

F17RotatedHybridComposition1Noise

```
public F17RotatedHybridComposition1Noise (int dimension, double bias, String file_data, String file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.19 F18RotatedHybridComposition2

```
public class F18RotatedHybridComposition2 extends TestFunc
```

Fields**DEFAULT_FILE_DATA**

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors

F18RotatedHybridComposition2

```
public F18RotatedHybridComposition2 (int dimension, double bias)
```

F18RotatedHybridComposition2

```
public F18RotatedHybridComposition2 (int dimension, double bias, String file_data, String file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.20 F19RotatedHybridComposition2NarrowBasinGlobalOpt

```
public class F19RotatedHybridComposition2NarrowBasinGlobalOpt extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors**F19RotatedHybridComposition2NarrowBasinGlobalOpt**

```
public F19RotatedHybridComposition2NarrowBasinGlobalOpt (int dimension, double bias)
```

F19RotatedHybridComposition2NarrowBasinGlobalOpt

```
public F19RotatedHybridComposition2NarrowBasinGlobalOpt (int dimension, double bias,  
String file_data, String file_m)
```

Methods

```
f
```

```
public double f (double[] x)
```

2.67.21 F20RotatedHybridComposition2GlobalOptBound

```
public class F20RotatedHybridComposition2GlobalOptBound extends TestFunc
```

Fields**DEFAULT_FILE_DATA**

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

public static final int **NUM_FUNC**

Constructors

F20RotatedHybridComposition2GlobalOptBound

public **F20RotatedHybridComposition2GlobalOptBound**(int *dimension*, double *bias*)

F20RotatedHybridComposition2GlobalOptBound

public **F20RotatedHybridComposition2GlobalOptBound**(int *dimension*, double *bias*, String *file_data*, String *file_m*)

Methods

f

public double **f**(double[] *x*)

2.67.22 F21RotatedHybridComposition3

public class **F21RotatedHybridComposition3** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final String **DEFAULT_FILE_DATA**

DEFAULT_FILE_MX_PREFIX

public static final String **DEFAULT_FILE_MX_PREFIX**

DEFAULT_FILE_MX_SUFFIX

public static final String **DEFAULT_FILE_MX_SUFFIX**

FUNCTION_NAME

public static final String **FUNCTION_NAME**

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors**F21RotatedHybridComposition3**

```
public F21RotatedHybridComposition3 (int dimension, double bias)
```

F21RotatedHybridComposition3

```
public F21RotatedHybridComposition3 (int dimension, double bias, String file_data, String file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.23 F22RotatedHybridComposition3HighCondNumMatrix

```
public class F22RotatedHybridComposition3HighCondNumMatrix extends TestFunc
```

Fields**DEFAULT_FILE_DATA**

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors

F22RotatedHybridComposition3HighCondNumMatrix

public **F22RotatedHybridComposition3HighCondNumMatrix** (int *dimension*, double *bias*)

F22RotatedHybridComposition3HighCondNumMatrix

public **F22RotatedHybridComposition3HighCondNumMatrix** (int *dimension*, double *bias*, [String](#) *file_data*, [String](#) *file_m*)

Methods

f

public double **f** (double[] *x*)

2.67.24 F23NoncontinuousRotatedHybridComposition3

public class **F23NoncontinuousRotatedHybridComposition3** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final [String](#) **DEFAULT_FILE_DATA**

DEFAULT_FILE_MX_PREFIX

public static final [String](#) **DEFAULT_FILE_MX_PREFIX**

DEFAULT_FILE_MX_SUFFIX

public static final [String](#) **DEFAULT_FILE_MX_SUFFIX**

FUNCTION_NAME

public static final [String](#) **FUNCTION_NAME**

NUM_FUNC

public static final int **NUM_FUNC**

Constructors

F23NoncontinuousRotatedHybridComposition3

```
public F23NoncontinuousRotatedHybridComposition3 (int dimension, double bias)
```

F23NoncontinuousRotatedHybridComposition3

```
public F23NoncontinuousRotatedHybridComposition3 (int dimension, double bias, String
                                                    file_data, String file_m)
```

Methods

f

```
public double f (double[] x)
```

2.67.25 F24RotatedHybridComposition4

```
public class F24RotatedHybridComposition4 extends TestFunc
```

Fields

DEFAULT_FILE_DATA

```
public static final String DEFAULT_FILE_DATA
```

DEFAULT_FILE_MX_PREFIX

```
public static final String DEFAULT_FILE_MX_PREFIX
```

DEFAULT_FILE_MX_SUFFIX

```
public static final String DEFAULT_FILE_MX_SUFFIX
```

FUNCTION_NAME

```
public static final String FUNCTION_NAME
```

NUM_FUNC

```
public static final int NUM_FUNC
```

Constructors

F24RotatedHybridComposition4

public **F24RotatedHybridComposition4** (int *dimension*, double *bias*)

F24RotatedHybridComposition4

public **F24RotatedHybridComposition4** (int *dimension*, double *bias*, [String](#) *file_data*, [String](#) *file_m*)

Methods

f

public double **f** (double[] *x*)

2.67.26 F25RotatedHybridComposition4Bound

public class **F25RotatedHybridComposition4Bound** extends *TestFunc*

Fields

DEFAULT_FILE_DATA

public static final [String](#) **DEFAULT_FILE_DATA**

DEFAULT_FILE_MX_PREFIX

public static final [String](#) **DEFAULT_FILE_MX_PREFIX**

DEFAULT_FILE_MX_SUFFIX

public static final [String](#) **DEFAULT_FILE_MX_SUFFIX**

FUNCTION_NAME

public static final [String](#) **FUNCTION_NAME**

NUM_FUNC

public static final int **NUM_FUNC**

Constructors

F25RotatedHybridComposition4Bound

public **F25RotatedHybridComposition4Bound** (int *dimension*, double *bias*)

F25RotatedHybridComposition4Bound

public **F25RotatedHybridComposition4Bound** (int *dimension*, double *bias*, [String](#) *file_data*, [String](#) *file_m*)

Methods

f

public double **f** (double[] *x*)

2.67.27 HCJob

public abstract class **HCJob**

Fields

C

public double **C**

biases

public double[] **biases**

fmax

public double[] **fmax**

lambda

public double[] **lambda**

linearTransformationMatrix

public double[][][] **linearTransformationMatrix**

numberOfBasicFunctions

public int **numberOfBasicFunctions**

numberOfDimensions

public int **numberOfDimensions**

shiftGlobalOptimum

public double[][] **shiftGlobalOptimum**

sigma

public double[] **sigma**

w

public double[] **w**

z

public double[][] **z**

zM

public double[][] **zM**

Constructors

HCJob

public **HCJob** ()

Methods

basicFunc

public abstract double **basicFunc** (int *func_no*, double[] *x*)

2.67.28 TestFunc

public abstract class **TestFunc**

Fields

mBias

protected double **mBias**

mDimension

protected int **mDimension**

mFuncName

protected [String](#) **mFuncName**

Constructors

TestFunc

public **TestFunc** (int *dimension*, double *bias*)

TestFunc

public **TestFunc** (int *dimension*, double *bias*, [String](#) *funcName*)

Methods

bias

public double **bias** ()

dimension

public int **dimension** ()

f

public abstract double **f** (double[] *x*)

name

public [String](#) **name** ()

2.68 org.uma.jmetal.qualityIndicator

2.68.1 CommandLineIndicatorRunner

public class **CommandLineIndicatorRunner**

Class for executing quality indicators from the command line. An optional argument allows to indicate whether the fronts are to be normalized by the quality indicators. Invoking command: `mvn -pl jmetal-exec exec:java -Dexec.mainClass="org.uma.jmetal.qualityIndicator.CommandLineIndicatorRunner" -Dexec.args="indicator referenceFront front normalize"`

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

2.69 org.uma.jmetal.qualityindicator

2.69.1 QualityIndicator

public interface **QualityIndicator**<Evaluate, Result> extends *DescribedEntity*, *Serializable*

Author Antonio J. Nebro

Parameters

- **<Evaluate>** – Entity to runAlgorithm
- **<Result>** – Result of the evaluation

Methods

evaluate

public Result **evaluate** (Evaluate *evaluate*)

2.70 org.uma.jmetal.qualityindicator.impl

2.70.1 Epsilon

public class **Epsilon**<S extends Solution<?>> extends *GenericIndicator*<S>

This class implements the unary epsilon additive indicator as proposed in E. Zitzler, E. Thiele, L. Laumanns, M., Fonseca, C., and Grunert da Fonseca. V (2003): Performance Assessment of Multiobjective Optimizers: An Analysis and Review. The code is the a Java version of the original metric implementation by Eckart Zitzler. It can be used also as a command line program just by typing `$java org.uma.jmetal.qualityindicator.impl.Epsilon`

Author Antonio J. Nebro , Juan J. Durillo

Constructors

Epsilon

```
public Epsilon ()
    Default constructor
```

Epsilon

```
public Epsilon (String referenceParetoFrontFile)
    Constructor
```

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

Epsilon

```
public Epsilon (Front referenceParetoFront)
    Constructor
```

Parameters

- **referenceParetoFront** –

Methods

evaluate

```
public Double evaluate (List<S> solutionList)
    Evaluate() method
```

Parameters

- **solutionList** –

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

isTheLowerTheIndicatorValueTheBetter

```
public boolean isTheLowerTheIndicatorValueTheBetter ()
```

2.70.2 EpsilonTest

public class **EpsilonTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull

public void **shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull** ()

shouldExecuteRaiseAnExceptionIfTheFrontApproximationListIsNull

public void **shouldExecuteRaiseAnExceptionIfTheFrontApproximationListIsNull** ()

shouldExecuteReturnTheCorrectValueCaseA

public void **shouldExecuteReturnTheCorrectValueCaseA** ()

Given a front with points [1.5,4.0], [2.0,3.0],[3.0,2.0] and a Pareto front with points [1.0,3.0], [1.5,2.0], [2.0, 1.5], the value of the epsilon indicator is 1

shouldExecuteReturnTheCorrectValueCaseB

public void **shouldExecuteReturnTheCorrectValueCaseB** ()

Given a front with points [1.5,4.0], [1.5,2.0],[2.0,1.5] and a Pareto front with points [1.0,3.0], [1.5,2.0], [2.0, 1.5], the value of the epsilon indicator is 0.5

shouldExecuteReturnTheRightValueIfTheFrontsContainOnePointWhichIsNotTheSame

public void **shouldExecuteReturnTheRightValueIfTheFrontsContainOnePointWhichIsNotTheSame** ()

Given a front with point [2,3] and a Pareto front with point [1,2], the value of the epsilon indicator is 1

shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame

public void **shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame** ()

shouldGetNameReturnTheCorrectValue

public void **shouldGetNameReturnTheCorrectValue** ()

The same case as shouldExecuteReturnTheCorrectValueCaseB() but using list of solutions

2.70.3 ErrorRatio

public class **ErrorRatio**<Evaluate extends List<? extends Solution<?>>> extends *SimpleDescribedEntity* implements *QualityIndicator*

The Error Ratio (ER) quality indicator reports the ratio of solutions in a front of points that are not members of the true Pareto front. NOTE: the indicator merely checks if the solutions in the front are not members of the second front. No assumption is made about the second front is a true Pareto front, i.e, the front could contain solutions that dominate some of those of the supposed Pareto front. It is a responsibility of the caller to ensure that this does not happen.

Author Antonio J. Nebro TODO: using an epsilon value

Constructors

ErrorRatio

public **ErrorRatio** (*String* *referenceParetoFrontFile*)

Constructor

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

ErrorRatio

public **ErrorRatio** (*Front* *referenceParetoFront*)

Constructor

Parameters

- **referenceParetoFront** –

Methods

evaluate

public *Double* **evaluate** (Evaluate *solutionList*)

Evaluate() method

Parameters

- **solutionList** –

getName

public *String* **getName** ()

2.70.4 ErrorRatioTest

public class **ErrorRatioTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull

public void **shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull** ()

shouldExecuteRaiseAnExceptionIfTheParetoFrontApproximationListIsNull

public void **shouldExecuteRaiseAnExceptionIfTheParetoFrontApproximationListIsNull** ()

shouldExecuteReturnOneIfTheFrontsContainADifferentPoint

public void **shouldExecuteReturnOneIfTheFrontsContainADifferentPoint** ()

shouldExecuteReturnTheCorrectValueCaseA

public void **shouldExecuteReturnTheCorrectValueCaseA** ()

Given a front with points [1.5,4.0], [1.5,2.0],[2.0,1.5] and a Pareto front with points [1.0,3.0], [1.5,2.0], [2.0, 1.5], the value of the epsilon indicator is 2/3

shouldExecuteReturnTheCorrectValueCaseB

public void **shouldExecuteReturnTheCorrectValueCaseB** ()

Given a list with solutions [1.5,3.0], [4.0,2.0] and another lists with solutions [-1.0,-1.0], [0.0,0.0], the value of the epsilon indicator is 1

shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame

public void **shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame** ()

shouldGetNameReturnTheCorrectValue

public void **shouldGetNameReturnTheCorrectValue** ()

2.70.5 GeneralizedSpread

public class **GeneralizedSpread**<S extends Solution<?>> extends *GenericIndicator*<S>

This class implements the generalized spread metric for two or more dimensions. Reference: A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, and E. Tsang Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion, 2006 IEEE Congress on Evolutionary Computation, 2006, pp. 3234-3241.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

GeneralizedSpread

public **GeneralizedSpread**()

Default constructor

GeneralizedSpread

public **GeneralizedSpread**(String *referenceParetoFrontFile*)

Constructor

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

GeneralizedSpread

public **GeneralizedSpread**(*Front* *referenceParetoFront*)

Constructor

Parameters

- **referenceParetoFront** –

Throws

- **FileNotFoundException** –

Methods

evaluate

public *Double* **evaluate**(List<S> *solutionList*)

Evaluate() method

Parameters

- **solutionList** –

generalizedSpread

public double **generalizedSpread** (*Front* front, *Front* referenceFront)

Calculates the generalized spread metric. Given the pareto front, the true pareto front as double [] and the number of objectives, the method return the value for the metric.

Parameters

- **front** – The front.
- **referenceFront** – The reference pareto front.

Returns the value of the generalized spread metric

getDescription

public *String* **getDescription** ()

getName

public *String* **getName** ()

isTheLowerTheIndicatorValueTheBetter

public boolean **isTheLowerTheIndicatorValueTheBetter** ()

2.70.6 GenerationalDistance

public class **GenerationalDistance**<S extends Solution<?>> extends *GenericIndicator*<S>

This class implements the generational distance indicator. Reference: Van Veldhuizen, D.A., Lamont, G.B.: Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Dept. Elec. Comput. Eng., Air Force Inst. Technol. (1998)

Author Antonio J. Nebro , Juan J. Durillo

Constructors

GenerationalDistance

public **GenerationalDistance** ()

Default constructor

GenerationalDistance

public **GenerationalDistance** (*String* referenceParetoFrontFile, double p)

Constructor

Parameters

- **referenceParetoFrontFile** –
- **p** –

Throws

- **FileNotFoundException** –

GenerationalDistance

public **GenerationalDistance** (*String* *referenceParetoFrontFile*)

Constructor

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

GenerationalDistance

public **GenerationalDistance** (*Front* *referenceParetoFront*)

Constructor

Parameters

- **referenceParetoFront** –

Methods**evaluate**

public *Double* **evaluate** (*List*<*S*> *solutionList*)

Evaluate() method

Parameters

- **solutionList** –

generationalDistance

public double **generationalDistance** (*Front* *front*, *Front* *referenceFront*)

Returns the generational distance value for a given front

Parameters

- **front** – The front
- **referenceFront** – The reference pareto front

getDescription

public *String* **getDescription** ()

getName

```
public String getName ()
```

isTheLowerTheIndicatorValueTheBetter

```
public boolean isTheLowerTheIndicatorValueTheBetter ()
```

2.70.7 GenerationalDistanceTest

```
public class GenerationalDistanceTest
```

Author Antonio J. Nebro

Fields

exception

```
public ExpectedException exception
```

Methods

shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull ()
```

shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull ()
```

shouldGetNameReturnTheCorrectValue

```
public void shouldGetNameReturnTheCorrectValue ()
```

2.70.8 GenericIndicator

```
public abstract class GenericIndicator<S> extends SimpleDescribedEntity implements QualityIndicator<List<S>, Double>
    Abstract class representing quality indicators that need a reference front to be computed
```

Author Antonio J. Nebro

Fields

referenceParetoFront

```
protected Front referenceParetoFront
```

Constructors

GenericIndicator

```
public GenericIndicator ()
    Default constructor
```

GenericIndicator

```
public GenericIndicator (String referenceParetoFrontFile)
```

GenericIndicator

```
public GenericIndicator (Front referenceParetoFront)
```

Methods

isTheLowerTheIndicatorValueTheBetter

```
public abstract boolean isTheLowerTheIndicatorValueTheBetter ()
    This method returns true if lower indicator values are preferred and false otherwise
```

setReferenceParetoFront

```
public void setReferenceParetoFront (String referenceParetoFrontFile)
```

setReferenceParetoFront

```
public void setReferenceParetoFront (Front referenceFront)
```

2.70.9 Hypervolume

```
public abstract class Hypervolume<S> extends GenericIndicator<S>
    This interface represents implementations of the Hypervolume quality indicator
    Author Antonio J. Nebro , Juan J. Durillo
```

Constructors

Hypervolume

```
public Hypervolume ()
```

Hypervolume

```
public Hypervolume (String referenceParetoFrontFile)
```

Hypervolume

public **Hypervolume** (*Front* *referenceParetoFront*)

Methods

computeHypervolumeContribution

public abstract **computeHypervolumeContribution** (*List*<*S*> *solutionList*, *List*<*S*> *referenceFrontList*)

getName

public *String* **getName** ()

getOffset

public abstract double **getOffset** ()

isTheLowerTheIndicatorValueTheBetter

public boolean **isTheLowerTheIndicatorValueTheBetter** ()

setOffset

public abstract void **setOffset** (double *offset*)

2.70.10 HypervolumeTest

public class **HypervolumeTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull

public void **shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull** ()

shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull

```
public void shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull ()
```

2.70.11 InvertedGenerationalDistance

```
public class InvertedGenerationalDistance<S extends Solution<?>> extends GenericIndicator<S>
```

This class implements the inverted generational distance metric. Reference: Van Veldhuizen, D.A., Lamont, G.B.: Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Dept. Elec. Comput. Eng., Air Force Inst. Technol. (1998)

Author Antonio J. Nebro , Juan J. Durillo

Constructors**InvertedGenerationalDistance**

```
public InvertedGenerationalDistance ()
```

Default constructor

InvertedGenerationalDistance

```
public InvertedGenerationalDistance (String referenceParetoFrontFile, double p)
```

Constructor

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

InvertedGenerationalDistance

```
public InvertedGenerationalDistance (String referenceParetoFrontFile)
```

Constructor

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

InvertedGenerationalDistance

```
public InvertedGenerationalDistance (Front referenceParetoFront)
```

Constructor

Parameters

- **referenceParetoFront** –

Throws

- **FileNotFoundException** –

Methods**evaluate**

```
public Double evaluate (List<S> solutionList)
```

Evaluate() method

Parameters

- **solutionList** –

getDescription

```
public String getDescription ()
```

getName

```
public String getName ()
```

invertedGenerationalDistance

```
public double invertedGenerationalDistance (Front front, Front referenceFront)
```

Returns the inverted generational distance value for a given front

Parameters

- **front** – The front
- **referenceFront** – The reference pareto front

isTheLowerTheIndicatorValueTheBetter

```
public boolean isTheLowerTheIndicatorValueTheBetter ()
```

2.70.12 InvertedGenerationalDistancePlus

```
public class InvertedGenerationalDistancePlus<S extends Solution<?>> extends GenericIndicator<S>
```

This class implements the inverted generational distance metric plus (IGD+) Reference: Ishibuchi et al 2015, “A Study on Performance Evaluation Ability of a Modified Inverted Generational Distance Indicator”, GECCO 2015

Author Antonio J. Nebro

Constructors

InvertedGenerationalDistancePlus

public **InvertedGenerationalDistancePlus** ()
 Default constructor

InvertedGenerationalDistancePlus

public **InvertedGenerationalDistancePlus** (*String* *referenceParetoFrontFile*)
 Constructor

Parameters

- **referenceParetoFrontFile** –

InvertedGenerationalDistancePlus

public **InvertedGenerationalDistancePlus** (*Front* *referenceParetoFront*)
 Constructor

Parameters

- **referenceParetoFront** –

Throws

- **FileNotFoundException** –

Methods

evaluate

public *Double* **evaluate** (*List*<*S*> *solutionList*)
 Evaluate() method

Parameters

- **solutionList** –

getDescription

public *String* **getDescription** ()

getName

public *String* **getName** ()

invertedGenerationalDistancePlus

public double **invertedGenerationalDistancePlus** (*Front* front, *Front* referenceFront)

Returns the inverted generational distance plus value for a given front

Parameters

- **front** – The front
- **referenceFront** – The reference pareto front

isTheLowerTheIndicatorValueTheBetter

public boolean **isTheLowerTheIndicatorValueTheBetter** ()

2.70.13 InvertedGenerationalDistancePlusTest

public class **InvertedGenerationalDistancePlusTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldConstructorRaiseAnExceptionIfFileNamelsNull

public void **shouldConstructorRaiseAnExceptionIfFileNameIsNull** ()

shouldConstructorRaiseAnExceptionIfTheParetoFrontIsNull

public void **shouldConstructorRaiseAnExceptionIfTheParetoFrontIsNull** ()

shouldEvaluateRaiseAnExceptionIfTheFrontApproximationIsNull

public void **shouldEvaluateRaiseAnExceptionIfTheFrontApproximationIsNull** ()

shouldEvaluateReturnTheCorrectValueCaseA

public void **shouldEvaluateReturnTheCorrectValueCaseA** ()

shouldEvaluateReturnTheCorrectValueCaseB

public void **shouldEvaluateReturnTheCorrectValueCaseB** ()

shouldEvaluateReturnZeroIfTheFrontAndTheReferenceFrontContainsTheSamePoints

```
public void shouldEvaluateReturnZeroIfTheFrontAndTheReferenceFrontContainsTheSamePoints ()
```

2.70.14 R2

```
public class R2<Evaluate extends List<? extends Solution<?>>> extends SimpleDescribedEntity implements QualityIndicator<Evaluate>
    TODO: Add comments here
```

Constructors

R2

```
public R2 (Front referenceParetoFront)
    Creates a new instance of the R2 indicator for a problem with two objectives and 100 lambda vectors
```

R2

```
public R2 ()
    Creates a new instance of the R2 indicator for a problem with two objectives and 100 lambda vectors
```

R2

```
public R2 (int nVectors)
    Creates a new instance of the R2 indicator for a problem with two objectives and N lambda vectors
```

R2

```
public R2 (String file, Front referenceParetoFront)
    Constructor Creates a new instance of the R2 indicator for nDimensions It loads the weight vectors from the file
    fileName
```

R2

```
public R2 (int nVectors, Front referenceParetoFront)
    Creates a new instance of the R2 indicator for a problem with two objectives and N lambda vectors
```

R2

```
public R2 (String file)
    Constructor Creates a new instance of the R2 indicator for nDimensions It loads the weight vectors from the file
    fileName
```

Methods

evaluate

public `Double` **evaluate** (Evaluate *solutionList*)

getName

public `String` **getName** ()

r2

public double **r2** (*Front front*)

2.70.15 R2Test

public class **R2Test**

Fields

description

`String` **description**

name

`String` **name**

Methods

testR2HasProperNameAndDescriptionWithEmptyConstructor

public void **testR2HasProperNameAndDescriptionWithEmptyConstructor** ()

testR2HasProperNameAndDescriptionWithFileConstructor

public void **testR2HasProperNameAndDescriptionWithFileConstructor** ()

testR2HasProperNameAndDescriptionWithFileFrontConstructor

public void **testR2HasProperNameAndDescriptionWithFileFrontConstructor** ()

testR2HasProperNameAndDescriptionWithFrontConstructor

public void **testR2HasProperNameAndDescriptionWithFrontConstructor** ()

testR2HasProperNameAndDescriptionWithVectorConstructor

```
public void testR2HasProperNameAndDescriptionWithVectorConstructor ()
```

testR2HasProperNameAndDescriptionWithVectorFrontConstructor

```
public void testR2HasProperNameAndDescriptionWithVectorFrontConstructor ()
```

2.70.16 SetCoverage

public class **SetCoverage** extends *SimpleDescribedEntity* implements *QualityIndicator*<Pair<List<? extends *Solution*<?>>, List<? extends *Solution*<?>>>, List<? extends *Solution*<?>>>>
Set coverage metric

Author Antonio J. Nebro

Constructors**SetCoverage**

```
public SetCoverage ()  
    Constructor
```

Methods**evaluate**

```
public Pair<Double, Double> evaluate (Pair<List<? extends Solution<?>>, List<? extends Solution<?>>>  
    pairOfSolutionLists)
```

evaluate

```
public double evaluate (List<? extends Solution<?>> set1, List<? extends Solution<?>> set2)  
    Calculates the set coverage of set1 over set2
```

Parameters

- **set1** –
- **set2** –

Returns The value of the set coverage

getName

```
public String getName ()
```

2.70.17 SetCoverageTest

public class **SetCoverageTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

setup

public void **setup** ()

shouldExecuteRaiseAnExceptionIfTheFirstFrontIsNull

public void **shouldExecuteRaiseAnExceptionIfTheFirstFrontIsNull** ()

shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull

public void **shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull** ()

shouldExecuteReturnOneIfTheSecondFrontIsEmpty

public void **shouldExecuteReturnOneIfTheSecondFrontIsEmpty** ()

shouldExecuteReturnTheCorrectValueCaseA

public void **shouldExecuteReturnTheCorrectValueCaseA** ()

Given a frontA with points [0.0,6.0], [2.0,3.0],[4.0,2.0] and a frontB with points [1.0,7.0], [2.0,3.0], [3.5, 1.0], the value of setCoverage(frontA, frontB) == 1/3 and setCoverage(frontB, frontA) == 1/3

shouldExecuteReturnTheCorrectValueCaseB

public void **shouldExecuteReturnTheCorrectValueCaseB** ()

Given a frontA with points [0.0,6.0], [2.0,3.0],[4.0,2.0] and a frontB with points [1.0,7.0], [2.5,3.0], [5.0, 2.5], the value of setCoverage(frontA, frontB) == 1 and setCoverage(frontB, frontA) == 0

shouldExecuteReturnTheRightValueIfTheFrontsContainOnePointWhichIsNotTheSame

public void **shouldExecuteReturnTheRightValueIfTheFrontsContainOnePointWhichIsNotTheSame** ()

Given a frontA with point [2,3] and a frontB with point [1,2], the value of the setCoverage(frontA, frontB) == 0 and setCoverage(frontB, frontA) == 1

shouldExecuteReturnZeroIfBothFrontsAreEmpty

```
public void shouldExecuteReturnZeroIfBothFrontsAreEmpty ()
```

shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame

```
public void shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame ()
```

shouldGetNameReturnTheCorrectValue

```
public void shouldGetNameReturnTheCorrectValue ()
```

The same case as `shouldExecuteReturnTheCorrectValueCaseB()` but using solution lists

2.70.18 Spread

```
public class Spread<S extends Solution<?>> extends GenericIndicator<S>
```

This class implements the spread quality indicator. It must be only to two bi-objective problem. Reference: Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evol. Computation 6 (2002) 182-197

Author Antonio J. Nebro , Juan J. Durillo

Constructors**Spread**

```
public Spread ()
```

Default constructor

Spread

```
public Spread (String referenceParetoFrontFile)
```

Constructor

Parameters

- **referenceParetoFrontFile** –

Throws

- **FileNotFoundException** –

Spread

```
public Spread (Front referenceParetoFront)
```

Constructor

Parameters

- **referenceParetoFront** –

Throws

- `FileNotFoundException` –

Methods

`evaluate`

```
public Double evaluate (List<S> solutionList)
```

Evaluate() method

Parameters

- `solutionList` –

`getDescription`

```
public String getDescription ()
```

`getName`

```
public String getName ()
```

`isTheLowerTheIndicatorValueTheBetter`

```
public boolean isTheLowerTheIndicatorValueTheBetter ()
```

`spread`

```
public double spread (Front front, Front referenceFront)
```

Calculates the Spread metric.

Parameters

- `front` – The front.
- `referenceFront` – The true pareto front.

2.71 org.uma.jmetal.qualityindicator.impl.hypervolume

2.71.1 PISAHypervolume

```
public class PISAHypervolume<S extends Solution<?>> extends Hypervolume<S>
```

This class implements the hypervolume indicator. The code is the a Java version of the original metric implementation by Eckart Zitzler. Reference: E. Zitzler and L. Thiele Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach, IEEE Transactions on Evolutionary Computation, vol. 3, no. 4, pp. 257-271, 1999.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

PISAHypervolume

public **PISAHypervolume** ()
Default constructor

PISAHypervolume

public **PISAHypervolume** (*String* *referenceParetoFrontFile*)
Constructor

Parameters

- *referenceParetoFrontFile* –

Throws

- *FileNotFoundException* –

PISAHypervolume

public **PISAHypervolume** (*Front* *referenceParetoFront*)
Constructor

Parameters

- *referenceParetoFront* –

Throws

- *FileNotFoundException* –

Methods

calculateHypervolume

public double **calculateHypervolume** (double[][] *front*, int *noPoints*, int *noObjectives*)

computeHypervolumeContribution

public *List*<*S*> **computeHypervolumeContribution** (*List*<*S*> *solutionList*, *List*<*S*> *referenceFrontList*)

evaluate

public *Double* **evaluate** (*List*<*S*> *paretoFrontApproximation*)
Evaluate() method

Parameters

- *paretoFrontApproximation* –

getDescription

```
public String getDescription ()
```

getOffset

```
public double getOffset ()
```

setOffset

```
public void setOffset (double offset)
```

2.71.2 PISAHypervolumeTest

```
public class PISAHypervolumeTest
```

Methods

shouldEvaluateWorkProperlyCase1

```
public void shouldEvaluateWorkProperlyCase1 ()
```

CASE 1: solution set -> front obtained from the ZDT1.rf file. Reference front: [0,1], [1,0]

Throws

- `FileNotFoundException` –

2.71.3 WFGHypervolume

```
public class WFGHypervolume<S> extends Solution<?>> extends Hypervolume<S>
```

Created by ajnebro on 2/2/15.

Constructors

WFGHypervolume

```
public WFGHypervolume ()
```

Default constructor

WFGHypervolume

```
public WFGHypervolume (String referenceParetoFrontFile)
```

Constructor

Parameters

- `referenceParetoFrontFile` –

Throws

- `FileNotFoundException` –

WFGHypervolume

public **WFGHypervolume** (*Front* *referenceParetoFront*)

Constructor

Parameters

- *referenceParetoFront* –

Throws

- `FileNotFoundException` –

Methods

computeHypervolume

public double **computeHypervolume** (*List*<*S*> *solutionList*, *Point* *referencePoint*)

computeHypervolumeContribution

public *List*<*S*> **computeHypervolumeContribution** (*List*<*S*> *solutionList*, *List*<*S*> *referenceFrontList*)

evaluate

public *Double* **evaluate** (*List*<*S*> *solutionList*)

get2DHFV

public double **get2DHFV** (*List*<? extends *Solution*<?>> *solutionSet*)

Computes the HFV of a solution list. **REQUIRES:** The problem is bi-objective **REQUIRES:** The setArchive is ordered in descending order by the second objective

getDescription

public *String* **getDescription** ()

getOffset

public double **getOffset** ()

setOffset

public void **setOffset** (double *offset*)

2.71.4 WFGHypervolumeTest

public class **WFGHypervolumeTest**
Created by ajnebro on 17/12/15.

Methods

setup

public void **setup** ()

shouldEvaluateWorkProperlyCase1

public void **shouldEvaluateWorkProperlyCase1** ()
CASE 1: solution set -> front composed of the points [0.25, 0.75] and [0.75, 0.25]. Reference point: [1.0, 1.0]

shouldEvaluateWorkProperlyCase2

public void **shouldEvaluateWorkProperlyCase2** ()
CASE 2: solution set -> front composed of the points [0.25, 0.75], [0.75, 0.25] and [0.5, 0.5]. Reference point: [1.0, 1.0]

shouldEvaluateWorkProperlyCase3

public void **shouldEvaluateWorkProperlyCase3** ()
CASE 3: solution set -> front composed of the points [0.25, 0.75], [0.75, 0.25] and [0.5, 0.5]. Reference point: [1.5, 1.5]

shouldEvaluateWorkProperlyCase4

public void **shouldEvaluateWorkProperlyCase4** ()
CASE 4: solution set -> front obtained from the ZDT1.rf file. Reference point: [1.0, 1.0]

Throws

- **FileNotFoundException** –

simpleTest

public void **simpleTest** ()

2.72 org.uma.jmetal.qualityindicator.impl.hypervolume.util

2.72.1 WfgHypervolumeFront

public class **WfgHypervolumeFront** extends *ArrayFront*
Created by ajnebro on 3/2/15.

Constructors

WfgHypervolumeFront

```
public WfgHypervolumeFront ()
```

WfgHypervolumeFront

```
public WfgHypervolumeFront (List<? extends Solution<?>> solutionList)
```

WfgHypervolumeFront

```
public WfgHypervolumeFront (int numberOfPoints, int dimensions)
```

Methods

getNumberOfPoints

```
public int getNumberOfPoints ()
```

getPoint

```
public Point getPoint (int index)
```

setNumberOfPoints

```
public void setNumberOfPoints (int numberOfPoints)
```

setPoint

```
public void setPoint (int index, Point point)
```

2.72.2 WfgHypervolumeVersion

```
public class WfgHypervolumeVersion  
    Created by ajnebro on 2/2/15.
```

Fields

OPT

```
static final int OPT
```

fs

WfgHypervolumeFront[] **fs**

maximizing

boolean **maximizing**

Constructors

WfgHypervolumeVersion

public **WfgHypervolumeVersion** (int *dimension*, int *maxNumberOfPoints*)

WfgHypervolumeVersion

public **WfgHypervolumeVersion** (int *dimension*, int *maxNumberOfPoints*, *Point* *referencePoint*)

Methods

dominates2way

int **dominates2way** (*Point* *p*, *Point* *q*)

get2DHV

public double **get2DHV** (*WfgHypervolumeFront* *front*)

getExclusiveHV

public double **getExclusiveHV** (*WfgHypervolumeFront* *front*, int *point*)

getHV

public double **getHV** (*WfgHypervolumeFront* *front*)

getInclusiveHV

public double **getInclusiveHV** (*Point* *point*)

getLessContributorHV

public int **getLessContributorHV** (List<*Solution*<?>> *solutionList*)

main

```
public static void main (String[] args)
```

makeDominatedBit

```
public void makeDominatedBit (WfgHypervolumeFront front, int p)
```

2.73 org.uma.jmetal.runner.multiobjective

2.73.1 ABYSSRunner

```
public class ABYSSRunner extends AbstractAlgorithmRunner
```

This class is the main program used to configure and run AbYSS, a multiobjective scatter search metaheuristics, which is described in: A.J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, A. Beham “AbYSS: Adapting Scatter Search to Multiobjective Optimization.” IEEE Transactions on Evolutionary Computation. Vol. 12, No. 4 (August 2008), pp. 439-457

Author Antonio J. Nebro

Methods**main**

```
public static void main (String[] args)
```

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java org.uma.jmetal.runner.multiobjective.AbYSSRunner problemName [referenceFront]

2.73.2 CDGRunner

```
public class CDGRunner extends AbstractAlgorithmRunner
```

Class for configuring and running the CDG algorithm The paper and Matlab code can be download at <http://xinyecai.github.io/>

Author Feng Zhang

Methods**main**

```
public static void main (String[] args)
```

Parameters

- **args** – Command line arguments.

Throws

- **ClassNotFoundException** –
- **SecurityException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.CDGRunner problemName [referenceFront]

2.73.3 CellDERunner

public class **CellDERunner** extends *AbstractAlgorithmRunner*

Class to configure and run the MOCeII algorithm

Author Antonio J. Nebro

Methods**main**

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- **JMetalException** –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.MOCeIIRunner problemName [referenceFront]

2.73.4 DMOPSOmeasuresRunner

public class **DMOPSOmeasuresRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the DMOPSO algorithm

Author Antonio J. Nebro

Methods**main**

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.DMOPSORunner problemName [referenceFront]

2.73.5 DMOPSORunner

public class **DMOPSORunner** extends *AbstractAlgorithmRunner*
 Class for configuring and running the DMOPSO algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- *SecurityException* – Invoking command: java
 org.uma.jmetal.runner.multiobjective.DMOPSORunner problemName [referenceFront]

2.73.6 ESPEARunner

public class **ESPEARunner** extends *AbstractAlgorithmRunner*
 Class to configure and run the ESPEA algorithm

Author Marlon Braun

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *FileNotFoundException* – Invoking command: java
 org.uma.jmetal.runner.multiobjective.ESPEARunner problemName [referenceFront]

2.73.7 GDE3BigDataRunner

public class **GDE3BigDataRunner**

Class for configuring and running the GDE3 algorithm for solving a problem of the Big Optimization competition at CEC2015

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: `mvn -pl jmetal-exec exec:java -Dexec.mainClass="org.uma.jmetal.runner.multiobjective.GDE3BigDataRunner" -Dexec.args="[problemName]"`

2.73.8 GDE3Runner

public class **GDE3Runner** extends *AbstractAlgorithmRunner*

Class for configuring and running the GDE3 algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: `java org.uma.jmetal.runner.multiobjective.GDE3Runner problemName [referenceFront]`

2.73.9 GNSGAII Measures With Charts Runner

public class **GNSGAII Measures With Charts Runner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-II algorithm (variant with measures) with the G-Dominance Comparator.

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java org.uma.jmetal.runner.multiobjective.NSGAIIIMeasuresRunner problemName [referenceFront]

2.73.10 GWASGFARunner

public class **GWASGFARunner** extends *AbstractAlgorithmRunner*

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java org.uma.jmetal.runner.multiobjective.GWASGFARunner problemName [referenceFront]

2.73.11 IBEARunner

public class **IBEARunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the IBEA algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *java.io.IOException* –
- **SecurityException** –
- **ClassNotFoundException** – Invoking command: java org.uma.jmetal.runner.multiobjective.IBEARunner problemName [referenceFront]

2.73.12 MOCHC45Runner

public class **MOCHC45Runner** extends *AbstractAlgorithmRunner*

This class executes the algorithm described in: A.J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, J.J. Durillo “Optimal antenna placement using a new multi-objective chc algorithm”. GECCO ‘07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England. July 2007.

Methods

main

```
public static void main (String[] args)
```

2.73.13 MOCHCRunner

public class **MOCHCRunner** extends *AbstractAlgorithmRunner*

This class executes the algorithm described in: A.J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, J.J. Durillo “Optimal antenna placement using a new multi-objective chc algorithm”. GECCO ‘07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England. July 2007.

Methods

main

```
public static void main (String[] args)
```

2.73.14 MOCeIIHVRunner

public class **MOCeIIHVRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the MOCeII algorithm

Author Antonio J. Nebro

Methods

main

```
public static void main (String[] args)
```

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.MOCeIIRunner problemName [referenceFront]

2.73.15 MOCellRunner

public class **MOCellRunner** extends *AbstractAlgorithmRunner*
 Class to configure and run the MOCell algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
 org.uma.jmetal.runner.multiobjective.MOCellRunner problemName [referenceFront]

2.73.16 MOEADRunner

public class **MOEADRunner** extends *AbstractAlgorithmRunner*
 Class for configuring and running the MOEA/DD algorithm

Author

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *SecurityException* – Invoking command: java
 org.uma.jmetal.runner.multiobjective.MOEADRunner problemName [referenceFront]

2.73.17 MOEADRunner

public class **MOEADRunner** extends *AbstractAlgorithmRunner*
 Class for configuring and running the MOEA/D algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *SecurityException* – Invoking command: java org.uma.jmetal.runner.multiobjective.MOEADRunner problemName [referenceFront]

2.73.18 MOEADSTMRunner

public class **MOEADSTMRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the MOEA/D algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *SecurityException* – Invoking command: java org.uma.jmetal.runner.multiobjective.MOEADRunner problemName [referenceFront]

2.73.19 MOMBI2Runner

public class **MOMBI2Runner** extends *AbstractAlgorithmRunner*

Class to configure and run the MOMBI2 algorithm

Author Juan J. Durillo Reference: Improved Metaheuristic Based on the R2 Indicator for Many-Objective Optimization. R. Hernández Gómez, C.A. Coello Coello. Proceeding GECCO '15 Proceedings of the 2015 on Genetic and Evolutionary Computation Conference. Pages 679-686 DOI: 10.1145/2739480.2754776

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.MOMBIRunner problemName [referenceFront]

2.73.20 MOMBIRunner

public class **MOMBIRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the MOMBI algorithm

Author Juan J. Durillo

Methods**main**

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.MOMBIRunner problemName [referenceFront]

2.73.21 NSGAI45Runner

public class **NSGAI45Runner** extends *AbstractAlgorithmRunner*

Class to configure and run the implementation of the NSGA-II algorithm included in *NSGAI45*

Author Antonio J. Nebro

Methods**main**

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAI45Runner problemName [referenceFront]

2.73.22 NSGAIIBigDataRunner

public class **NSGAIIBigDataRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the NSGA-II algorithm to solve a problem of the CEC 2015 Big Optimization competition

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *java.io.IOException* –
- *SecurityException* –
- *ClassNotFoundException* – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAIIBigDataRunner problemName [referenceFront]

2.73.23 NSGAIIBinaryRunner

public class **NSGAIIBinaryRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the NSGA-II algorithm (binary encoding)

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- *SecurityException* –
- *ClassNotFoundException* – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAIIBinaryRunner problemName [referenceFront]

2.73.24 NSGAII EbesRunner

public class **NSGAII EbesRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-II algorithm to solve the Ebes problem

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAIIRunner problemName [referenceFront]

2.73.25 NSGAIII Runner

public class **NSGAIII Runner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-III algorithm

Methods

main

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- *java.io.IOException* –
- *SecurityException* –
- **ClassNotFoundException** – Usage: three op-
tions - org.uma.jmetal.runner.multiobjective.NSGAIIIRunner -
org.uma.jmetal.runner.multiobjective.NSGAIIIRunner problemName -
org.uma.jmetal.runner.multiobjective.NSGAIIIRunner problemName paretoFrontFile

2.73.26 NSGAII IntegerRunner

public class **NSGAII IntegerRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the NSGA-II algorithm (integer encoding)

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- *SecurityException* –
- *ClassNotFoundException* – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAIIIntegerRunner problemName [reference-Front]

2.73.27 NSGAII Measures Runner

public class **NSGAII Measures Runner** extends *AbstractAlgorithmRunner*
Class to configure and run the NSGA-II algorithm (variant with measures)

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *SecurityException* – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAII Measures Runner problemName [referenceFront]

2.73.28 NSGAII Measures With Charts Runner

public class **NSGAII Measures With Charts Runner** extends *AbstractAlgorithmRunner*
Class to configure and run the NSGA-II algorithm (variant with measures)

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAII MeasuresRunner problemName [referenceFront]

2.73.29 NSGAII Measures With Hypervolume Runner

public class **NSGAII Measures With Hypervolume Runner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-II algorithm (variant with measures)

Methods

main

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAII MeasuresRunner problemName [referenceFront]

2.73.30 NSGAII Runner

public class **NSGAII Runner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-II algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAII Runner problemName [referenceFront]

2.73.31 NSGAIIStoppingByTimeRunner

public class **NSGAIIStoppingByTimeRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-II algorithm (version NSGAIIStoppingByTime)

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAIIRunner problemName [referenceFront]

2.73.32 NSGAIIITSPRunner

public class **NSGAIIITSPRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the NSGA-II algorithm to solve the bi-objective TSP

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *java.io.IOException* –
- *SecurityException* –
- **ClassNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.NSGAIIITSPRunner problemName [referenceFront]

2.73.33 OMOPSORunner

public class **OMOPSORunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the OMOPSO algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- *SecurityException* – Invoking command: java org.uma.jmetal.runner.multiobjective.OMOPSORunner problemName [referenceFront]

2.73.34 PAESRunner

public class **PAESRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the PAES algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *SecurityException* – Invoking command: java org.uma.jmetal.runner.multiobjective.PAESRunner problemName [referenceFront]

2.73.35 PESA2Runner

public class **PESA2Runner** extends *AbstractAlgorithmRunner*

Class for configuring and running the PESA2 algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.PESA2Runner problemName [referenceFront]`

2.73.36 ParallelGDE3Runner

public class **ParallelGDE3Runner** extends *AbstractAlgorithmRunner*
Class for configuring and running the GDE3 algorithm (parallel version)

Author Antonio J. Nebro

Methods**main**

public static void **main** (`String[] args`)

Parameters

- **args** – Command line arguments.

Throws

- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.ParallelGDE3Runner problemName [reference-Front]`

2.73.37 ParallelNSGAIIRunner

public class **ParallelNSGAIIRunner** extends *AbstractAlgorithmRunner*
Class for configuring and running the NSGA-II algorithm (parallel version)

Author Antonio J. Nebro

Methods**main**

public static void **main** (`String[] args`)

Parameters

- **args** – Command line arguments.

Throws

- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.ParallelNSGAIIRunner problemName [reference-Front]`

2.73.38 ParallelPESA2Runner

public class **ParallelPESA2Runner** extends *AbstractAlgorithmRunner*
 Class for configuring and running the PESA2 algorithm (parallel version)

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java org.uma.jmetal.runner.multiobjective.ParallelPESA2Runner problemName [reference-Front]

2.73.39 ParallelSMPSORunner

public class **ParallelSMPSORunner** extends *AbstractAlgorithmRunner*
 Class for configuring and running the SMPSO algorithm (parallel version)

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments. The first (optional) argument specifies the problem to solve.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- **SecurityException** – Invoking command: java org.uma.jmetal.runner.multiobjective.ParallelSMPSORunner problemName [reference-Front]

2.73.40 ParallelSPEA2Runner

public class **ParallelSPEA2Runner** extends *AbstractAlgorithmRunner*
 /** Class for configuring and running the SPEA2 algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.ParallelSPEA2Runner *problemName* [reference-
Front]

2.73.41 RNSGAIIConstraintRunner

public class **RNSGAIIConstraintRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the R-NSGA-II algorithm

Author Antonio J. Nebro , Cristobal Barba

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.RNSGAIIRunner *problemName* [referenceFront]

2.73.42 RNSGAIIRunner

public class **RNSGAIIRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the R-NSGA-II algorithm

Author Antonio J. Nebro , Cristobal Barba

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.RNSGAIIRunner problemName [referenceFront]

2.73.43 RandomSearchRunner

public class **RandomSearchRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the random search algorithm

Author Antonio J. Nebro

Methods**main**

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments.

Throws

- *SecurityException* – Invoking command: java
org.uma.jmetal.runner.multiobjective.RandomSearchRunner problemName [reference-Front]

2.73.44 SMPSOBigDataRunner

public class **SMPSOBigDataRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SMPSO algorithm to solve a problem of the CEC2015 Big Optimization competition

Author Antonio J. Nebro

Methods**main**

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments. The first (optional) argument specifies the problem to solve.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –

- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.SMPSOBigDataRunner` `problemName` `[referenceFront]`

2.73.45 SMPSOHv2Runner

public class **SMPSOHv2Runner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SMPSO algorithm using an HypervolumeArchive, i.e, the SMPSOhv algorithm described in: A.J Nebro, J.J. Durillo, C.A. Coello Coello. Analysis of Leader Selection Strategies in a Multi-Objective Particle Swarm Optimizer. 2013 IEEE Congress on Evolutionary Computation. June 2013 DOI: 10.1109/CEC.2013.6557955 This is a variant using the WFG Hypervolume implementation

Author Antonio J. Nebro

Methods

main

public static void **main** (`String[] args`)

Parameters

- **args** – Command line arguments. The first (optional) argument specifies the problem to solve.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.SMPSOHvRunner` `problemName` `[referenceFront]`

2.73.46 SMPSOHvRunner

public class **SMPSOHvRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SMPSO algorithm using an HypervolumeArchive, i.e, the SMPSOhv algorithm described in: A.J Nebro, J.J. Durillo, C.A. Coello Coello. Analysis of Leader Selection Strategies in a Multi-Objective Particle Swarm Optimizer. 2013 IEEE Congress on Evolutionary Computation. June 2013 DOI: 10.1109/CEC.2013.6557955

Author Antonio J. Nebro

Methods

main

public static void **main** (`String[] args`)

Parameters

- **args** – Command line arguments. The first (optional) argument specifies the problem to solve.

Throws

- `org.uma.jmetal.util.JMetalException` –
- `java.io.IOException` –
- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.SMPSOHyRunner` `problemName` [referenceFront]

2.73.47 SMPSOMeasuresRunner

public class **SMPSOMeasuresRunner** extends *AbstractAlgorithmRunner*
 Class to configure and run the NSGA-II algorithm (variant with measures)

Methods

main

public static void **main** (`String[] args`)

Parameters

- **args** – Command line arguments.

Throws

- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.NSGAII MeasuresRunner` `problemName` [referenceFront]

2.73.48 SMPSOMeasuresWithChartsRunner

public class **SMPSOMeasuresWithChartsRunner** extends *AbstractAlgorithmRunner*
 Class to configure and run the NSGA-II algorithm (variant with measures)

Methods

main

public static void **main** (`String[] args`)

Parameters

- **args** – Command line arguments.

Throws

- `SecurityException` – Invoking command: `java org.uma.jmetal.runner.multiobjective.NSGAII MeasuresRunner` `problemName` [referenceFront]

2.73.49 SMPSORPChangingTheReferencePointsAndChartsRunner

public class **SMPSORPChangingTheReferencePointsAndChartsRunner**

Methods

main

public static void **main** (`String[] args`)

Program to run the SMPSORP algorithm allowing to change a reference point interactively. SMPSORP is described in “Extending the Speed-constrained Multi-Objective PSO (SMPSO) With Reference Point Based Preference Articulation. Antonio J. Nebro, Juan J. Durillo, José García-Nieto, Cristóbal Barba-González, Javier Del Ser, Carlos A. Coello Coello, Antonio Benítez-Hidalgo, José F. Aldana-Montes. Parallel Problem Solving from Nature – PPSN XV. Lecture Notes In Computer Science, Vol. 11101, pp. 298-310. 2018.” This runner is the one used in the use case included in the paper. In the current implementation, only one reference point can be modified interactively.

Author Antonio J. Nebro

2.73.50 SMPSORPWithMultipleReferencePointsAndChartsRunner

public class **SMPSORPWithMultipleReferencePointsAndChartsRunner**

Methods

main

public static void **main** (`String[] args`)

Program to run the SMPSORP algorithm with three reference points and plotting a graph during the algorithm execution. SMPSORP is described in “Extending the Speed-constrained Multi-Objective PSO (SMPSO) With Reference Point Based Preference Articulation. Antonio J. Nebro, Juan J. Durillo, José García-Nieto, Cristóbal Barba-González, Javier Del Ser, Carlos A. Coello Coello, Antonio Benítez-Hidalgo, José F. Aldana-Montes. Parallel Problem Solving from Nature – PPSN XV. Lecture Notes In Computer Science, Vol. 11101, pp. 298-310. 2018”.

Author Antonio J. Nebro

2.73.51 SMPSORPWithMultipleReferencePointsRunner

public class **SMPSORPWithMultipleReferencePointsRunner**

Methods

main

public static void **main** (`String[] args`)

Program to run the SMPSORP algorithm with two reference points. SMPSORP is described in “Extending the Speed-constrained Multi-Objective PSO (SMPSO) With Reference Point Based Preference Articulation. Antonio J. Nebro, Juan J. Durillo, José García-Nieto, Cristóbal Barba-González, Javier Del Ser, Carlos A. Coello Coello, Antonio Benítez-Hidalgo, José F. Aldana-Montes. Parallel Problem Solving from Nature – PPSN XV. Lecture Notes In Computer Science, Vol. 11101, pp. 298-310. 2018”

Author Antonio J. Nebro

2.73.52 SMPSORPWithOneReferencePointRunner

public class **SMPSORPWithOneReferencePointRunner**

Methods

main

public static void **main** (*String*[] args)

Program to run the SMPSORP algorithm with one reference point. SMPSORP is described in “Extending the Speed-constrained Multi-Objective PSO (SMPSO) With Reference Point Based Preference * Articulation. Antonio J. Nebro, Juan J. Durillo, José García-Nieto, Cristóbal Barba-González, * Javier Del Ser, Carlos A. Coello Coello, Antonio Benítez-Hidalgo, José F. Aldana-Montes. * Parallel Problem Solving from Nature – PPSN XV. Lecture Notes In Computer Science, Vol. 11101, * pp. 298-310. 2018

Author Antonio J. Nebro

2.73.53 SMPSORunner

public class **SMPSORunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SMPSO algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] args)

Parameters

- **args** – Command line arguments. The first (optional) argument specifies the problem to solve.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- *SecurityException* – Invoking command: java org.uma.jmetal.runner.multiobjective.SMPSORunner problemName [referenceFront]

2.73.54 SMSEMOARunner

public class **SMSEMOARunner** extends *AbstractAlgorithmRunner*

Class to configure and run the SMSEMOA algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java org.uma.jmetal.runner.multiobjective.SMSEMOARunner problemName [referenceFront]

2.73.55 SPEA2BinaryRunner

public class **SPEA2BinaryRunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SPEA2 algorithm (binary encoding)

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- **SecurityException** – Invoking command: java org.uma.jmetal.runner.multiobjective.SPEA2BinaryRunner problemName [reference-Front]

2.73.56 SPEA2Runner

public class **SPEA2Runner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SPEA2 algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- `java.io.IOException` –
- `SecurityException` –
- `ClassNotFoundException` – Invoking command: java
org.uma.jmetal.runner.multiobjective.SPEA2BinaryRunner problemName [referenceFront]

2.73.57 SteadyStateNSGAIIRunner

public class **SteadyStateNSGAIIRunner** extends *AbstractAlgorithmRunner*

Class to configure and run the NSGA-II (steady state version) algorithm

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- `FileNotFoundException` – Invoking command: java
org.uma.jmetal.runner.multiobjective.SteadyStateNSGAIIRunner problemName [referenceFront]

2.73.58 WASFGABinaryRunner

public class **WASFGABinaryRunner** extends *AbstractAlgorithmRunner*

Methods

main

public static void **main** (*String[] args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- `FileNotFoundException` – Invoking command: java
org.uma.jmetal.runner.multiobjective.WASFGARunner problemName [referenceFront]

2.73.59 WASFGAMEasuresRunner

public class **WASFGAMEasuresRunner** extends *AbstractAlgorithmRunner*

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- *IOException* –

2.73.60 WASFGAMEasuresRunner3D

public class **WASFGAMEasuresRunner3D** extends *AbstractAlgorithmRunner*

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- *IOException* –

2.73.61 WASFGARunner

public class **WASFGARunner** extends *AbstractAlgorithmRunner*

Methods

main

public static void **main** (*String*[] *args*)

Parameters

- **args** – Command line arguments.

Throws

- *JMetalException* –
- **FileNotFoundException** – Invoking command: java
org.uma.jmetal.runner.multiobjective.WASFGABinaryRunner problemName [referenceFront]

2.74 org.uma.jmetal.runner.singleobjective

2.74.1 CoralReefsOptimizationRunner

public class **CoralReefsOptimizationRunner**

Class to configure and run a coral reefs optimization algorithm. The target problem is OneMax.

Author Inacio Medeiros

Methods

main

public static void **main** (*String*[] args)

Usage: java org.uma.jmetal.runner.singleobjective.CoralReefsOptimizationRunner

2.74.2 CovarianceMatrixAdaptationEvolutionStrategyRunner

public class **CovarianceMatrixAdaptationEvolutionStrategyRunner**

Methods

main

public static void **main** (*String*[] args)

2.74.3 DifferentialEvolutionRunner

public class **DifferentialEvolutionRunner**

Class to configure and run a differential evolution algorithm. The algorithm can be configured to use threads. The number of cores is specified as an optional parameter. The target problem is Sphere.

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] args)

Usage: java org.uma.jmetal.runner.singleobjective.DifferentialEvolutionRunner [cores]

2.74.4 ElitistEvolutionStrategyRunner

public class **ElitistEvolutionStrategyRunner**

Class to configure and run an elitist ($\mu + \lambda$) evolution strategy. The target problem is OneMax.

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Usage: java org.uma.jmetal.runner.singleobjective.ElitistEvolutionStrategyRunner

2.74.5 GenerationalGeneticAlgorithmBinaryEncodingRunner

public class **GenerationalGeneticAlgorithmBinaryEncodingRunner**

Class to configure and run a generational genetic algorithm. The target problem is OneMax.

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Usage: java org.uma.jmetal.runner.singleobjective.GenerationalGeneticAlgorithmBinaryEncodingRunner

2.74.6 GenerationalGeneticAlgorithmDoubleEncodingRunner

public class **GenerationalGeneticAlgorithmDoubleEncodingRunner**

Class to configure and run a generational genetic algorithm. The target problem is OneMax.

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] *args*)

Usage: java org.uma.jmetal.runner.singleobjective.GenerationalGeneticAlgorithmDoubleEncodingRunner

2.74.7 GenerationalGeneticAlgorithmTSPRunner

public class **GenerationalGeneticAlgorithmTSPRunner**

Class to configure and run a generational genetic algorithm. The target problem is TSP.

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)
Usage: java org.uma.jmetal.runner.singleobjective.BinaryGenerationalGeneticAlgorithmRunner

2.74.8 LocalSearchRunner

public class **LocalSearchRunner**
Class to configure and run a single objective local search. The target problem is OneMax.
Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)
Usage: java org.uma.jmetal.runner.singleobjective.LocalSearchRunner

2.74.9 NonElitistEvolutionStrategyRunner

public class **NonElitistEvolutionStrategyRunner**
Class to configure and run a non elitist (mu,lamba) evolution strategy. The target problem is OneMax.
Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)
Usage: java org.uma.jmetal.runner.singleobjective.NonElitistEvolutionStrategyRunner

2.74.10 ParallelGenerationalGeneticAlgorithmRunner

public class **ParallelGenerationalGeneticAlgorithmRunner**
Class to configure and run a parallel (multithreaded) generational genetic algorithm. The number of cores is specified as an optional parameter. A default value is used is the parameter is not provided. The target problem is OneMax
Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)
Usage: java org.uma.jmetal.runner.singleobjective.ParallelGenerationalGeneticAlgorithmRunner [cores]

2.74.11 SMPSORunner

public class **SMPSORunner** extends *AbstractAlgorithmRunner*

Class for configuring and running the SMPSO algorithm to solve a single-objective problem

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

Parameters

- **args** – Command line arguments. The first (optional) argument specifies the problem to solve.

Throws

- *org.uma.jmetal.util.JMetalException* –
- *java.io.IOException* –
- *SecurityException* – Invoking command: java org.uma.jmetal.runner.multiobjective.SMPSORunner problemName [referenceFront]

2.74.12 StandardPSO2007Runner

public class **StandardPSO2007Runner**

Class to configure and run a StandardPSO2007. The algorithm can be configured to use threads. The number of cores is specified as an optional parameter. The target problem is Sphere.

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

Usage: java org.uma.jmetal.runner.singleobjective.StandardPSO2007Runner [cores]

2.74.13 StandardPSO2011Runner

public class **StandardPSO2011Runner**

Class to configure and run a StandardPSO2007. The algorithm can be configured to use threads. The number of cores is specified as an optional parameter. The target problem is Sphere.

Author Antonio J. Nebro

Methods

main

public static void **main** ([String](#)[] args)
Usage: java org.uma.jmetal.runner.singleobjective.StandardPSO2007Runner [cores]

2.74.14 SteadyStateGeneticAlgorithmBinaryEncodingRunner

public class **SteadyStateGeneticAlgorithmBinaryEncodingRunner**
Class to configure and run a steady-state genetic algorithm. The target problem is TSP

Author Antonio J. Nebro

Methods

main

public static void **main** ([String](#)[] args)
Usage: java org.uma.jmetal.runner.singleobjective.SteadyStateGeneticAlgorithmBinaryEncodingRunner

2.74.15 SteadyStateGeneticAlgorithmRunner

public class **SteadyStateGeneticAlgorithmRunner**
Class to configure and run a steady-state genetic algorithm. The target problem is Sphere

Author Antonio J. Nebro

Methods

main

public static void **main** ([String](#)[] args)
Usage: java org.uma.jmetal.runner.singleobjective.SteadyStateGeneticAlgorithmRunner

2.75 org.uma.jmetal.solution

2.75.1 BinarySolution

public interface **BinarySolution** extends [Solution](#)<[BinarySet](#)>
Interface representing a binary (bitset) solutions

Author Antonio J. Nebro

Methods

getNumberOfBits

public int **getNumberOfBits** (int *index*)

getTotalNumberOfBits

public int **getTotalNumberOfBits** ()

2.75.2 DoubleBinarySolution

public interface **DoubleBinarySolution** extends *Solution*<*Object*>

Interface representing a solution having an array of real values and a bitset

Author Antonio J. Nebro

Methods

getLowerBound

public *Double* **getLowerBound** (int *index*)

getNumberOfBits

public int **getNumberOfBits** ()

getNumberOfDoubleVariables

public int **getNumberOfDoubleVariables** ()

getUpperBound

public *Double* **getUpperBound** (int *index*)

2.75.3 DoubleSolution

public interface **DoubleSolution** extends *Solution*<*Double*>

Interface representing a double solutions

Author Antonio J. Nebro

Methods

getLowerBound

public *Double* **getLowerBound** (int *index*)

getUpperBound

public `Double` **getUpperBound** (int *index*)

2.75.4 IntegerDoubleSolution

public interface **IntegerDoubleSolution** extends `Solution<Number>`

Interface representing a solution composed of integers and real values

Author Antonio J. Nebro

Methods

getLowerBound

public `Number` **getLowerBound** (int *index*)

getNumberOfDoubleVariables

public int **getNumberOfDoubleVariables** ()

getNumberOfIntegerVariables

public int **getNumberOfIntegerVariables** ()

getUpperBound

public `Number` **getUpperBound** (int *index*)

2.75.5 IntegerSolution

public interface **IntegerSolution** extends `Solution<Integer>`

Interface representing a integer solutions

Author Antonio J. Nebro

Methods

getLowerBound

public `Integer` **getLowerBound** (int *index*)

getUpperBound

public `Integer` **getUpperBound** (int *index*)

2.75.6 PermutationSolution

public interface **PermutationSolution**<T> extends *Solution*<T>
Interface representing permutation based solutions

Author Antonio J. Nebro

2.75.7 Solution

public interface **Solution**<T> extends *Serializable*
Interface representing a Solution

Author Antonio J. Nebro

Parameters

- <T> – Type (Double, Integer, etc.)

Methods

copy

Solution<T> **copy** ()

getAttribute

Object **getAttribute** (*Object id*)

getNumberOfObjectives

int **getNumberOfObjectives** ()

getNumberOfVariables

int **getNumberOfVariables** ()

getObjective

double **getObjective** (int *index*)

getObjectives

double[] **getObjectives** ()

getVariableValue

T **getVariableValue** (int *index*)

getVariableValueString

`String getVariableValueString (int index)`

setAttribute

`void setAttribute (Object id, Object value)`

setObjective

`void setObjective (int index, double value)`

setVariableValue

`void setVariableValue (int index, T value)`

2.75.8 SolutionBuilder

public interface **SolutionBuilder**<Solution>

A *SolutionBuilder* allows to generate a *Solution* by setting its fundamental information, in other words by providing the values of its *Variables*.

Author Matthieu Vergne

Parameters

- <Solution> –

Methods**build**

public *Solution* **build**()

This method generates a valid *Solution* based on all the Values prepared by calling *prepare(Variable, Object)*. Usually, all the *Variables* should have been prepared before to be able to build a valid *Solution*, but it depends on the definition of the *Solution* (e.g. there could have *Variables* depending on each other, such that preparing one is equivalent to prepare others). Specific implementation could provide a method to know whether or not *build()* can be called, or other facilities to ensure that a *Solution* is properly prepared when *build()* is called.

Returns a new *Solution* instance

getVariables

public *Collection*<*Variable*<*Solution*, ?>> **getVariables**()

Returns the list of *Variables* managed by this *SolutionBuilder*

prepare

```
public <Value> void prepare (Variable<Solution, Value> variable, Value value)
```

This method tells which Value to assign to the next *Solution*, generated by *build()*, for a given *Variable*. Once all the required *Variables* are prepared, *build()* can be called to generate the *Solution*. If this method is called several time on the same *Variable* before to call *build()*, the last prepared Value should be considered.

Parameters

- **variable** – the *Variable* to consider
- **value** – the Value to prepare for this *Variable*

2.75.9 SolutionBuilder.Variable

```
public static interface Variable<Solution, Value> extends DescribedEntity
```

A *Variable* represents the fundamental information of a set of homogeneous *Solutions* (e.g. a population of solutions returned by an *Algorithm*). For instance, an *Algorithm* used to solve a TSP problem would manage a whole population of *Solutions*, each representing a different path, and a *Variable* would represent a type of information which defines these *Solutions*, like the path they represent or something more fine grained like the i^{th} city.

Author Matthieu Vergne

Parameters

- **<Solution>** –
- **<Value>** –

Methods

get

```
public Value get (Solution solution)
```

Parameters

- **solution** – the *Solution* to read

Returns the Value of the *Variable* for this *Solution*

2.75.10 SolutionEvaluator

```
public interface SolutionEvaluator<Solution>
```

A *SolutionEvaluator* allows to evaluate a *Solution* on one or several dimensions, in other words to compute its *Objective* values.

Author Matthieu Vergne

Parameters

- **<Solution>** –

Methods

getObjectives

public `Collection<Objective<Solution, ?>>` **getObjectives** ()

Returns the list of *Objectives* managed by this *SolutionEvaluator*

2.75.11 SolutionEvaluator.Objective

public static interface **Objective**<Solution, Value> extends *DescribedEntity*

An *Objective* represents the evaluation information of a set of homogeneous *Solutions* (e.g. a population of solutions returned by an *Algorithm*). For instance, an *Algorithm* used to solve a TSP problem would manage a whole population of *Solutions*, each representing a different path, and an *Objective* would represent a type of information which evaluates these *Solutions*, like the length of the path, the time needed to travel through this path, or the amount of fuel consumed.

Author Matthieu Vergne

Parameters

- **<Solution>** –
- **<Value>** –

Methods

get

public Value **get** (*Solution* solution)

Parameters

- **solution** – the *Solution* to read

Returns the Value of the *Objective* for this *Solution*

2.76 org.uma.jmetal.solution.impl

2.76.1 AbstractGenericSolution

public abstract class **AbstractGenericSolution**<T, P extends Problem<?>> implements *Solution*<T>

Abstract class representing a generic solution

Author Antonio J. Nebro

Fields

attributes

protected Map<Object, Object> **attributes**

problem

protected P **problem**

randomGenerator

protected final *JMetalRandom* **randomGenerator**

Constructors

AbstractGenericSolution

protected **AbstractGenericSolution** (P *problem*)
 Constructor

Methods

equals

public boolean **equals** (Object *o*)

getAttribute

public Object **getAttribute** (Object *id*)

getNumberOfObjectives

public int **getNumberOfObjectives** ()

getNumberOfVariables

public int **getNumberOfVariables** ()

getObjective

public double **getObjective** (int *index*)

getObjectives

public double[] **getObjectives** ()

getVariableValue

public T **getVariableValue** (int *index*)

hashCode

```
public int hashCode ()
```

initializeObjectiveValues

```
protected void initializeObjectiveValues ()
```

setAttribute

```
public void setAttribute (Object id, Object value)
```

setObjective

```
public void setObjective (int index, double value)
```

setVariableValue

```
public void setVariableValue (int index, T value)
```

toString

```
public String toString ()
```

2.76.2 ArrayDoubleSolution

```
public class ArrayDoubleSolution implements DoubleSolution  
    Implementation of DoubleSolution using arrays.
```

Author Antonio J. Nebro

Fields**attributes**

```
protected Map<Object, Object> attributes
```

problem

```
protected DoubleProblem problem
```

randomGenerator

```
protected final JMetalRandom randomGenerator
```

Constructors

ArrayDoubleSolution

public **ArrayDoubleSolution** (*DoubleProblem* problem)
Constructor

ArrayDoubleSolution

public **ArrayDoubleSolution** (*ArrayDoubleSolution* solution)
Copy constructor

Parameters

- **solution** – to copy

Methods

copy

public *Solution*<*Double*> **copy** ()

equals

public boolean **equals** (*Object* o)

getAttribute

public *Object* **getAttribute** (*Object* id)

getLowerBound

public *Double* **getLowerBound** (int *index*)

getNumberOfObjectives

public int **getNumberOfObjectives** ()

getNumberOfVariables

public int **getNumberOfVariables** ()

getObjective

public double **getObjective** (int *index*)

getObjectives

```
public double[] getObjectives ()
```

getUpperBound

```
public Double getUpperBound (int index)
```

getVariableValue

```
public Double getVariableValue (int index)
```

getVariableValueString

```
public String getVariableValueString (int index)
```

hashCode

```
public int hashCode ()
```

setAttribute

```
public void setAttribute (Object id, Object value)
```

setObjective

```
public void setObjective (int index, double value)
```

setVariableValue

```
public void setVariableValue (int index, Double value)
```

2.76.3 ArrayDoubleSolutionTest

```
public class ArrayDoubleSolutionTest
```

Author Antonio J. Nebro

Methods**setup**

```
public void setup ()
```

shouldConstructorCreateAnObject

```
public void shouldConstructorCreateAnObject ()
```

shouldCopyConstructorCreateAnIdenticalSolution

```
public void shouldCopyConstructorCreateAnIdenticalSolution ()
```

shouldGetLowerBoundReturnTheRightValue

```
public void shouldGetLowerBoundReturnTheRightValue ()
```

shouldGetUpperBoundReturnTheRightValue

```
public void shouldGetUpperBoundReturnTheRightValue ()
```

2.76.4 DefaultBinarySolution

public class **DefaultBinarySolution** extends *AbstractGenericSolution<BinarySet, BinaryProblem>* implements *BinarySolution*
Defines an implementation of a binary solution

Author Antonio J. Nebro

Constructors

DefaultBinarySolution

```
public DefaultBinarySolution (BinaryProblem problem)  
    Constructor
```

DefaultBinarySolution

```
public DefaultBinarySolution (DefaultBinarySolution solution)  
    Copy constructor
```

Methods

copy

```
public DefaultBinarySolution copy ()
```

getNumberOfBits

```
public int getNumberOfBits (int index)
```

getTotalNumberOfBits

```
public int getTotalNumberOfBits ()
```

getVariableValueString

```
public String getVariableValueString (int index)
```

2.76.5 DefaultBinarySolutionTest

```
public class DefaultBinarySolutionTest
```

Fields**problem**

BinaryProblem **problem**

Methods**setUp**

```
public void setUp ()
```

shouldCopyReturnAnIdenticalVariable

```
public void shouldCopyReturnAnIdenticalVariable ()
```

shouldGetNumberOfBitsBeEqualToTheNumberOfOfBitsPerVariable

```
public void shouldGetNumberOfBitsBeEqualToTheNumberOfOfBitsPerVariable ()
```

shouldGetTotalNumberOfBitsBeEqualToTheSumOfBitsPerVariable

```
public void shouldGetTotalNumberOfBitsBeEqualToTheSumOfBitsPerVariable ()
```

shouldGetVariableValueStringReturnARightStringRepresentation

```
public void shouldGetVariableValueStringReturnARightStringRepresentation ()
```

shouldTheHashCodeOfTwoidenticalSolutionsBeTheSame

```
public void shouldTheHashCodeOfTwoIdenticalSolutionsBeTheSame ()
```

shouldTheSumOfGetNumberOfBitsBeEqualToTheSumOfBitsPerVariable

```
public void shouldTheSumOfGetNumberOfBitsBeEqualToTheSumOfBitsPerVariable ()
```

tearDown

```
public void tearDown ()
```

2.76.6 DefaultDoubleBinarySolution

```
public class DefaultDoubleBinarySolution extends AbstractGenericSolution<Object, DoubleBinaryProblem<?>> implements
```

Description: - this solution contains an array of double value + a binary string - `getNumberOfVariables()` returns the number of double values + 1 (the string) - `getNumberOfDoubleVariables()` returns the number of double values - `getNumberOfVariables() = getNumberOfDoubleVariables() + 1` - the bitset is the last variable

Author Antonio J. Nebro

Constructors

DefaultDoubleBinarySolution

```
public DefaultDoubleBinarySolution (DoubleBinaryProblem<?> problem)
```

Constructor

DefaultDoubleBinarySolution

```
public DefaultDoubleBinarySolution (DefaultDoubleBinarySolution solution)
```

Copy constructor

Methods

copy

```
public DefaultDoubleBinarySolution copy ()
```

getLowerBound

```
public Double getLowerBound (int index)
```

getNumberOfBits

```
public int getNumberOfBits ()
```

getNumberOfDoubleVariables

```
public int getNumberOfDoubleVariables ()
```

getUpperBound

```
public Double getUpperBound (int index)
```

getVariableValueString

```
public String getVariableValueString (int index)
```

2.76.7 DefaultDoubleSolution

public class **DefaultDoubleSolution** extends *AbstractGenericSolution*<Double, *DoubleProblem*> implements *DoubleSolution*
Defines an implementation of a double solution

Author Antonio J. Nebro

Constructors**DefaultDoubleSolution**

```
public DefaultDoubleSolution (DoubleProblem problem)  
    Constructor
```

DefaultDoubleSolution

```
public DefaultDoubleSolution (DefaultDoubleSolution solution)  
    Copy constructor
```

Methods**copy**

```
public DefaultDoubleSolution copy ()
```

getLowerBound

```
public Double getLowerBound (int index)
```

getUpperBound

```
public Double getUpperBound (int index)
```

getVariableValueString

```
public String getVariableValueString (int index)
```

2.76.8 DefaultIntegerDoubleSolution

public class **DefaultIntegerDoubleSolution** extends *AbstractGenericSolution*<Number, *IntegerDoubleProblem*<?>> implements *IntegerDoubleSolution*
Defines an implementation of a class for solutions having integers and doubles

Author Antonio J. Nebro

Constructors

DefaultIntegerDoubleSolution

public **DefaultIntegerDoubleSolution** (*IntegerDoubleProblem*<?> *problem*)
Constructor

DefaultIntegerDoubleSolution

public **DefaultIntegerDoubleSolution** (*DefaultIntegerDoubleSolution* *solution*)
Copy constructor

Methods

copy

public *DefaultIntegerDoubleSolution* **copy** ()

getLowerBound

public Number **getLowerBound** (int *index*)

getNumberOfDoubleVariables

public int **getNumberOfDoubleVariables** ()

getNumberOfIntegerVariables

public int **getNumberOfIntegerVariables** ()

getUpperBound

public Number **getUpperBound** (int *index*)

getVariableValueString

public String **getVariableValueString** (int *index*)

2.76.9 DefaultIntegerPermutationSolution

public class **DefaultIntegerPermutationSolution** extends *AbstractGenericSolution*<*Integer*, *PermutationProblem*<?>> implements *IntegerSolution*
 Defines an implementation of solution composed of a permutation of integers

Author Antonio J. Nebro

Constructors

DefaultIntegerPermutationSolution

public **DefaultIntegerPermutationSolution** (*PermutationProblem*<?> *problem*)
 Constructor

DefaultIntegerPermutationSolution

public **DefaultIntegerPermutationSolution** (*DefaultIntegerPermutationSolution* *solution*)
 Copy Constructor

Methods

copy

public *DefaultIntegerPermutationSolution* **copy** ()

getVariableValueString

public *String* **getVariableValueString** (int *index*)

2.76.10 DefaultIntegerPermutationSolutionTest

public class **DefaultIntegerPermutationSolutionTest**

Author Antonio J. Nebro

Methods

shouldConstructorCreateAValidSolution

public void **shouldConstructorCreateAValidSolution** ()

2.76.11 DefaultIntegerSolution

public class **DefaultIntegerSolution** extends *AbstractGenericSolution*<*Integer*, *IntegerProblem*> implements *IntegerSolution*
 Defines an implementation of an integer solution

Author Antonio J. Nebro

Constructors

DefaultIntegerSolution

```
public DefaultIntegerSolution (IntegerProblem problem)
    Constructor
```

DefaultIntegerSolution

```
public DefaultIntegerSolution (DefaultIntegerSolution solution)
    Copy constructor
```

Methods

copy

```
public DefaultIntegerSolution copy ()
```

getLowerBound

```
public Integer getLowerBound (int index)
```

getUpperBound

```
public Integer getUpperBound (int index)
```

getVariableValueString

```
public String getVariableValueString (int index)
```

2.76.12 DoubleSolutionComparisonIT

```
public class DoubleSolutionComparisonIT
    Integration test to compare the performance of ArrayDoubleSolution against
    DefaultDoubleSolution
    Author Antonio J. Nebro
```

Methods

compareDoubleSolutionImplementationsWhenCreatingSolutions

```
public void compareDoubleSolutionImplementationsWhenCreatingSolutions ()
```


compareDoubleSolutionImplementationsWhenEvaluatingSolutions

```
public void compareDoubleSolutionImplementationsWhenEvaluatingSolutions ()
```

2.76.13 ObjectiveFactory

```
public class ObjectiveFactory
```

This factory provides facilities to generate *Objectives* from usual situations.

Author Matthieu Vergne

Methods

createFromGetters

```
public <Solution> Collection<Objective<Solution, ?>> createFromGetters (Class<Solution> solution-
                                                                    Class)
```

This method retrieves all the values accessible through a getter (`getX()` method) in order to build the corresponding set of *Objectives*. Notice that *Objectives* are supposed to represent evaluations of a *Solution*, so if the *Solution* has other kinds of information accessible through getters, they will also be retrieved as *Objectives*. In such a case, you should filter the returned *Objectives*, rely on more advanced methods, or generate the *Objectives* manually.

Parameters

- **solutionClass** – the *Solution* class to analyze

Returns the set of *Objectives* retrieved from this class

See also: `.createFromLonelyGetters(Class)`

createFromGettersWithoutSetters

```
public <Solution> Collection<Objective<Solution, ?>> createFromGettersWithoutSetters (Class<Solution>
                                                                                          solu-
                                                                                          tion-
                                                                                          Class)
```

This method retrieves all the values accessible through a getter (`getX()` method) in order to build the corresponding set of *Objectives*. At the opposite of `createFromGetters(Class)`, an additional filter is used: we build an *Objective* for each getter which does not correspond to a setter (`setX()` method with the same X than the getter). This method is adapted for *Solution* implementations which provide setters only for their fundamental values (e.g. the path of a TSP *Solution*) and use getters only for the computed values (e.g. the length of such a path). Notice that, if all the relevant getters are not present, the corresponding *Objectives* will not be retrieved. On the opposite, any additional getter which does not correspond to a relevant *Objective* will be mistakenly retrieved. So be sure that the relevant elements (and only these ones) have their getter (and no setter). Otherwise, you should use a different method or generate the *Objectives* manually.

Parameters

- **solutionClass** – the *Solution* class to analyze

Returns the set of *Objectives* retrieved from this class

2.76.14 ObjectiveFactoryTest

```
public class ObjectiveFactoryTest
```

Methods

testCreateFromGettersRetrievesAllGetters

```
public void testCreateFromGettersRetrievesAllGetters ()
```

testCreateFromGettersWithoutSettersRetrievesOnlyGettersWithoutSetters

```
public void testCreateFromGettersWithoutSettersRetrievesOnlyGettersWithoutSetters ()
```

testRetrieveObjectiveNames

```
public void testRetrieveObjectiveNames ()
```

2.76.15 VariableFactory

```
public class VariableFactory
```

This factory provides facilities to generate *Variables* from usual situations.

Author Matthieu Vergne

Methods

createFromGetters

```
public <Solution> Collection<Variable<Solution, ?>> createFromGetters (Class<Solution> solution-  
                                                                    Class)
```

This method retrieves all the values accessible through a getter (`getX()` method) in order to build the corresponding set of *Variables*. Notice that *Variables* are supposed to represent the fundamental description of a *Solution*, so if the *Solution* has computation or other additional methods which are named as getters, they will also be retrieved as *Variables*. In such a case, you should filter the returned *Variables*, rely on more advanced methods, or generate the *Variables* manually.

Parameters

- **solutionClass** – the *Solution* class to analyze

Returns the set of *Variables* retrieved from this class

See also: `.createFromGettersAndSetters(Class)`, `.createFromGettersAndConstructors(Class)`

createFromGettersAndConstructors

```
public <Solution> Collection<Variable<Solution, ?>> createFromGettersAndConstructors (Class<Solution>
                                                                    solu-
                                                                    tion-
                                                                    Class)
```

This method retrieves all the values accessible through a getter (`getX()` method) in order to build the corresponding set of *Variables*. At the opposite of `createFromGetters(Class)`, an additional filter is used: we build a *Variable* for each getter which corresponds to a constructor argument (argument of the same type). This method is adapted for static *Solution* implementations, which usually have a constructor which takes all the relevant values and provide getters to retrieve them. Because Java reflection does not always provide the required information (e.g. names of constructor arguments), this method can be applied only on solution classes which meet strict constraints:

- only one getter should return a given type
- for each constructor and between constructors, only one argument should be of a given type (it can appear in several constructors, but it should be always the same argument)

If all the constraints are not met, an exception will be thrown.

Parameters

- **solutionClass** – the *Solution* class to analyze

Throws

- **IllegalArgumentException** – if one of the constraints is not met
- **IsInterfaceException** – if the *Solution* class to analyze is an interface, thus constructors make no sense

Returns the set of *Variables* retrieved from this class

createFromGettersAndSetters

```
public <Solution> Collection<Variable<Solution, ?>> createFromGettersAndSetters (Class<Solution>
                                                                    solution-
                                                                    Class)
```

This method retrieves all the values accessible through a getter (`getX()` method) in order to build the corresponding set of *Variables*. At the opposite of `createFromGetters(Class)`, an additional filter is used: we build a *Variable* for each getter which corresponds to a setter (`setX()` method with the same X than the getter). This method is adapted for dynamic *Solution* implementations, thus allowing to change the value of its *Variables* (e.g. change the path of a TSP *Solution*). Notice that, if all the relevant setters are not present (or they do not strictly respect the naming of the getter), the corresponding *Variables* will not be retrieved. On the opposite, any additional setter/getter couple which does not correspond to a relevant *Variable* will be mistakenly retrieved. So be sure that the relevant elements (and only these ones) have their setter and getter. Otherwise, you should use a different method or generate the *Variables* manually.

Parameters

- **solutionClass** – the *Solution* class to analyze

Returns the set of *Variables* retrieved from this class

2.76.16 VariableFactory.IsInterfaceException

```
public static class IsInterfaceException extends RuntimeException
```

Constructors

IsInterfaceException

```
public IsInterfaceException (Class<?> solutionClass)
```

2.76.17 VariableFactoryTest

```
public class VariableFactoryTest
```

Methods

testCreateFromGettersAndConstructorsRetrievesOnlyGettersWithConstructorArgument

```
public void testCreateFromGettersAndConstructorsRetrievesOnlyGettersWithConstructorArgument ()
```

testCreateFromGettersAndConstructorsThrowExceptionIfInterface

```
public void testCreateFromGettersAndConstructorsThrowExceptionIfInterface ()
```

testCreateFromGettersAndConstructorsThrowExceptionIfOverlappingTypes

```
public void testCreateFromGettersAndConstructorsThrowExceptionIfOverlappingTypes ()
```

testCreateFromGettersAndSettersRetrievesOnlyGettersWithSetters

```
public void testCreateFromGettersAndSettersRetrievesOnlyGettersWithSetters ()
```

testCreateFromGettersRetrievesAllGetters

```
public void testCreateFromGettersRetrievesAllGetters ()
```

testRetrieveVariableNames

```
public void testRetrieveVariableNames ()
```

2.77 org.uma.jmetal.solution.util

2.77.1 RepairDoubleSolution

```
public interface RepairDoubleSolution extends Serializable
```

Author Antonio J. Nebro

Methods

repairSolutionVariableValue

public double **repairSolutionVariableValue** (double *value*, double *lowerBound*, double *upperBound*)

Checks if a given value is between its bounds and repairs it otherwise

Parameters

- **value** – The value to be checked
- **lowerBound** –
- **upperBound** –

Returns The same value if it is between the limits or a repaired value otherwise

2.77.2 RepairDoubleSolutionAtBounds

public class **RepairDoubleSolutionAtBounds** implements *RepairDoubleSolution*

Author Antonio J. Nebro

Methods

repairSolutionVariableValue

public double **repairSolutionVariableValue** (double *value*, double *lowerBound*, double *upperBound*)

Checks if the value is between its bounds; if not, the lower or upper bound is returned

Parameters

- **value** – The value to be checked
- **lowerBound** –
- **upperBound** –

Returns The same value if it is in the limits or a repaired value otherwise

2.77.3 RepairDoubleSolutionAtBoundsTest

public class **RepairDoubleSolutionAtBoundsTest**

Author Antonio J. Nebro

Methods

setup

public void **setup** ()

shouldRRRepairDoubleSolutionAtBoundsAssignTheLowerBoundIfValueIsLessThanIt

```
public void shouldRRRepairDoubleSolutionAtBoundsAssignTheLowerBoundIfValueIsLessThanIt ()
```

shouldRRRepairDoubleSolutionAtBoundsAssignTheUpperBoundIfValueIsGreaterThanIt

```
public void shouldRRRepairDoubleSolutionAtBoundsAssignTheUpperBoundIfValueIsGreaterThanIt ()
```

shouldRRRepairDoubleSolutionAtBoundsRaiseAnExceptionIfTheBoundsAreIncorrect

```
public void shouldRRRepairDoubleSolutionAtBoundsRaiseAnExceptionIfTheBoundsAreIncorrect ()
```

2.77.4 RepairDoubleSolutionAtRandom

public class **RepairDoubleSolutionAtRandom** implements *RepairDoubleSolution*

Author Antonio J. Nebro

Constructors

RepairDoubleSolutionAtRandom

```
public RepairDoubleSolutionAtRandom ()  
    Constructor
```

RepairDoubleSolutionAtRandom

```
public RepairDoubleSolutionAtRandom (BoundedRandomGenerator<Double> randomGenerator)  
    Constructor
```

Methods

repairSolutionVariableValue

```
public double repairSolutionVariableValue (double value, double lowerBound, double upper-  
                                             Bound)
```

Checks if the value is between its bounds; if not, a random value between the limits is returned

Parameters

- **value** – The value to be checked
- **lowerBound** –
- **upperBound** –

Returns The same value if it is between the limits or a repaired value otherwise

2.77.5 RepairDoubleSolutionAtRandomTest

public class **RepairDoubleSolutionAtRandomTest**

Author Antonio J. Nebro

Methods

setup

public void **setup** ()

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

public void **shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided** ()

shouldRRRepairDoubleSolutionAtRandomAssignARandomValueIfValueIsGreaterThanTheUpperBound

public void **shouldRRRepairDoubleSolutionAtRandomAssignARandomValueIfValueIsGreaterThanTheUpperBound**

shouldRRRepairDoubleSolutionAtRandomAssignARandomValueIfValueIsLessThanTheLowerBound

public void **shouldRRRepairDoubleSolutionAtRandomAssignARandomValueIfValueIsLessThanTheLowerBound**

shouldRRRepairDoubleSolutionAtRandomRaiseAnExceptionIfTheBoundsAreIncorrect

public void **shouldRRRepairDoubleSolutionAtRandomRaiseAnExceptionIfTheBoundsAreIncorrect** ()

2.78 org.uma.jmetal.util

2.78.1 AbstractAlgorithmRunner

public abstract class **AbstractAlgorithmRunner**

Abstract class for Runner classes

Author Antonio J. Nebro

Methods

printFinalSolutionSet

public static void **printFinalSolutionSet** (*List*<? extends *Solution*<?>> *population*)

Write the population into two files and prints some data on screen

Parameters

- *population* –

printQualityIndicators

```
public static <S extends Solution<?>> void printQualityIndicators (List<S> population, String
                                                                paretoFrontFile)
```

Print all the available quality indicators

Parameters

- **population** –
- **paretoFrontFile** –

Throws

- **FileNotFoundException** –

2.78.2 AdaptiveGrid

```
public class AdaptiveGrid<S extends Solution<?>>
```

This class defines an adaptive grid over a list of solutions as the one used by algorithm PAES.

Author Antonio J. Nebro, Juan J. Durillo

Constructors

AdaptiveGrid

```
public AdaptiveGrid (int bisections, int objectives)
```

Constructor. Creates an instance of AdaptiveGrid.

Parameters

- **bisections** – Number of bi-divisions of the objective space.
- **objectives** – Number of number of objectives of the problem.

Methods

addSolution

```
public void addSolution (int location)
```

Increases the number of solutions into a specific hypercube.

Parameters

- **location** – Number of hypercube.

calculateOccupied

```
public void calculateOccupied ()
```

Calculates the number of hypercubes having one or more solutions. return the number of hypercubes with more than zero solutions.

getAverageOccupation

public double **getAverageOccupation** ()
 Return the average number of solutions in the occupied hypercubes

getBisections

public int **getBisections** ()
 Returns the number of bi-divisions performed in each objective.
Returns the number of bi-divisions.

getHypercubes

public int[] **getHypercubes** ()

getLocationDensity

public int **getLocationDensity** (int *location*)
 Returns the number of solutions into a specific hypercube.
Parameters

- **location** – Number of the hypercube.

Returns The number of solutions into a specific hypercube.

getMostPopulatedHypercube

public int **getMostPopulatedHypercube** ()
 Returns the value of the most populated hypercube.
Returns The hypercube with the maximum number of solutions.

location

public int **location** (S *solution*)
 Calculates the hypercube of a solution
Parameters

- **solution** – The Solution.

occupiedHypercubes

public int **occupiedHypercubes** ()
 Returns the number of hypercubes with more than zero solutions.
Returns the number of hypercubes with more than zero solutions.

randomOccupiedHypercube

public int **randomOccupiedHypercube** ()
Returns a random hypercube that has more than zero solutions.
Returns The hypercube.

randomOccupiedHypercube

public int **randomOccupiedHypercube** (*BoundedRandomGenerator<Integer> randomGenerator*)
Returns a random hypercube that has more than zero solutions.

Parameters

- **randomGenerator** – the *BoundedRandomGenerator* to use for selecting the hypercube

Returns The hypercube.

removeSolution

public void **removeSolution** (int *location*)
Decreases the number of solutions into a specific hypercube.

Parameters

- **location** – Number of hypercube.

rouletteWheel

public int **rouletteWheel** ()
Returns a random hypercube using a rouletteWheel method.
Returns the number of the selected hypercube.

rouletteWheel

public int **rouletteWheel** (*BoundedRandomGenerator<Double> randomGenerator*)
Returns a random hypercube using a rouletteWheel method.

Parameters

- **randomGenerator** – the *BoundedRandomGenerator* to use for the roulette

Returns the number of the selected hypercube.

toString

public *String* **toString** ()
Returns a String representing the grid.
Returns The String.

updateGrid

public void **updateGrid** (*List<S> solutionList*)

Updates the grid limits and the grid content adding the solutions contained in a specific *solutionList*.

Parameters

- **solutionList** – The *solutionList*.

updateGrid

public void **updateGrid** (*S solution*, *List<S> solutionSet*)

Updates the grid limits and the grid content adding a new *Solution*. If the solution falls out of the grid bounds, the limits and content of the grid must be re-calculated.

Parameters

- **solution** – *Solution* considered to update the grid.
- **solutionSet** – *SolutionSet* used to update the grid.

2.78.3 AdaptiveGridTest

public class **AdaptiveGridTest**

Created by ajnebro on 16/3/17.

Methods

shouldConstructorCreateAValidInstance

public void **shouldConstructorCreateAValidInstance** ()

shouldGetAverageOccupationReturnTheRightValue

public void **shouldGetAverageOccupationReturnTheRightValue** ()

shouldGetAverageOccupationReturnZeroIfThereAreNoOccupiedHypercubes

public void **shouldGetAverageOccupationReturnZeroIfThereAreNoOccupiedHypercubes** ()

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvidedInRandomOccupiedHypercube

public void **shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvidedInRandomOccupiedHypercube**

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvidedInRouletteWheel

public void **shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvidedInRouletteWheel**

shouldOccupiedHypercubesReturnTheNumberOfOccupiedHypercubes

```
public void shouldOccupiedHypercubesReturnTheNumberOfOccupiedHypercubes ()
```

shouldOccupiedHypercubesReturnZeroIfThereAreNotOccupiedHypercubes

```
public void shouldOccupiedHypercubesReturnZeroIfThereAreNotOccupiedHypercubes ()
```

2.78.4 AlgorithmBuilder

```
public interface AlgorithmBuilder<A extends Algorithm<?>>
```

Interface representing algorithm builders

Author Antonio J. Nebro

Methods

build

```
public A build ()
```

2.78.5 AlgorithmRunner

```
public class AlgorithmRunner
```

Class for running algorithms in a concurrent thread

Author Antonio J. Nebro

Methods

getComputingTime

```
public long getComputingTime ()
```

2.78.6 AlgorithmRunner.Executor

```
public static class Executor
```

Executor class

Fields

algorithm

```
Algorithm<?> algorithm
```

computingTime

long `computingTime`

Constructors

Executor

public **Executor** (*Algorithm*<?> *algorithm*)

Methods

execute

public *AlgorithmRunner* **execute** ()

2.78.7 JMetalException

public class **JMetalException** extends *RuntimeException* implements *Serializable*
jMetal exception class

Author Antonio J. Nebro

Constructors

JMetalException

public **JMetalException** (*String message*)

JMetalException

public **JMetalException** (*Exception e*)

JMetalException

public **JMetalException** (*String message*, *Exception e*)

2.78.8 JMetalLogger

public class **JMetalLogger** implements *Serializable*

This class provides some facilities to manage loggers. One might use the static logger of this class or use its own, custom logger. Also, we provide the static method *configureLoggers* (*File*) for configuring the loggers easily. This method is automatically called before any use of the static logger, but if you want it to apply on other loggers it is preferable to call it explicitly at the beginning of your main() method.

Author Antonio J. Nebro , Matthieu Vergne

Fields

logger

public static final `Logger` **logger**

Methods

configureLoggers

public static void **configureLoggers** (File *propertyFile*)

This method provides a single-call method to configure the `Logger` instances. A default configuration is considered, enriched with a custom property file for more convenient logging. The custom file is considered after the default configuration, so it can override it if necessary. The custom file might be provided as an argument of this method, otherwise we look for a file named “jMetal.log.ini”. If no custom file is provided, then only the default configuration is considered.

Parameters

- **propertyFile** – the property file to use for custom configuration, `null` to use only the default configuration

Throws

- `IOException` –

2.78.9 ProblemUtils

public class **ProblemUtils**

Author Antonio J. Nebro

Methods

loadProblem

public static <S> *Problem*<S> **loadProblem** (String *problemName*)

Create an instance of problem passed as argument

Parameters

- **problemName** – A full qualified problem name

Returns An instance of the problem

2.78.10 SolutionListUtils

public class **SolutionListUtils**

Created by Antonio J. Nebro on 04/10/14. Modified by Juanjo 13/03/15

Methods

distanceMatrix

public static <S extends Solution<?>> double[][] **distanceMatrix** (List<S> *solutionSet*)

Returns a matrix with the euclidean distance between each pair of solutions in the population. Distances are measured in the objective space

Parameters

- **solutionSet** –

fillPopulationWithNewSolutions

public static <S> void **fillPopulationWithNewSolutions** (List<S> *solutionList*, *Problem*<S> *problem*, int *maxListSize*)

Fills a population with new solutions until its size is *maxListSize*

Parameters

- **solutionList** – The list of solutions
- **problem** – The problem being solved
- **maxListSize** – The target size of the list
- **<S>** – The type of the solutions to be created

findBestSolution

public static <S> S **findBestSolution** (List<S> *solutionList*, Comparator<S> *comparator*)

findIndexOfBestSolution

public static <S> int **findIndexOfBestSolution** (List<S> *solutionList*, Comparator<S> *comparator*)

Finds the index of the best solution in the list according to a comparator

Parameters

- **solutionList** –
- **comparator** –

Returns The index of the best solution

findIndexOfWorstSolution

public static <S> int **findIndexOfWorstSolution** (List<? extends S> *solutionList*, Comparator<S> *comparator*)

Finds the index of the worst solution in the list according to a comparator

Parameters

- **solutionList** –
- **comparator** –

Returns The index of the best solution

findWorstSolution

```
public <S> S findWorstSolution (Collection<S> solutionList, Comparator<S> comparator)
```

getInvertedFront

```
public static <S extends Solution<?>> List<S> getInvertedFront (List<S> solutionSet)
```

This method receives a normalized list of non-dominated solutions and return the inverted one. This operation is needed for minimization problem

Parameters

- **solutionSet** – The front to invert

Returns The inverted front

getNondominatedSolutions

```
public static <S extends Solution<?>> List<S> getNondominatedSolutions (List<S> solutionList)
```

getNormalizedFront

```
public static List<Solution<?>> getNormalizedFront (List<Solution<?>> solutionList, List<Double>
                                                    maximumValue, List<Double> minimumValue)
```

This method receives a list of non-dominated solutions and maximum and minimum values of the objectives, and returns a the normalized set of solutions.

Parameters

- **solutionList** – A list of non-dominated solutions
- **maximumValue** – The maximum values of the objectives
- **minimumValue** – The minimum values of the objectives

Returns the normalized list of non-dominated solutions

getObjectiveArrayFromSolutionList

```
public static <S extends Solution<?>> double[] getObjectiveArrayFromSolutionList (List<S>
                                                                                    solution-
                                                                                    List,      int
                                                                                    objective)
```

Given a solution list and the identifier of an objective (0, 1, etc), returns an array with the values of that objective in all the solutions of the list

Parameters

- **solutionList** –
- **objective** –
- **<S>** –

isSolutionDominatedBySolutionList

```
public static <S extends Solution<?>> boolean isSolutionDominatedBySolutionList (S solution,
                                                                                   List<? extends S>
                                                                                   solutionSet)
```

removeSolutionsFromList

```
public static <S> void removeSolutionsFromList (List<S> solutionList, int numberOfSolutionsToRe-
                                                                                   move)
```

Removes a number of solutions from a list

Parameters

- **solutionList** – The list of solutions
- **numberOfSolutionsToRemove** –

restart

```
public static <S> void restart (List<S> solutionList, Problem<S> problem, int percentageOfSolutionsToRe-
                                                                                   move)
```

This methods takes a list of solutions, removes a percentage of its solutions, and it is filled with new random generated solutions

Parameters

- **solutionList** –
- **problem** –
- **percentageOfSolutionsToRemove** –

selectNRandomDifferentSolutions

```
public static <S> List<S> selectNRandomDifferentSolutions (int numberOfSolutionsToBeRe-
                                                                                   turned, List<S> solutionList)
```

This method receives a normalized list of non-dominated solutions and return the inverted one. This operation is needed for minimization problem

Parameters

- **solutionList** – The front to invert

Returns The inverted front

selectNRandomDifferentSolutions

```
public static <S> List<S> selectNRandomDifferentSolutions (int numberOfSolutionsToBeRe-
                                                                                   turned, List<S> solutionList, Bound-
                                                                                   edRandomGenerator<Integer>
                                                                                   randomGenerator)
```

This method receives a normalized list of non-dominated solutions and return the inverted one. This operation is needed for minimization problem

Parameters

- **solutionList** – The front to invert
- **randomGenerator** – The random generator to use

Returns The inverted front

solutionListsAreEquals

public static <S> boolean **solutionListsAreEquals** (List<S> *solutionList*, List<S> *newSolutionList*)
Compares two solution lists to determine if both are equals

Parameters

- **solutionList** – A Solution list
- **newSolutionList** – A Solution list

Returns true if both are contains the same solutions, false in other case

writeObjectivesToMatrix

public static <S extends Solution<?>> double[][] **writeObjectivesToMatrix** (List<S> *solutionList*)

2.78.11 SolutionListUtilsTest

public class **SolutionListUtilsTest**

Author Antonio J. Nebro

Fields**exception**

public ExpectedException **exception**

Methods**shouldExecuteReturnTheSolutionInTheListIfTheListContainsASolution**

public void **shouldExecuteReturnTheSolutionInTheListIfTheListContainsASolution** ()

shouldFillPopulationWithNewSolutionsDoNothingIfTheMaxSizeIsLowerThanTheListSize

public void **shouldFillPopulationWithNewSolutionsDoNothingIfTheMaxSizeIsLowerThanTheListSize** ()

shouldFillPopulationWithNewSolutionsIncreaseTheListLengthToTheIndicatedValue

public void **shouldFillPopulationWithNewSolutionsIncreaseTheListLengthToTheIndicatedValue** ()

shouldFindBestSolutionRaiseAnExceptionIfTheComparatorIsNull

```
public void shouldFindBestSolutionRaiseAnExceptionIfTheComparatorIsNull ()
```

shouldFindBestSolutionRaiseAnExceptionIfTheSolutionListIsEmpty

```
public void shouldFindBestSolutionRaiseAnExceptionIfTheSolutionListIsEmpty ()
```

shouldFindBestSolutionRaiseAnExceptionIfTheSolutionListIsNull

```
public void shouldFindBestSolutionRaiseAnExceptionIfTheSolutionListIsNull ()
    ** Unit tests to method findBestSolution ***
```

shouldFindBestSolutionReturnTheLastOneIfThisIsTheBestSolutionInALastInAListWithFiveSolutions

```
public void shouldFindBestSolutionReturnTheLastOneIfThisIsTheBestSolutionInALastInAListWithFiveSolutions
```

shouldFindBestSolutionReturnTheSecondSolutionInTheListIfIsTheBestOutOfTwoSolutions

```
public void shouldFindBestSolutionReturnTheSecondSolutionInTheListIfIsTheBestOutOfTwoSolutions
```

shouldFindBestSolutionReturnTheSolutionInTheListWhenItContainsOneSolution

```
public void shouldFindBestSolutionReturnTheSolutionInTheListWhenItContainsOneSolution ()
```

shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheComparatorIsNull

```
public void shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheComparatorIsNull ()
```

shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheSolutionListIsEmpty

```
public void shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheSolutionListIsEmpty ()
```

shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheSolutionListIsNull

```
public void shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheSolutionListIsNull ()
    ** Unit tests to method findIndexOfBestSolution ***
```

shouldFindIndexOfBestSolutionReturn4IfTheBestSolutionIsTheLastInAListWithFiveSolutions

```
public void shouldFindIndexOfBestSolutionReturn4IfTheBestSolutionIsTheLastInAListWithFiveSolutions
```

shouldFindIndexOfBestSolutionReturnOneIfTheSecondSolutionIsTheBestOutOfTwoSolutionsInTheList

```
public void shouldFindIndexOfBestSolutionReturnOneIfTheSecondSolutionIsTheBestOutOfTwoSolutionsInTheList()
```

shouldFindIndexOfBestSolutionReturnZeroIfTheFirstSolutionIsTheBestOutOfTwoSolutionsInTheList

```
public void shouldFindIndexOfBestSolutionReturnZeroIfTheFirstSolutionIsTheBestOutOfTwoSolutionsInTheList()
```

shouldFindIndexOfBestSolutionReturnZeroIfTheListWhenItContainsOneSolution

```
public void shouldFindIndexOfBestSolutionReturnZeroIfTheListWhenItContainsOneSolution()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvidedInSelectNRandomDifferentSolutions

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvidedInSelectNRandomDifferentSolutions()
```

shouldRestartRemoveTheRequestedPercentageOfSolutions

```
public void shouldRestartRemoveTheRequestedPercentageOfSolutions()
    TODO
```

shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheListSizeIsOneAndTwoSolutionsAreRequested

```
public void shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheListSizeIsOneAndTwoSolutionsAreRequested()
```

shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheSolutionListIsEmpty

```
public void shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheSolutionListIsEmpty()
```

shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheSolutionListIsNull

```
public void shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheSolutionListIsNull()
    ** Unit tests to method selectNRandomDifferentSolutions ***
```

shouldSelectNRandomDifferentSolutionsReturnASingleSolution

```
public void shouldSelectNRandomDifferentSolutionsReturnASingleSolution()
```

shouldSelectNRandomDifferentSolutionsReturnTheCorrectListOfSolutions

```
public void shouldSelectNRandomDifferentSolutionsReturnTheCorrectListOfSolutions()
    If the list contains 4 solutions, the result list must return all of them
```

shouldSelectNRandomDifferentSolutionsReturnTheCorrectNumberOfSolutions

```
public void shouldSelectNRandomDifferentSolutionsReturnTheCorrectNumberOfSolutions ()
```

shouldSelectNRandomDifferentSolutionsReturnTheSolutionSInTheListIfTheListContainsTwoSolutions

```
public void shouldSelectNRandomDifferentSolutionsReturnTheSolutionSInTheListIfTheListContainsTwoSolutions
```

shouldSolutionListsAreEqualsReturnIfTwoidenticalSolutionListsAreCompared

```
public void shouldSolutionListsAreEqualsReturnIfTwoIdenticalSolutionListsAreCompared ()
```

shouldSolutionListsAreEqualsReturnIfTwoSolutionListsWithIdenticalSolutionsAreCompared

```
public void shouldSolutionListsAreEqualsReturnIfTwoSolutionListsWithIdenticalSolutionsAreCompared
```

shouldselectNRandomDifferentSolutionsRaiseAnExceptionIfTheListSizeIsTwoAndFourSolutionsAreRequested

```
public void shouldselectNRandomDifferentSolutionsRaiseAnExceptionIfTheListSizeIsTwoAndFourSolutionsAreRequested
```

2.78.12 SolutionUtils

```
public class SolutionUtils
```

Created by Antonio J. Nebro on 6/12/14.

Methods**averageDistanceToSolutionList**

```
public static <S extends Solution<?>> double averageDistanceToSolutionList (S solution, List<S> solutionList)
```

Returns the average euclidean distance of a solution to the solutions of a list.

distanceBetweenObjectives

```
public static <S extends Solution<?>> double distanceBetweenObjectives (S firstSolution, S secondSolution)
```

Returns the euclidean distance between a pair of solutions in the objective space

distanceBetweenSolutionsInObjectiveSpace

```
public static double distanceBetweenSolutionsInObjectiveSpace (DoubleSolution solutionI, DoubleSolution solutionJ)
```

Returns the distance between two solutions in the search space.

Parameters

- **solutionI** – The first *Solution*.

- **solutionJ** – The second `Solution`.

Returns the distance between solutions.

distanceToSolutionListInSolutionSpace

```
public static double distanceToSolutionListInSolutionSpace (DoubleSolution      solution,  
                                                         List<DoubleSolution>  solutionList)
```

Returns the minimum distance from a `Solution` to a `SolutionSet` according to the encodings.variable values.

Parameters

- **solution** – The `Solution`.
- **solutionList** – The `List`.

Returns The minimum distance between solution and the set.

getBestSolution

```
public static <S extends Solution<?>> S getBestSolution (S solution1, S solution2, Comparator<S> comparator)
```

Return the best solution between those passed as arguments. If they are equal or incomparable one of them is chosen randomly.

Returns The best solution

getBestSolution

```
public static <S extends Solution<?>> S getBestSolution (S solution1, S solution2, Comparator<S> comparator, RandomGenerator<Double> randomGenerator)
```

Return the best solution between those passed as arguments. If they are equal or incomparable one of them is chosen randomly.

Parameters

- **randomGenerator** – *RandomGenerator* for the equality case

Returns The best solution

getBestSolution

```
public static <S extends Solution<?>> S getBestSolution (S solution1, S solution2, Comparator<S> comparator, BinaryOperator<S> equalityPolicy)
```

Return the best solution between those passed as arguments. If they are equal or incomparable one of them is chosen based on the given policy.

Returns The best solution

2.78.13 SolutionUtilsTest

```
public class SolutionUtilsTest
```

Methods

shouldAverageDistanceToSolutionListWorkProperlyCaseA

```
public void shouldAverageDistanceToSolutionListWorkProperlyCaseA()
    Case A. Solution = [1], solutionList = [1]]
```

shouldAverageDistanceToSolutionListWorkProperlyCaseB

```
public void shouldAverageDistanceToSolutionListWorkProperlyCaseB()
    Case B. Solution = [1], solutionList = [[2]]
```

shouldAverageDistanceToSolutionListWorkProperlyCaseC

```
public void shouldAverageDistanceToSolutionListWorkProperlyCaseC()
    Case C. Solution = [1], solutionList = [[1], [2]]
```

shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOneObjectiveCaseA

```
public void shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOneObjectiveCaseA()
    Case A: the two solutions are the same
```

shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOneObjectiveCaseB

```
public void shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOneObjectiveCaseB()
    Case B: the two solutions are not the same
```

shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoObjectivesCaseA

```
public void shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoObjectivesCaseA()
    Case A: the two solutions are the same
```

shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoObjectivesCaseB

```
public void shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoObjectivesCaseB()
    Case B: the two solutions are not the same
```

2.79 org.uma.jmetal.util.archive

2.79.1 Archive

```
public interface Archive<S> extends Serializable
    Interface representing an archive of solutions
```

Author Antonio J. Nebro

Methods

add

boolean **add** (*S solution*)

get

S **get** (int *index*)

getSolutionList

List<*S*> **getSolutionList** ()

size

int **size** ()

2.79.2 BoundedArchive

public interface **BoundedArchive**<*S*> extends *Archive*<*S*>

Interface representing a bounded archive of solutions

Author Antonio J. Nebro

Methods

computeDensityEstimator

void **computeDensityEstimator** ()

getComparator

Comparator<*S*> **getComparator** ()

getMaxSize

int **getMaxSize** ()

sortByDensityEstimator

void **sortByDensityEstimator** ()

2.80 org.uma.jmetal.util.archive.impl

2.80.1 AbstractBoundedArchive

public abstract class **AbstractBoundedArchive**<S> extends `Solution<?>` implements *BoundedArchive*<S>

Author Antonio J. Nebro

Parameters

- <S> –

Fields

archive

protected *NonDominatedSolutionListArchive*<S> **archive**

maxSize

protected int **maxSize**

Constructors

AbstractBoundedArchive

public **AbstractBoundedArchive** (int *maxSize*)

Methods

add

public boolean **add** (S *solution*)

get

public S **get** (int *index*)

getMaxSize

public int **getMaxSize** ()

getSolutionList

public *List*<S> **getSolutionList** ()

join

```
public Archive<S> join (Archive<S> archive)
```

prune

```
public abstract void prune ()
```

size

```
public int size ()
```

2.80.2 AdaptiveGridArchive

```
public class AdaptiveGridArchive<S extends Solution<?>> extends AbstractBoundedArchive<S>
```

This class implements an archive (solution list) based on an adaptive grid used in PAES

Author Antonio J. Nebro , Juan J. Durillo

Constructors

AdaptiveGridArchive

```
public AdaptiveGridArchive (int maxSize, int bisections, int objectives)
```

Constructor.

Parameters

- **maxSize** – The maximum size of the setArchive
- **bisections** – The maximum number of bi-divisions for the adaptive grid.
- **objectives** – The number of objectives.

Methods

add

```
public boolean add (S solution)
```

Adds a *Solution* to the setArchive. If the *Solution* is dominated by any member of the setArchive then it is discarded. If the *Solution* dominates some members of the setArchive, these are removed. If the setArchive is full and the *Solution* has to be inserted, one *Solution* of the most populated hypercube of the adaptive grid is removed.

Parameters

- **solution** – The *Solution*

Returns true if the *Solution* has been inserted, false otherwise.

computeDensityEstimator

```
public void computeDensityEstimator ()
```

getComparator

```
public Comparator<S> getComparator ()
```

getGrid

```
public AdaptiveGrid<S> getGrid ()
```

prune

```
public void prune ()
```

sortByDensityEstimator

```
public void sortByDensityEstimator ()
```

2.80.3 AdaptiveGridArchiveTest

```
public class AdaptiveGridArchiveTest  
    Created by ajnebro on 16/11/16.
```

Methods**shouldConstructorCreateAnArchiveWithTheRightCapacity**

```
public void shouldConstructorCreateAnArchiveWithTheRightCapacity ()
```

shouldConstructorCreateAnEmptyArchive

```
public void shouldConstructorCreateAnEmptyArchive ()
```

shouldPruneDoNothingIfTheArchivesEmpty

```
public void shouldPruneDoNothingIfTheArchiveIsEmpty ()
```

2.80.4 CrowdingDistanceArchive

```
public class CrowdingDistanceArchive<S extends Solution<?>> extends AbstractBoundedArchive<S>  
    Created by Antonio J. Nebro on 24/09/14. Modified by Juanjo on 07/04/2015
```

Constructors

CrowdingDistanceArchive

public **CrowdingDistanceArchive** (int *maxSize*)

Methods

computeDensityEstimator

public void **computeDensityEstimator** ()

getComparator

public [Comparator](#)<S> **getComparator** ()

prune

public void **prune** ()

sortByDensityEstimator

public void **sortByDensityEstimator** ()

2.80.5 HypervolumeArchive

public class **HypervolumeArchive**<S extends [Solution](#)<?>> extends [AbstractBoundedArchive](#)<S>
Created by Antonio J. Nebro on 24/09/14.

Fields

hypervolume

[Hypervolume](#)<S> **hypervolume**

Constructors

HypervolumeArchive

public **HypervolumeArchive** (int *maxSize*, [Hypervolume](#)<S> *hypervolume*)

Methods

computeDensityEstimator

public void **computeDensityEstimator** ()

getComparator

```
public Comparator<S> getComparator ()
```

prune

```
public void prune ()
```

sortByDensityEstimator

```
public void sortByDensityEstimator ()
```

2.80.6 NonDominatedSolutionListArchive

```
public class NonDominatedSolutionListArchive<S extends Solution<?>> implements Archive<S>
```

This class implements an archive containing non-dominated solutions

Author Antonio J. Nebro , Juan J. Durillo

Constructors

NonDominatedSolutionListArchive

```
public NonDominatedSolutionListArchive ()  
    Constructor
```

NonDominatedSolutionListArchive

```
public NonDominatedSolutionListArchive (DominanceComparator<S> comparator)  
    Constructor
```

Methods

add

```
public boolean add (S solution)  
    Inserts a solution in the list
```

Parameters

- **solution** – The solution to be inserted.

Returns true if the operation success, and false if the solution is dominated or if an identical individual exists. The decision variables can be null if the solution is read from a file; in that case, the domination tests are omitted

get

```
public S get (int index)
```

getSolutionList

```
public List<S> getSolutionList ()
```

join

```
public Archive<S> join (Archive<S> archive)
```

main

```
public static void main (String[] args)
```

size

```
public int size ()
```

2.80.7 NonDominatedSolutionListArchiveTest

```
public class NonDominatedSolutionListArchiveTest
```

Author Antonio J. Nebro .

Methods

shouldAddADominantSolutionInAnArchiveOfSize1DiscardTheExistingSolution

```
public void shouldAddADominantSolutionInAnArchiveOfSize1DiscardTheExistingSolution ()
```

shouldAddADominantSolutionInAnArchiveOfSize3DiscardTheRestOfSolutions

```
public void shouldAddADominantSolutionInAnArchiveOfSize3DiscardTheRestOfSolutions ()
```

shouldAddADominatedSolutionInAnArchiveOfSize1DiscardTheNewSolution

```
public void shouldAddADominatedSolutionInAnArchiveOfSize1DiscardTheNewSolution ()
```

shouldAddANonDominantSolutionInAnArchiveOfSize1IncorporateTheNewSolution

```
public void shouldAddANonDominantSolutionInAnArchiveOfSize1IncorporateTheNewSolution ()
```

shouldAddASolutionEqualsToOneAlreadyInTheArchiveDoNothing

```
public void shouldAddASolutionEqualsToOneAlreadyInTheArchiveDoNothing ()
```

shouldAddOnAnEmptyListHaveSizeOne

```
public void shouldAddOnAnEmptyListHaveSizeOne ()
```

shouldAddOnAnEmptyListInsertTheElement

```
public void shouldAddOnAnEmptyListInsertTheElement ()
```

shouldConstructorAssignThePassedComparator

```
public void shouldConstructorAssignThePassedComparator ()
```

shouldConstructorCreateAnEmptyArchive

```
public void shouldConstructorCreateAnEmptyArchive ()
```

shouldJoinAnEAnEmptyArchiveProduceAnArchiveWithTheSameSolutions

```
public void shouldJoinAnEAnEmptyArchiveProduceAnArchiveWithTheSameSolutions ()
```

shouldJoinTwoEmptyArchivesReturnAnEmptyArchive

```
public void shouldJoinTwoEmptyArchivesReturnAnEmptyArchive ()
```

shouldJoinWithAnEmptyArchivesRemainTheArchiveWithTheSameNumberOfSolutions

```
public void shouldJoinWithAnEmptyArchivesRemainTheArchiveWithTheSameNumberOfSolutions ()
```

2.81 org.uma.jmetal.util.archivewithreferencepoint

2.81.1 ArchiveWithReferencePoint

public abstract class **ArchiveWithReferencePoint**<S extends Solution<?>> extends *AbstractBoundedArchive*<S>

This class defines a bounded archive that has associated a reference point as described in the paper “Extending the Speed-constrained Multi-Objective PSO (SMPSO) With Reference Point Based Preference Articulation Accepted in PPSN 2018.

Parameters

- <S> –

Fields**comparator**

```
protected Comparator<S> comparator
```

referencePoint

protected `List<Double>` **referencePoint**

referencePointSolution

protected `S` **referencePointSolution**

Constructors

ArchiveWithReferencePoint

public **ArchiveWithReferencePoint** (int *maxSize*, `List<Double>` *referencePoint*, `Comparator<S>` *comparator*)

Methods

add

public synchronized boolean **add** (`S` *solution*)

changeReferencePoint

public synchronized void **changeReferencePoint** (`List<Double>` *newReferencePoint*)

prune

public synchronized void **prune** ()

2.82 org.uma.jmetal.util.archivewithreferencepoint.impl

2.82.1 CrowdingDistanceArchiveWithReferencePoint

public class **CrowdingDistanceArchiveWithReferencePoint**<`S` extends `Solution`<?>> extends *ArchiveWithReferencePoint*
Class representing a *ArchiveWithReferencePoint* archive using a crowding distance based density estimator

Author Antonio J. Nebro

Constructors

CrowdingDistanceArchiveWithReferencePoint

public **CrowdingDistanceArchiveWithReferencePoint** (int *maxSize*, `List<Double>` *refPointDM*)

Methods

computeDensityEstimator

```
public void computeDensityEstimator ()
```

getComparator

```
public Comparator<S> getComparator ()
```

sortByDensityEstimator

```
public void sortByDensityEstimator ()
```

2.82.2 HypervolumeArchiveWithReferencePoint

```
public class HypervolumeArchiveWithReferencePoint<S extends Solution<?>> extends ArchiveWithReferencePoint<S>  
    Class representing a ArchiveWithReferencePoint archive using a hypervolume contribution based density estimator.
```

Author Antonio J. Nebro

Constructors

HypervolumeArchiveWithReferencePoint

```
public HypervolumeArchiveWithReferencePoint (int maxSize, List<Double> refPointDM)
```

Methods

computeDensityEstimator

```
public void computeDensityEstimator ()
```

getComparator

```
public Comparator<S> getComparator ()
```

sortByDensityEstimator

```
public void sortByDensityEstimator ()
```

2.83 org.uma.jmetal.util.artificialdecisionmaker

2.83.1 ArtificialDecisionMaker

public abstract class **ArtificialDecisionMaker**<S, R> implements *Algorithm*<R>

Fields

algorithm

protected *InteractiveAlgorithm*<S, R> **algorithm**

indexOfRelevantObjectiveFunctions

protected *List*<*Integer*> **indexOfRelevantObjectiveFunctions**

paretoOptimalSolutions

protected *List*<S> **paretoOptimalSolutions**

problem

protected *Problem*<S> **problem**

Constructors

ArtificialDecisionMaker

public **ArtificialDecisionMaker** (*Problem*<S> *problem*, *InteractiveAlgorithm*<S, R> *algorithm*)

Methods

calculateReferencePoints

protected abstract *List*<*Double*> **calculateReferencePoints** (*List*<*Integer*> *indexOfRelevantObjectiveFunctions*, R *front*, *List*<S> *paretoOptimalSolutions*)

generatePreferenceInformation

protected abstract *List*<*Double*> **generatePreferenceInformation** ()

getDescription

public *String* **getDescription** ()

getDistances

```
public abstract List<Double> getDistances ()
```

getName

```
public String getName ()
```

getReferencePoints

```
public abstract List<Double> getReferencePoints ()
```

getResult

```
public R getResult ()
```

initProgress

```
protected abstract void initProgress ()
```

isStoppingConditionReached

```
protected abstract boolean isStoppingConditionReached ()
```

relevantObjectiveFunctions

```
protected abstract List<Integer> relevantObjectiveFunctions (R front)
```

run

```
public void run ()
```

updateParetoOptimal

```
protected abstract void updateParetoOptimal (R front, List<S> paretoOptimalSolutions)
```

updateProgress

```
protected abstract void updateProgress ()
```

2.83.2 DecisionTreeEstimator

```
public class DecisionTreeEstimator<S extends Solution<?>>
```

Constructors

DecisionTreeEstimator

```
public DecisionTreeEstimator (List<S> solutionList)
```

Methods

doPrediction

```
public double doPrediction (int index, S testSolution)
```

doPredictionVariable

```
public double doPredictionVariable (int index, S testSolution)
```

2.84 org.uma.jmetal.util.artificialdecisionmaker.impl

2.84.1 ArtificialDecisionMakerDecisionTree

```
public class ArtificialDecisionMakerDecisionTree<S extends Solution<?>> extends ArtificialDecisionMaker<S, List<S>
```

Class implementing the Towards automatic testing of reference point based interactive methods described in: Ojalehto, V., Podkopaev, D., & Miettinen, K. (2016, September). Towards automatic testing of reference point based interactive methods. In International Conference on Parallel Problem Solving from Nature (pp. 483-492). Springer, Cham.

Author Cristobal Barba

Fields

allReferencePoints

```
protected List<Double> allReferencePoints
```

asp

```
protected List<Double> asp
```

considerationProbability

```
protected double considerationProbability
```

currentReferencePoint

```
protected List<Double> currentReferencePoint
```

distances

protected List<Double> **distances**

evaluations

protected int **evaluations**

idealObjectiveVector

protected List<Double> **idealObjectiveVector**

maxEvaluations

protected int **maxEvaluations**

nadirObjectiveVector

protected List<Double> **nadirObjectiveVector**

numberOfObjectives

protected int **numberOfObjectives**

random

protected *JMetalRandom* **random**

rankingCoefficient

protected List<Double> **rankingCoefficient**

tolerance

protected double **tolerance**

varyingProbability

protected double **varyingProbability**

Constructors

ArtificialDecisionMakerDecisionTree

```
public ArtificialDecisionMakerDecisionTree (Problem<S> problem, InteractiveAlgorithm<S,  
List<S>> algorithm, double considerationProbability, double tolerance, int maxEvaluations,  
List<Double> rankingCoefficient, List<Double> asp)
```

Methods

calculateReferencePoints

```
protected List<Double> calculateReferencePoints (List<Integer> indexOfRelevantObjectiveFunctions,  
List<S> front, List<S> paretoOptimalSolutions)
```

generatePreferenceInformation

```
protected List<Double> generatePreferenceInformation ()
```

getDistances

```
public List<Double> getDistances ()
```

getReferencePoints

```
public List<Double> getReferencePoints ()
```

initProgress

```
protected void initProgress ()
```

isStoppingConditionReached

```
protected boolean isStoppingConditionReached ()
```

relevantObjectiveFunctions

```
protected List<Integer> relevantObjectiveFunctions (List<S> front)
```

updateParetoOptimal

```
protected void updateParetoOptimal (List<S> front, List<S> paretoOptimalSolutions)
```

updateProgress

```
protected void updateProgress ()
```

2.84.2 ArtificialDecisionMakerBuilder

```
public class ArtificialDecisionMakerBuilder<S extends Solution<?>> implements AlgorithmBuilder<ArtificialDecisionMakerBuilder<S>>
```

Author Antonio J. Nebro

Constructors**ArtificialDecisionMakerBuilder**

```
public ArtificialDecisionMakerBuilder (Problem<S> problem, InteractiveAlgorithm<S, List<S>> algorithm)
    ArtificialDecisionMakerBuilder constructor
```

Methods**build**

```
public ArtificialDecisionMakerDecisionTree<S> build ()
```

getMaxIterations

```
public int getMaxIterations ()
```

getProblem

```
public Problem<S> getProblem ()
```

setAlgorithm

```
public ArtificialDecisionMakerBuilder<S> setAlgorithm (InteractiveAlgorithm<S, List<S>> algorithm)
```

setAsp

```
public ArtificialDecisionMakerBuilder<S> setAsp (List<Double> asp)
```

setConsiderationProbability

```
public ArtificialDecisionMakerBuilder<S> setConsiderationProbability (double considerationProbability)
```

setMaxEvaluations

public *ArtificiallDecisionMakerBuilder*<S> **setMaxEvaluations** (int *maxEvaluations*)

setNumberReferencePoints

public *ArtificiallDecisionMakerBuilder*<S> **setNumberReferencePoints** (int *numberReferencePoints*)

setRankingCoefficient

public *ArtificiallDecisionMakerBuilder*<S> **setRankingCoefficient** (*List*<*Double*> *rankingCoefficient*)

setTolerance

public *ArtificiallDecisionMakerBuilder*<S> **setTolerance** (double *tolerance*)

2.85 org.uma.jmetal.util.binarySet

2.85.1 BinarySet

public class **BinarySet** extends *BitSet*

Class representing a bit set including a method to get the total number of bits

Author Antonio J. Nebro

Constructors**BinarySet**

public **BinarySet** (int *numberOfBits*)

Constructor

Parameters

- **numberOfBits** –

Methods**getBinarySetLength**

public int **getBinarySetLength** ()

Returns the total number of bits

Returns the number of bits of the binary set

2.86 org.uma.jmetal.util.chartcontainer

2.86.1 ChartContainer

public class **ChartContainer**

Class for configuring and displaying a XChart.

Author Jorge Rodriguez Ordonez

Constructors

ChartContainer

public **ChartContainer** (*String name*)

ChartContainer

public **ChartContainer** (*String name*, int *delay*)

Methods

addIndicatorChart

public void **addIndicatorChart** (*String indicator*)

getChart

public XYChart **getChart** (*String chartName*)

getDelay

public int **getDelay** ()

getFrontChart

public XYChart **getFrontChart** ()

getName

public *String* **getName** ()

getVarChart

public XYChart **getVarChart** ()

initChart

public void **initChart** ()

refreshCharts

public void **refreshCharts** ()

refreshCharts

public void **refreshCharts** (int *delay*)

removeIndicator

public void **removeIndicator** (String *indicator*)

repaint

public void **repaint** ()

saveChart

public void **saveChart** (String *fileName*, BitmapFormat *format*)

setDelay

public *ChartContainer* **setDelay** (int *delay*)

setFrontChart

public void **setFrontChart** (int *objective1*, int *objective2*)

setFrontChart

public void **setFrontChart** (int *objective1*, int *objective2*, String *referenceFrontFileName*)

setName

public *ChartContainer* **setName** (String *name*)

setReferencePoint

public void **setReferencePoint** (List<Double> *referencePoint*)

setVarChart

```
public void setVarChart (int variable1, int variable2)
```

updateFrontCharts

```
public void updateFrontCharts (List<DoubleSolution> solutionList)
```

updateIndicatorChart

```
public void updateIndicatorChart (String indicator, Double value)
```

2.86.2 ChartContainerWithReferencePoints

```
public class ChartContainerWithReferencePoints
```

Class for configuring and displaying a chart with a number of subpopulations and reference points. Designed to be used with the SMPSORP.

Author Antonio J. Nebro

Constructors

ChartContainerWithReferencePoints

```
public ChartContainerWithReferencePoints (String name)
```

ChartContainerWithReferencePoints

```
public ChartContainerWithReferencePoints (String name, int delay)
```

Methods

getChart

```
public XYChart getChart (String chartName)
```

getDelay

```
public int getDelay ()
```

getFrontChart

```
public XYChart getFrontChart ()
```

getName

```
public String getName ()
```

getVarChart

```
public XYChart getVarChart ()
```

initChart

```
public void initChart ()
```

refreshCharts

```
public void refreshCharts ()
```

refreshCharts

```
public void refreshCharts (int delay)
```

repaint

```
public void repaint ()
```

saveChart

```
public void saveChart (String fileName, BitmapFormat format)
```

setDelay

```
public ChartContainerWithReferencePoints setDelay (int delay)
```

setFrontChart

```
public void setFrontChart (int objective1, int objective2)
```

setFrontChart

```
public void setFrontChart (int objective1, int objective2, String referenceFrontFileName)
```

setName

```
public ChartContainerWithReferencePoints setName (String name)
```

setReferencePoint

public synchronized void **setReferencePoint** (*List<List<Double>> referencePoint*)

updateFrontCharts

public void **updateFrontCharts** (*List<DoubleSolution> solutionList*)

updateReferencePoint

public synchronized void **updateReferencePoint** (*List<List<Double>> referencePoint*)

2.87 org.uma.jmetal.util.comparator

2.87.1 ConstraintViolationComparator

public interface **ConstraintViolationComparator**<S> extends *Comparator<S>*, *Serializable*

Methods**compare**

public int **compare** (*S solution1*, *S solution2*)

2.87.2 CrowdingDistanceComparator

public class **CrowdingDistanceComparator**<S extends *Solution*<?>> implements *Comparator<S>*, *Serializable*
Compares two solutions according to the crowding distance attribute. The higher the distance the better

Author Antonio J. Nebro

Methods**compare**

public int **compare** (*S solution1*, *S solution2*)

Compare two solutions.

Parameters

- **solution1** – Object representing the first *Solution*.
- **solution2** – Object representing the second *Solution*.

Returns -1, or 0, or 1 if solution1 is has greater, equal, or less distance value than solution2, respectively.

2.87.3 CrowdingDistanceComparatorTest

public class **CrowdingDistanceComparatorTest**

Author Antonio J. Nebro

Methods

setup

public void **setup** ()

shouldCompareReturnMinusOnelfSolutionBHasHigherDistance

public void **shouldCompareReturnMinusOneIfSolutionBHasHigherDistance** ()

shouldCompareReturnMinusOnelfTheSecondSolutionIsNull

public void **shouldCompareReturnMinusOneIfTheSecondSolutionIsNull** ()

shouldCompareReturnOnelfSolutionAHasLessDistance

public void **shouldCompareReturnOneIfSolutionAHasLessDistance** ()

shouldCompareReturnOnelfTheFirstSolutionIsNull

public void **shouldCompareReturnOneIfTheFirstSolutionIsNull** ()

shouldCompareReturnZeroIfBothSolutionsAreNull

public void **shouldCompareReturnZeroIfBothSolutionsAreNull** ()

shouldCompareReturnZeroIfBothSolutionsHaveNoCrowdingDistanceAttribute

public void **shouldCompareReturnZeroIfBothSolutionsHaveNoCrowdingDistanceAttribute** ()

shouldCompareReturnZeroIfBothSolutionsHaveTheSameDistance

public void **shouldCompareReturnZeroIfBothSolutionsHaveTheSameDistance** ()

2.87.4 DominanceComparator

public class **DominanceComparator**<S extends Solution<?>> implements [Comparator](#)<S>, [Serializable](#)

This class implements a solution comparator taking into account the violation constraints

Author Antonio J. Nebro

Constructors

DominanceComparator

```
public DominanceComparator ()
    Constructor
```

DominanceComparator

```
public DominanceComparator (double epsilon)
    Constructor
```

DominanceComparator

```
public DominanceComparator (ConstraintViolationComparator<S> constraintComparator)
    Constructor
```

DominanceComparator

```
public DominanceComparator (ConstraintViolationComparator<S> constraintComparator, double epsilon)
    Constructor
```

Methods

compare

```
public int compare (S solution1, S solution2)
    Compares two solutions.
```

Parameters

- **solution1** – Object representing the first *Solution*.
- **solution2** – Object representing the second *Solution*.

Returns -1, or 0, or 1 if solution1 dominates solution2, both are non-dominated, or solution1 is dominated by solution2, respectively.

2.87.5 DominanceComparatorTest

```
public class DominanceComparatorTest
```

Author Antonio J. Nebro

Fields

exception

```
public ExpectedException exception
```

Methods

shouldCompareRaiseAnExceptionIfTheFirstSolutionIsNull

```
public void shouldCompareRaiseAnExceptionIfTheFirstSolutionIsNull ()
```

shouldCompareRaiseAnExceptionIfTheSecondSolutionIsNull

```
public void shouldCompareRaiseAnExceptionIfTheSecondSolutionIsNull ()
```

shouldCompareRaiseAnExceptionIfTheSolutionsHaveNotTheSameNumberOfObjectives

```
public void shouldCompareRaiseAnExceptionIfTheSolutionsHaveNotTheSameNumberOfObjectives ()
```

shouldCompareReturnMinusOneIfTheFirstSolutionDominatesTheSecondOneCaseA

```
public void shouldCompareReturnMinusOneIfTheFirstSolutionDominatesTheSecondOneCaseA ()  
    Case A: solution1 has objectives [-1.0, 5.0, 9.0] and solution2 has [2.0, 6.0, 15.0]
```

shouldCompareReturnMinusOneIfTheFirstSolutionDominatesTheSecondOneCaseB

```
public void shouldCompareReturnMinusOneIfTheFirstSolutionDominatesTheSecondOneCaseB ()  
    Case B: solution1 has objectives [-1.0, 5.0, 9.0] and solution2 has [-1.0, 5.0, 10.0]
```

shouldCompareReturnMinusOneIfTheTwoSolutionsHasOneObjectiveAndTheFirstOnesLower

```
public void shouldCompareReturnMinusOneIfTheTwoSolutionsHasOneObjectiveAndTheFirstOneIsLower ()
```

shouldCompareReturnOneIfTheSecondSolutionDominatesTheFirstOneCaseC

```
public void shouldCompareReturnOneIfTheSecondSolutionDominatesTheFirstOneCaseC ()  
    Case C: solution1 has objectives [-1.0, 5.0, 9.0] and solution2 has [-2.0, 5.0, 9.0]
```

shouldCompareReturnOneIfTheSecondSolutionDominatesTheFirstOneCaseD

```
public void shouldCompareReturnOneIfTheSecondSolutionDominatesTheFirstOneCaseD ()  
    Case D: solution1 has objectives [-1.0, 5.0, 9.0] and solution2 has [-1.0, 5.0, 8.0]
```

shouldCompareReturnOneIfTheTwoSolutionsHasOneObjectiveAndTheSecondOnesLower

```
public void shouldCompareReturnOneIfTheTwoSolutionsHasOneObjectiveAndTheSecondOneIsLower ()
```

shouldCompareReturnTheValueReturnedByTheConstraintViolationComparator

```
public void shouldCompareReturnTheValueReturnedByTheConstraintViolationComparator ()
```


shouldCompareReturnZeroIfTheTwoSolutionsHaveOneObjectiveWithTheSameValue

```
public void shouldCompareReturnZeroIfTheTwoSolutionsHaveOneObjectiveWithTheSameValue ()
```

2.87.6 EqualSolutionsComparator

```
public class EqualSolutionsComparator<S extends Solution<?>> implements Comparator<S>, Serializable
```

This class implements a **Comparator** (a method for comparing **Solution** objects) based whether all the objective values are equal or not. A dominance test is applied to decide about what solution is the best.

Author Antonio J. Nebro

Methods

compare

```
public int compare (S solution1, S solution2)
```

Compares two solutions.

Parameters

- **solution1** – First **Solution**.
- **solution2** – Second **Solution**.

Returns -1, or 0, or 1, or 2 if solution1 dominates solution2, solution1 and solution2 are equals, or solution1 is greater than solution2, respectively.

2.87.7 FitnessComparator

```
public class FitnessComparator<S extends Solution<?>> implements Comparator<S>, Serializable
```

This class implements a **Comparator** (a method for comparing **Solution** objects) based on the fitness value returned by the method `getFitness`.

Author Antonio J. Nebro

Methods

compare

```
public int compare (S solution1, S solution2)
```

Compares two solutions.

Parameters

- **solution1** – Object representing the first **Solution**.
- **solution2** – Object representing the second **Solution**.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.87.8 GDominanceComparator

public class **GDominanceComparator**<S extends Solution<?>> implements **Comparator**<S>, **Serializable**

This class implements a solution comparator according to the concept of g-dominance (<https://doi.org/10.1016/j.ejor.2008.07.015>)

Author Antonio J. Nebro

Constructors

GDominanceComparator

public **GDominanceComparator** (**List**<**Double**> *referencePoint*)
Constructor

Methods

compare

public int **compare** (S *solution1*, S *solution2*)
Compares two solutions.

Parameters

- **solution1** – Object representing the first **Solution**.
- **solution2** – Object representing the second **Solution**.

Returns -1, or 0, or 1 if solution1 dominates solution2, both are non-dominated, or solution1 is dominated by solution2, respectively.

2.87.9 HypervolumeContributionComparator

public class **HypervolumeContributionComparator**<S extends Solution<?>> implements **Comparator**<S>, **Serializable**

Compares two solutions according to the crowding distance attribute. The higher the distance the better

Author Antonio J. Nebro

Methods

compare

public int **compare** (S *solution1*, S *solution2*)
Compare two solutions.

Parameters

- **solution1** – Object representing the first **Solution**.
- **solution2** – Object representing the second **Solution**.

Returns -1, or 0, or 1 if solution1 is has lower, equal, or higher contribution value than solution2, respectively.

2.87.10 ObjectiveComparator

public class **ObjectiveComparator**<S extends Solution<?>> implements [Comparator](#)<S>, [Serializable](#)
 This class implements a comparator based on a given objective

Author Antonio J. Nebro

Constructors

ObjectiveComparator

public **ObjectiveComparator** (int *objectiveId*)
 Constructor.

Parameters

- **objectiveId** – The index of the objective to compare

ObjectiveComparator

public **ObjectiveComparator** (int *objectiveId*, [Ordering](#) *order*)
 Comparator.

Parameters

- **objectiveId** – The index of the objective to compare
- **order** – Ascending or descending order

Methods

compare

public int **compare** (S *solution1*, S *solution2*)
 Compares two solutions according to a given objective.

Parameters

- **solution1** – The first solution
- **solution2** – The second solution

Returns -1, or 0, or 1 if solution1 is less than, equal, or greater than solution2, respectively, according to the established order

2.87.11 ObjectiveComparator.Ordering

public enum **Ordering**

Enum Constants

ASCENDING

public static final [ObjectiveComparator.Ordering](#) **ASCENDING**

DESCENDING

public static final *ObjectiveComparator.Ordering* **DESCENDING**

2.87.12 ObjectiveComparatorTest

public class **ObjectiveComparatorTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldCompareRaiseAnExceptionIfSolution1HasLessObjectivesThanTheOneRequested

public void **shouldCompareRaiseAnExceptionIfSolution1HasLessObjectivesThanTheOneRequested**()

shouldCompareRaiseAnExceptionIfSolution2HasLessObjectivesThanTheOneRequested

public void **shouldCompareRaiseAnExceptionIfSolution2HasLessObjectivesThanTheOneRequested**()

shouldCompareReturnMinusOneIfTheObjectiveOfSolution1IsGreaterInDescendingOrder

public void **shouldCompareReturnMinusOneIfTheObjectiveOfSolution1IsGreaterInDescendingOrder**()

shouldCompareReturnMinusOneIfTheObjectiveOfSolution1IsLower

public void **shouldCompareReturnMinusOneIfTheObjectiveOfSolution1IsLower**()

shouldCompareReturnMinusOneIfTheSecondSolutionIsNull

public void **shouldCompareReturnMinusOneIfTheSecondSolutionIsNull**()

shouldCompareReturnOneIfTheFirstSolutionIsNull

public void **shouldCompareReturnOneIfTheFirstSolutionIsNull**()

shouldCompareReturnOneIfTheObjectiveOfSolution2IsGreaterInDescendingOrder

public void **shouldCompareReturnOneIfTheObjectiveOfSolution2IsGreaterInDescendingOrder**()

shouldCompareReturnOneIfTheObjectiveOfSolution2IsLower

```
public void shouldCompareReturnOneIfTheObjectiveOfSolution2IsLower ()
```

shouldCompareReturnZeroIfBothSolutionsAreNull

```
public void shouldCompareReturnZeroIfBothSolutionsAreNull ()
```

shouldCompareReturnZeroIfTheObjectiveOfTheSolutionsIsTheSame

```
public void shouldCompareReturnZeroIfTheObjectiveOfTheSolutionsIsTheSame ()
```

2.87.13 RankingAndCrowdingDistanceComparator

```
public class RankingAndCrowdingDistanceComparator<S extends Solution<?>> implements Comparator<S>, Serializable
```

Author Antonio J. Nebro

Methods**compare**

```
public int compare (S solution1, S solution2)
```

Compares two solutions.

Parameters

- **solution1** – Object representing the first solution
- **solution2** – Object representing the second solution.

Returns -1, or 0, or 1 if solution1 is less than, equal, or greater than solution2, respectively.

2.87.14 RankingAndCrowdingDistanceComparatorTest

```
public class RankingAndCrowdingDistanceComparatorTest
```

Author Antonio J. Nebro

Methods**setup**

```
public void setup ()
```

shouldCompareTwoNullSolutionsReturnZero

```
public void shouldCompareTwoNullSolutionsReturnZero ()
```

shouldCompareWhenRankingYieldingAZeroReturnTheCrowdingDistanceValue

```
public void shouldCompareWhenRankingYieldingAZeroReturnTheCrowdingDistanceValue ()
```

shouldCompareWithANullSolutionAsFirstArgumentReturnOne

```
public void shouldCompareWithANullSolutionAsFirstArgumentReturnOne ()
```

shouldCompareWithANullSolutionAsSecondArgumentReturnMinusOne

```
public void shouldCompareWithANullSolutionAsSecondArgumentReturnMinusOne ()
```

shouldCompareWithNullRankingAttributeSolutionAsFirstArgumentReturnOne

```
public void shouldCompareWithNullRankingAttributeSolutionAsFirstArgumentReturnOne ()
```

shouldCompareWithRankingYieldingANonZeroValueReturnThatValue

```
public void shouldCompareWithRankingYieldingANonZeroValueReturnThatValue ()
```

teardown

```
public void teardown ()
```

2.87.15 RankingComparator

```
public class RankingComparator<S extends Solution<?>> implements Comparator<S>, Serializable
```

Author Antonio J. Nebro This class implements a comparator based on the rank of the solutions.

Methods

compare

```
public int compare (S solution1, S solution2)
```

Compares two solutions according to the ranking attribute. The lower the ranking the better

Parameters

- **solution1** – Object representing the first solution.
- **solution2** – Object representing the second solution.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.87.16 RankingComparatorTest

public class **RankingComparatorTest**

Author Antonio J. Nebro

Methods

setup

public void **setup** ()

shouldCompareReturnMinusOnelfSolutionAHasLessRanking

public void **shouldCompareReturnMinusOneIfSolutionAHasLessRanking** ()

shouldCompareReturnMinusOnelfTheSecondSolutionIsNull

public void **shouldCompareReturnMinusOneIfTheSecondSolutionIsNull** ()

shouldCompareReturnOnelfSolutionBHasLessRanking

public void **shouldCompareReturnOneIfSolutionBHasLessRanking** ()

shouldCompareReturnOnelfTheFirstSolutionIsNull

public void **shouldCompareReturnOneIfTheFirstSolutionIsNull** ()

shouldCompareReturnZeroIfBothSolutionsAreNull

public void **shouldCompareReturnZeroIfBothSolutionsAreNull** ()

shouldCompareReturnZeroIfBothSolutionsHaveNoRankingAttribute

public void **shouldCompareReturnZeroIfBothSolutionsHaveNoRankingAttribute** ()

shouldCompareReturnZeroIfBothSolutionsHaveTheSameRanking

public void **shouldCompareReturnZeroIfBothSolutionsHaveTheSameRanking** ()

2.87.17 StrengthFitnessComparator

public class **StrengthFitnessComparator**<S extends [Solution](#)<?>> implements [Comparator](#)<S>, [Serializable](#)

Author Juan J. Durillo

Parameters

- <S> –

Methods

compare

public int **compare** (S *solution1*, S *solution2*)

2.88 org.uma.jmetal.util.comparator.impl

2.88.1 OverallConstraintViolationComparator

public class **OverallConstraintViolationComparator**<S extends [Solution](#)<?>> implements [ConstraintViolationComparator](#)

This class implements a [Comparator](#) (a method for comparing [Solution](#) objects) based on the overall constraint violation of the solutions, as done in NSGA-II.

Author Antonio J. Nebro

Constructors

OverallConstraintViolationComparator

public **OverallConstraintViolationComparator** ()
Constructor

Methods

compare

public int **compare** (S *solution1*, S *solution2*)
Compares two solutions. If the solutions has no constraints the method return 0

Parameters

- **solution1** – Object representing the first [Solution](#).
- **solution2** – Object representing the second [Solution](#).

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.88.2 ViolationThresholdComparator

public class **ViolationThresholdComparator**<S extends Solution<?>> implements *ConstraintViolationComparator*<S>
 This class implements the ViolationThreshold Comparator *

Author Juan J. Durillo

Constructors

ViolationThresholdComparator

public **ViolationThresholdComparator** ()
 Constructor

Methods

compare

public int **compare** (S *solution1*, S *solution2*)
 Compares two solutions. If the solutions has no constraints the method return 0

Parameters

- **solution1** – Object representing the first *Solution*.
- **solution2** – Object representing the second *Solution*.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

feasibilityRatio

public double **feasibilityRatio** (List<S> *solutionSet*)
 Computes the feasibility ratio Return the ratio of feasible solutions

meanOverallViolation

public double **meanOverallViolation** (List<S> *solutionSet*)
 Computes the feasibility ratio Return the ratio of feasible solutions

needToCompare

public boolean **needToCompare** (S *solution1*, S *solution2*)
 Returns true if solutions s1 and/or s2 have an overall constraint violation with value less than 0

updateThreshold

public void **updateThreshold** (List<S> *set*)
 Updates the threshold value using the population

2.89 org.uma.jmetal.util.distance

2.89.1 Distance

public interface **Distance**<E, J>

Interface representing distances between two entities

Author

Methods

getDistance

double **getDistance** (E *element1*, J *element2*)

2.90 org.uma.jmetal.util.distance.impl

2.90.1 CosineDistanceBetweenSolutionsInObjectiveSpace

public class **CosineDistanceBetweenSolutionsInObjectiveSpace**<S extends Solution<?>> implements *Distance*<S, S>

Class for calculating the cosine distance between two *Solution* objects in objective space.

Author

Constructors

CosineDistanceBetweenSolutionsInObjectiveSpace

public **CosineDistanceBetweenSolutionsInObjectiveSpace** (S *referencePoint*)

Methods

getDistance

public double **getDistance** (S *solution1*, S *solution2*)

2.90.2 CosineDistanceBetweenSolutionsInObjectiveSpaceTest

public class **CosineDistanceBetweenSolutionsInObjectiveSpaceTest**

Created by ajnebro on 12/2/16.

Methods

shouldIdenticalPointsHaveADistanceOfOne

public void **shouldIdenticalPointsHaveADistanceOfOne** ()

shouldPointsInTheSameDirectionHaveADistanceOfOne

```
public void shouldPointsInTheSameDirectionHaveADistanceOfOne ()
```

shouldTwoPerpendicularPointsHaveADistanceOfZero

```
public void shouldTwoPerpendicularPointsHaveADistanceOfZero ()
```

2.90.3 EuclideanDistanceBetweenSolutionAndASolutionListInObjectiveSpace

public class **EuclideanDistanceBetweenSolutionAndASolutionListInObjectiveSpace**<S extends Solution<Double>>
 Class for calculating the Euclidean distance between a *Solution* object a list of *Solution* objects in objective space.

Author

Constructors**EuclideanDistanceBetweenSolutionAndASolutionListInObjectiveSpace**

```
public EuclideanDistanceBetweenSolutionAndASolutionListInObjectiveSpace ()
```

Methods**getDistance**

```
public double getDistance (S solution, L solutionList)
```

2.90.4 EuclideanDistanceBetweenSolutionsInObjectiveSpace

public class **EuclideanDistanceBetweenSolutionsInObjectiveSpace**<S extends Solution<?>> implements *Distance*<S>
 Class for calculating the Euclidean distance between two *Solution* objects in objective space.

Author

Methods**getDistance**

```
public double getDistance (S solution1, S solution2)
```

2.90.5 EuclideanDistanceBetweenSolutionsInSolutionSpace

public class **EuclideanDistanceBetweenSolutionsInSolutionSpace**<S extends Solution<Double>> implements *Distance*<S>
 Class for calculating the Euclidean distance between two *DoubleSolution* objects in solution space.

Author

Methods

getDistance

public double **getDistance** (*S solution1*, *S solution2*)

2.91 org.uma.jmetal.util.evaluator

2.91.1 SolutionListEvaluator

public interface **SolutionListEvaluator**<S> extends [Serializable](#)
Created by Antonio J. Nebro on 30/05/14.

Methods

evaluate

[List](#)<S> **evaluate** ([List](#)<S> *solutionList*, [Problem](#)<S> *problem*)

shutdown

void **shutdown** ()

2.92 org.uma.jmetal.util.evaluator.impl

2.92.1 MultithreadedSolutionListEvaluator

public class **MultithreadedSolutionListEvaluator**<S> implements [SolutionListEvaluator](#)<S>
Author Antonio J. Nebro

Constructors

MultithreadedSolutionListEvaluator

public **MultithreadedSolutionListEvaluator** (int *numberOfThreads*, [Problem](#)<S> *problem*)

Methods

evaluate

public [List](#)<S> **evaluate** ([List](#)<S> *solutionList*, [Problem](#)<S> *problem*)

getNumberOfThreads

```
public int getNumberOfThreads ()
```

shutdown

```
public void shutdown ()
```

2.92.2 SequentialSolutionListEvaluator

```
public class SequentialSolutionListEvaluator<S> implements SolutionListEvaluator<S>
```

Author Antonio J. Nebro

Methods

evaluate

```
public List<S> evaluate (List<S> solutionList, Problem<S> problem)
```

shutdown

```
public void shutdown ()
```

2.93 org.uma.jmetal.util.experiment

2.93.1 Experiment

```
public class Experiment<S extends Solution<?>, Result>
```

Class for describing the configuration of a jMetal experiment. Created by Antonio J. Nebro on 17/07/14.

Constructors

Experiment

```
public Experiment (ExperimentBuilder<S, Result> builder)
```

Constructor

Methods

getAlgorithmList

```
public List<ExperimentAlgorithm<S, Result>> getAlgorithmList ()
```

getExperimentBaseDirectory

```
public String getExperimentBaseDirectory ()
```

getExperimentName

```
public String getExperimentName ()
```

getIndependentRuns

```
public int getIndependentRuns ()
```

getIndicatorList

```
public List<GenericIndicator<S>> getIndicatorList ()
```

getNumberOfCores

```
public int getNumberOfCores ()
```

getOutputParetoFrontFileName

```
public String getOutputParetoFrontFileName ()
```

getOutputParetoSetFileName

```
public String getOutputParetoSetFileName ()
```

getProblemList

```
public List<ExperimentProblem<S>> getProblemList ()
```

getReferenceFrontDirectory

```
public String getReferenceFrontDirectory ()
```

removeDuplicatedAlgorithms

```
public void removeDuplicatedAlgorithms ()
```

The list of algorithms contain an algorithm instance per problem. This is not convenient for calculating statistical data, because a same algorithm will appear many times. This method remove duplicated algorithms and leave only an instance of each one.

setAlgorithmList

```
public void setAlgorithmList (List<ExperimentAlgorithm<S, Result>> algorithmList)
```

setReferenceFrontDirectory

```
public void setReferenceFrontDirectory (String referenceFrontDirectory)
```

2.93.2 ExperimentBuilder

```
public class ExperimentBuilder<S extends Solution<?>, Result>  
    Builder for class Experiment
```

Author Antonio J. Nebro

Constructors

ExperimentBuilder

```
public ExperimentBuilder (String experimentName)
```

Methods

build

```
public Experiment<S, Result> build ()
```

getAlgorithmList

```
public List<ExperimentAlgorithm<S, Result>> getAlgorithmList ()
```

getExperimentBaseDirectory

```
public String getExperimentBaseDirectory ()
```

getExperimentName

```
public String getExperimentName ()
```

getIndependentRuns

```
public int getIndependentRuns ()
```

getIndicatorList

```
public List<GenericIndicator<S>> getIndicatorList ()
```

getNumberOfCores

```
public int getNumberOfCores ()
```

getOutputParetoFrontFileName

```
public String getOutputParetoFrontFileName ()
```

getOutputParetoSetFileName

```
public String getOutputParetoSetFileName ()
```

getProblemList

```
public List<ExperimentProblem<S>> getProblemList ()
```

getReferenceFrontDirectory

```
public String getReferenceFrontDirectory ()
```

setAlgorithmList

```
public ExperimentBuilder<S, Result> setAlgorithmList (List<ExperimentAlgorithm<S, Result>> algorithmList)
```

setExperimentBaseDirectory

```
public ExperimentBuilder<S, Result> setExperimentBaseDirectory (String experimentBaseDirectory)
```

setIndependentRuns

```
public ExperimentBuilder<S, Result> setIndependentRuns (int independentRuns)
```

setIndicatorList

```
public ExperimentBuilder<S, Result> setIndicatorList (List<GenericIndicator<S>> indicatorList)
```


setNumberOfCores

```
public ExperimentBuilder<S, Result> setNumberOfCores (int numberOfCores)
```

setOutputParetoFrontFileName

```
public ExperimentBuilder<S, Result> setOutputParetoFrontFileName (String outputParetoFront-  
FileName)
```

setOutputParetoSetFileName

```
public ExperimentBuilder<S, Result> setOutputParetoSetFileName (String outputParetoSetFile-  
Name)
```

setProblemList

```
public ExperimentBuilder<S, Result> setProblemList (List<ExperimentProblem<S>> problemList)
```

setReferenceFrontDirectory

```
public ExperimentBuilder<S, Result> setReferenceFrontDirectory (String referenceFrontDirec-  
tory)
```

2.93.3 ExperimentComponent

```
public interface ExperimentComponent
```

An experiment is composed of instances of this interface.

Author Antonio J. Nebro

Methods**run**

```
void run ()
```

2.94 org.uma.jmetal.util.experiment.component**2.94.1 ComputeQualityIndicators**

```
public class ComputeQualityIndicators<S extends Solution<?>, Result> implements ExperimentComponent
```

This class computes the *QualityIndicators* of an experiment. Once the algorithms of an experiment have been executed through running an instance of class *ExecuteAlgorithms*, the list of indicators is obtained from the *#getIndicatorsList()* method. Then, for every combination algorithm + problem, the indicators are applied to all the FUN files and the resulting values are store in a file called as *#getName()*, which is located in the same directory of the FUN files.

Author Antonio J. Nebro

Constructors

ComputeQualityIndicators

```
public ComputeQualityIndicators (Experiment<S, Result> experiment)
```

Methods

findBestIndicatorFronts

```
public void findBestIndicatorFronts (Experiment<?, Result> experiment)
```

run

```
public void run ()
```

2.94.2 ExecuteAlgorithms

```
public class ExecuteAlgorithms<S extends Solution<?>, Result> implements ExperimentComponent
```

This class executes the algorithms the have been configured with a instance of class *Experiment*. Java 8 parallel streams are used to run the algorithms in parallel.

The result of the execution is a pair of files FUNrunId.tsv and VARrunID.tsv per experiment, which are stored in the directory `#getExperimentBaseDirectory()/algorithmName/problemName`.

Author Antonio J. Nebro

Constructors

ExecuteAlgorithms

```
public ExecuteAlgorithms (Experiment<S, Result> configuration)  
    Constructor
```

Methods

run

```
public void run ()
```

2.94.3 GenerateBoxplotsWithR

```
public class GenerateBoxplotsWithR<Result> implements ExperimentComponent
```

This class generates a R script that generates an eps file containing boxplots The results are a set of R files that are written in the directory `#getExperimentBaseDirectory()/R`. Each file is called as indicator-Name.Wilcoxon.R To run the R script: `Rscript indicatorName.Wilcoxon.R` To generate the resulting Latex file: `pdflatex indicatorName.Wilcoxon.tex`

Author Antonio J. Nebro

Constructors

GenerateBoxplotsWithR

```
public GenerateBoxplotsWithR (Experiment<?, Result> experimentConfiguration)
```

Methods

run

```
public void run ()
```

setColumns

```
public GenerateBoxplotsWithR<Result> setColumns (int columns)
```

setDisplayNotch

```
public GenerateBoxplotsWithR<Result> setDisplayNotch ()
```

setRows

```
public GenerateBoxplotsWithR<Result> setRows (int rows)
```

2.94.4 GenerateFriedmanTestTables

```
public class GenerateFriedmanTestTables<Result> implements ExperimentComponent
```

This class computes the Friedman test ranking and generates a Latex script that produces a table per quality indicator containing the ranking. The results are a set of Latex files that are written in the directory `#getExperimentBaseDirectory()/latex`. Each file is called as `FriedmanTest[indicatorName].tex`. The implementation is based on the one included in Keel: J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, F. Herrera. KEEL: A Software Tool to Assess Evolutionary Algorithms to Data Mining Problems. *Soft Computing* 13:3 (2009) 307-318. Doi: 10.1007/s00500-008-0323-y

Author Antonio J. Nebro

Constructors

GenerateFriedmanTestTables

```
public GenerateFriedmanTestTables (Experiment<?, Result> experimentConfiguration)
```

Methods

prepareFileOutputContents

```
public String prepareFileOutputContents (double[] averageRanking)
```

run

```
public void run ()
```

2.94.5 GenerateLatexTablesWithStatistics

public class **GenerateLatexTablesWithStatistics** implements *ExperimentComponent*

This class computes a number of statistical values (mean, median, standard deviation, interquartile range) from the indicator files generated after executing *ExecuteAlgorithms* and *ComputeQualityIndicators*. After reading the data files and calculating the values, a Latex file is created containing an script that generates tables with the best and second best values per indicator. The name of the file is *#getExperimentName().tex*, which is located by default in the directory *#getExperimentBaseDirectory()/latex* Although the maximum, minimum, and total number of items are also computed, no tables are generated with them (this is a pending work).

Author Antonio J. Nebro

Constructors

GenerateLatexTablesWithStatistics

```
public GenerateLatexTablesWithStatistics (Experiment<?, ?> configuration)
```

Methods

printEndLatexCommands

```
void printEndLatexCommands (String fileName)
```

printHeaderLatexCommands

```
void printHeaderLatexCommands (String fileName)
```

run

```
public void run ()
```

2.94.6 GenerateReferenceParetoFront

public class **GenerateReferenceParetoFront** implements *ExperimentComponent*

This class computes a reference Pareto front from a set of files. Once the algorithms of an experiment have been executed through running an instance of class *ExecuteAlgorithms*, all the obtained fronts of all the algorithms are gathered per problem; then, the dominated solutions are removed and the final result is a file per problem containing the reference Pareto front. By default, the files are stored in a directory called “referenceFront”, which is located in the experiment base directory. Each front is named following the scheme “problemName.rf”.

Author Antonio J. Nebro

Constructors

GenerateReferenceParetoFront

public **GenerateReferenceParetoFront** (*Experiment*<?, ?> *experimentConfiguration*)

Methods

run

public void **run** ()

The run() method creates the output directory and compute the fronts

2.94.7 GenerateReferenceParetoSetAndFrontFromDoubleSolutions

public class **GenerateReferenceParetoSetAndFrontFromDoubleSolutions** implements *ExperimentComponent*

This class computes the reference Pareto set and front from a set of data files containing the variable (VARx.tsv file) and objective (FUNx.tsv) values. A requirement is that the variable values MUST correspond to *DoubleSolution* solutions, i.e., the solved problems must be instances of *DoubleProblem*. Once the algorithms of an experiment have been executed through running an instance of class *ExecuteAlgorithms*, all the obtained fronts of all the algorithms are gathered per problem; then, the dominated solutions are removed thus yielding to the reference Pareto front. By default, the files are stored in a directory called “referenceFront”, which is located in the experiment base directory. The following files are generated per problem: - “problemName.pf”: the reference Pareto front. - “problemName.ps”: the reference Pareto set (i.e., the variable values of the solutions of the reference Pareto front. - “problemName.algorithmName.pf”: the objectives values of the contributed solutions by the algorithm called “algorithmName” to “problemName.pf” - “problemName.algorithmName.ps”: the variable values of the contributed solutions by the algorithm called “algorithmName” to “problemName.ps”

Author Antonio J. Nebro

Constructors

GenerateReferenceParetoSetAndFrontFromDoubleSolutions

public **GenerateReferenceParetoSetAndFrontFromDoubleSolutions** (*Experiment*<?, ?> *experimentConfiguration*)

Methods

run

public void **run** ()

The run() method creates the output directory and compute the fronts

2.94.8 GenerateWilcoxonTestTablesWithR

public class **GenerateWilcoxonTestTablesWithR**<Result> implements *ExperimentComponent*

This class generates a R script that computes the Wilcoxon Signed Rank Test and generates a Latex script that produces a table per quality indicator containing the pairwise comparison between all the algorithms on all the solved problems. The results are a set of R files that are written in the directory *#getExperimentBaseDirectory()/R*. Each file is called as indicatorName.Wilcoxon.R To run the R script: Rscript indicatorName.Wilcoxon.R To generate the resulting Latex file: pdflatex indicatorName.Wilcoxon.tex

Author Antonio J. Nebro

Constructors

GenerateWilcoxonTestTablesWithR

public **GenerateWilcoxonTestTablesWithR** (*Experiment*<?, Result> *experimentConfiguration*)

Methods

run

public void **run** ()

2.95 org.uma.jmetal.util.experiment.util

2.95.1 ExperimentAlgorithm

public class **ExperimentAlgorithm**<S extends Solution<?>, Result>

Class defining tasks for the execution of algorithms in parallel.

Author Antonio J. Nebro

Constructors

ExperimentAlgorithm

public **ExperimentAlgorithm** (*Algorithm*<Result> *algorithm*, *String* *algorithmTag*, *ExperimentProblem*<S> *problem*, int *runId*)

Constructor

ExperimentAlgorithm

```
public ExperimentAlgorithm(Algorithm<Result> algorithm, ExperimentProblem<S> problem, int  
                           runId)
```

Methods

getAlgorithm

```
public Algorithm<Result> getAlgorithm()
```

getAlgorithmTag

```
public String getAlgorithmTag()
```

getProblemTag

```
public String getProblemTag()
```

getReferenceParetoFront

```
public String getReferenceParetoFront()
```

getRunId

```
public int getRunId()
```

runAlgorithm

```
public void runAlgorithm(Experiment<?, ?> experimentData)
```

2.95.2 ExperimentProblem

```
public class ExperimentProblem<S extends Solution<?>>  
    Class used to add a tag field to a problem.
```

Author Antonio J. Nebro

Constructors

ExperimentProblem

```
public ExperimentProblem(Problem<S> problem, String tag)
```

ExperimentProblem

public **ExperimentProblem** (*Problem*<S> *problem*)

Methods

changeReferenceFrontTo

public *ExperimentProblem*<S> **changeReferenceFrontTo** (*String* *referenceFront*)

getProblem

public *Problem*<S> **getProblem** ()

getReferenceFront

public *String* **getReferenceFront** ()

getTag

public *String* **getTag** ()

2.96 org.uma.jmetal.util.extremevalues

2.96.1 ExtremeValuesFinder

public interface **ExtremeValuesFinder**<Source, Result>

Interface representing classes aimed at finding the extreme values of Source objects (e.g., lists)

Author Antonio J. Nebro

Parameters

- <Source> –
- <Result> –

Methods

findHighestValues

Result **findHighestValues** (Source *source*)

findLowestValues

Result **findLowestValues** (Source *source*)

2.97 org.uma.jmetal.util.extremevalues.impl

2.97.1 FrontExtremeValues

public class **FrontExtremeValues** implements *ExtremeValuesFinder*<*Front*, List<Double>>

Class for finding the extreme values of front objects

Author Antonio J. Nebro

Methods

findHighestValues

public List<Double> **findHighestValues** (*Front* front)

findLowestValues

public List<Double> **findLowestValues** (*Front* front)

2.97.2 SolutionListExtremeValues

public class **SolutionListExtremeValues** implements *ExtremeValuesFinder*<List<*Solution*<?>>, List<Double>>

Class for finding the extreme values of a list of objects

Author Antonio J. Nebro

Methods

findHighestValues

public List<Double> **findHighestValues** (List<*Solution*<?>> solutionList)

findLowestValues

public List<Double> **findLowestValues** (List<*Solution*<?>> solutionList)

2.98 org.uma.jmetal.util.fileinput.util

2.98.1 ReadDoubleDataFile

public class **ReadDoubleDataFile**

Utility class that reads a file containing a double value per line and returns an array with all of them.

Author Antonio J. Nebro

Methods

readFile

```
public double[] readFile (String fileName)
```

2.99 org.uma.jmetal.util.fileoutput

2.99.1 FileOutputContext

public interface **FileOutputContext** extends *Serializable*

This interface represents output contexts, which are classes providing a mean for getting a buffer reader object.

Author Antonio J. Nebro

Methods

getFileName

```
public String getFileName ()
```

getFileWriter

```
public BufferedWriter getFileWriter ()
```

getSeparator

```
public String getSeparator ()
```

setSeparator

```
public void setSeparator (String separator)
```

2.99.2 SolutionListOutput

```
public class SolutionListOutput
```

Author Antonio J. Nebro

Constructors

SolutionListOutput

```
public SolutionListOutput (List<? extends Solution<?>> solutionList)
```

Methods

print

```
public void print ()
```

printObjectivesToFile

```
public void printObjectivesToFile (FileOutputContext context, List<? extends Solution<?>> solution-  
List)
```

printObjectivesToFile

```
public void printObjectivesToFile (FileOutputContext context, List<? extends Solution<?>> solution-  
List, List<Boolean> minimizeObjective)
```

printObjectivesToFile

```
public void printObjectivesToFile (String fileName)
```

printObjectivesToFile

```
public void printObjectivesToFile (String fileName, List<Boolean> minimizeObjective)
```

printVariablesToFile

```
public void printVariablesToFile (FileOutputContext context, List<? extends Solution<?>> solution-  
List)
```

printVariablesToFile

```
public void printVariablesToFile (String fileName)
```

setFunFileOutputContext

```
public SolutionListOutput setFunFileOutputContext (FileOutputContext fileContext)
```

setObjectiveMinimizingObjectiveList

```
public SolutionListOutput setObjectiveMinimizingObjectiveList (List<Boolean> isObjective-  
ToBeMinimized)
```

setSeparator

```
public SolutionListOutput setSeparator (String separator)
```

setVarFileOutputContext

public *SolutionListOutput* **setVarFileOutputContext** (*FileOutputContext* *fileContext*)

2.100 org.uma.jmetal.util.fileoutput.impl

2.100.1 DefaultFileOutputContext

public class **DefaultFileOutputContext** implements *FileOutputContext*

Class using the default method for getting a buffered writer

Author Antonio J. Nebro

Fields

fileName

protected *String* **fileName**

separator

protected *String* **separator**

Constructors

DefaultFileOutputContext

public **DefaultFileOutputContext** (*String* *fileName*)

Methods

getFileName

public *String* **getFileName** ()

getFileWriter

public *BufferedWriter* **getFileWriter** ()

getSeparator

public *String* **getSeparator** ()

setSeparator

public void **setSeparator** (*String separator*)

2.101 org.uma.jmetal.util.front

2.101.1 Front

public interface **Front** extends *Serializable*

A front is a list of points

Author Antonio J. Nebro

Methods

getNumberOfPoints

public int **getNumberOfPoints** ()

getPoint

public *Point* **getPoint** (int *index*)

getPointDimensions

public int **getPointDimensions** ()

setPoint

public void **setPoint** (int *index*, *Point point*)

sort

public void **sort** (*Comparator<Point> comparator*)

2.102 org.uma.jmetal.util.front.imp

2.102.1 ArrayFront

public class **ArrayFront** implements *Front*

This class implements the *Front* interface by using an array of *Point* objects

Author Antonio J. Nebro

Fields

numberOfPoints

protected int **numberOfPoints**

points

protected *Point*[] **points**

Constructors

ArrayFront

public **ArrayFront** ()
Constructor

ArrayFront

public **ArrayFront** (List<? extends *Solution*<?>> *solutionList*)
Constructor

ArrayFront

public **ArrayFront** (*Front* *front*)
Copy Constructor

ArrayFront

public **ArrayFront** (int *numberOfPoints*, int *dimensions*)
Constructor

ArrayFront

public **ArrayFront** (*String* *fileName*)
Constructor

Parameters

- **fileName** – File containing the data. Each line of the file is a list of objective values

Throws

- **FileNotFoundException** –

Methods

createInputStream

public InputStream **createInputStream** (*String fileName*)

equals

public boolean **equals** (*Object o*)

getNumberOfPoints

public int **getNumberOfPoints** ()

getPoint

public *Point* **getPoint** (int *index*)

getPointDimensions

public int **getPointDimensions** ()

hashCode

public int **hashCode** ()

setPoint

public void **setPoint** (int *index*, *Point point*)

sort

public void **sort** (Comparator<*Point*> *comparator*)

toString

public *String* **toString** ()

2.102.2 ArrayFrontTest

public class **ArrayFrontTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldConstructorCreateAnArranFrontFromAFileContainingA2DFront

public void **shouldConstructorCreateAnArranFrontFromAFileContainingA2DFront** ()

shouldConstructorCreateAnArranFrontFromAFileContainingA3DFront

public void **shouldConstructorCreateAnArranFrontFromAFileContainingA3DFront** ()

shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneDoubleSolutionObject

public void **shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneDoubleSolutionObject** ()

shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneSingleSolutionObject

public void **shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneSingleSolutionObject** ()

shouldCreateAnArrayFrontFromAListOfSolutionsHavingTwoDoubleSolutionObject

public void **shouldCreateAnArrayFrontFromAListOfSolutionsHavingTwoDoubleSolutionObject** ()

shouldCreateAnArrayFrontFromANullFrontRaiseAnException

public void **shouldCreateAnArrayFrontFromANullFrontRaiseAnException** ()

shouldCreateAnArrayFrontFromANullListRaiseAnAnException

public void **shouldCreateAnArrayFrontFromANullListRaiseAnAnException** ()

shouldCreateAnArrayFrontFromASolutionListResultInTwoEqualsFronts

public void **shouldCreateAnArrayFrontFromASolutionListResultInTwoEqualsFronts** ()

shouldCreateAnArrayFrontFromAnEmptyFrontRaiseAnException

public void **shouldCreateAnArrayFrontFromAnEmptyFrontRaiseAnException** ()

shouldCreateAnArrayFrontFromAnEmptyListRaiseAnException

```
public void shouldCreateAnArrayFrontFromAnEmptyListRaiseAnException ()
```

shouldCreateAnArrayFrontFromAnotherFrontResultInTwoEqualsFrontssss

```
public void shouldCreateAnArrayFrontFromAnotherFrontResultInTwoEqualsFrontssss ()
```

shouldCreateInputStreamThrownAnExceptionIfFileDoesNotExist

```
public void shouldCreateInputStreamThrownAnExceptionIfFileDoesNotExist ()
```

shouldDefaultConstructorCreateAnEmptyArrayFront

```
public void shouldDefaultConstructorCreateAnEmptyArrayFront ()
```

shouldEqualsReturnFalseIfPointDimensionsOfTheFrontsIsDifferent

```
public void shouldEqualsReturnFalseIfPointDimensionsOfTheFrontsIsDifferent ()
```

shouldEqualsReturnFalseIfTheArgumentIsFromAWrongClass

```
public void shouldEqualsReturnFalseIfTheArgumentIsFromAWrongClass ()
```

shouldEqualsReturnFalseIfTheArgumentIsNull

```
public void shouldEqualsReturnFalseIfTheArgumentIsNull ()
```

shouldEqualsReturnFalseIfTheComparedFrontHasADifferentNumberOfPoints

```
public void shouldEqualsReturnFalseIfTheComparedFrontHasADifferentNumberOfPoints ()
```

shouldEqualsReturnFalseIfTheFrontsAreDifferent

```
public void shouldEqualsReturnFalseIfTheFrontsAreDifferent ()
```

shouldEqualsReturnTrueIfTheArgumentIsEqual

```
public void shouldEqualsReturnTrueIfTheArgumentIsEqual ()
```

shouldEqualsReturnTrueIfTheArgumentIsTheSameObject

```
public void shouldEqualsReturnTrueIfTheArgumentIsTheSameObject ()
```

shouldGetPointRaiseAnExceptionWhenTheIndexIsGreaterThanTheFrontSize

```
public void shouldGetPointRaiseAnExceptionWhenTheIndexIsGreaterThanTheFrontSize ()
```

shouldGetPointRaiseAnExceptionWhenTheIndexIsNegative

```
public void shouldGetPointRaiseAnExceptionWhenTheIndexIsNegative ()
```

shouldGetPointReturnTheCorrectObject

```
public void shouldGetPointReturnTheCorrectObject ()
```

shouldReadFrontAFileWithOnePointCreateTheCorrectFront

```
public void shouldReadFrontAFileWithOnePointCreateTheCorrectFront ()  
    Test using a file containing: 1.0 2.0 -3.0
```

shouldReadFrontAnEmptyFileCreateAnEmptyFront

```
public void shouldReadFrontAnEmptyFileCreateAnEmptyFront ()
```

shouldReadFrontFourPointsCreateTheCorrectFront

```
public void shouldReadFrontFourPointsCreateTheCorrectFront ()  
    Test using a file containing: 1 2 3 4 5 6 7 8 9 10 11 12 -1 -2 -3 -4
```

shouldReadFrontWithALineContainingWrongDataRaiseAnException

```
public void shouldReadFrontWithALineContainingWrongDataRaiseAnException ()  
    Test using a file containing: 3.0 2.3 asdfg
```

shouldReadFrontWithALineWithALineMissingDataRaiseAnException

```
public void shouldReadFrontWithALineWithALineMissingDataRaiseAnException ()  
    Test using a file containing: -30 234.234 90.25 15 -5.23
```

shouldSetPointAssignTheCorrectObject

```
public void shouldSetPointAssignTheCorrectObject ()
```

shouldSetPointRaiseAnExceptionWhenTheIndexIsGreaterThanTheFrontSize

```
public void shouldSetPointRaiseAnExceptionWhenTheIndexIsGreaterThanTheFrontSize ()
```

shouldSetPointRaiseAnExceptionWhenTheIndexIsNegative

```
public void shouldSetPointRaiseAnExceptionWhenTheIndexIsNegative ()
```

shouldSetPointRaiseAnExceptionWhenThePointIsNull

```
public void shouldSetPointRaiseAnExceptionWhenThePointIsNull ()
```

shouldSortReturnAnOrderedFront

```
public void shouldSortReturnAnOrderedFront ()
```

2.102.3 FrontUtilsTest

```
public class FrontUtilsTest
```

Author Antonio J. Nebro

Fields**exception**

```
public ExpectedException exception
```

Methods**shouldConvertFrontToArrayRaiseAnExceptionIfTheFrontIsNull**

```
public void shouldConvertFrontToArrayRaiseAnExceptionIfTheFrontIsNull ()
```

shouldConvertFrontToArrayReturnAnEmptyArrayIfTheFrontIsEmpty

```
public void shouldConvertFrontToArrayReturnAnEmptyArrayIfTheFrontIsEmpty ()
```

shouldConvertFrontToArrayReturnTheCorrectArrayCaseA

```
public void shouldConvertFrontToArrayReturnTheCorrectArrayCaseA ()
```

Case A: The front has one point

shouldConvertFrontToArrayReturnTheCorrectArrayCaseB

```
public void shouldConvertFrontToArrayReturnTheCorrectArrayCaseB ()
```

Case A: The front has one three points

shouldConvertFrontToSolutionListRaiseAnExceptionIfTheFrontIsNull

```
public void shouldConvertFrontToSolutionListRaiseAnExceptionIfTheFrontIsNull ()
```

shouldConvertFrontToSolutionListReturnAnEmptyListIfTheFrontIsEmpty

```
public void shouldConvertFrontToSolutionListReturnAnEmptyListIfTheFrontIsEmpty ()
```

shouldConvertFrontToSolutionListReturnTheCorrectListCaseA

```
public void shouldConvertFrontToSolutionListReturnTheCorrectListCaseA ()  
    Case A: The front has one point
```

shouldConvertFrontToSolutionListReturnTheCorrectListCaseB

```
public void shouldConvertFrontToSolutionListReturnTheCorrectListCaseB ()  
    Case A: The front has one three points
```

shouldDistanceToClosestPointRaiseAnExceptionIfTheFrontIsEmpty

```
public void shouldDistanceToClosestPointRaiseAnExceptionIfTheFrontIsEmpty ()
```

shouldDistanceToClosestPointRaiseAnExceptionIfTheFrontIsNull

```
public void shouldDistanceToClosestPointRaiseAnExceptionIfTheFrontIsNull ()
```

shouldDistanceToClosestPointRaiseAnExceptionIfThePointIsNull

```
public void shouldDistanceToClosestPointRaiseAnExceptionIfThePointIsNull ()
```

shouldDistanceToClosestPointReturnMaxZeroIfThePointIsTheOnlyPointInTheFront

```
public void shouldDistanceToClosestPointReturnMaxZeroIfThePointIsTheOnlyPointInTheFront ()
```

shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasHasOnePoint

```
public void shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasHasOnePoint ()
```

shouldDistanceToNearestPointClosestTheCorrectValueIfTheFrontHasTwoPointsCaseA

```
public void shouldDistanceToNearestPointClosestTheCorrectValueIfTheFrontHasTwoPointsCaseA ()  
    Case A: the front has two points and one of them is the point passed as a parameter
```

shouldDistanceToNearestPointClosestTheCorrectValueIfTheFrontHasTwoPointsCaseB

```
public void shouldDistanceToNearestPointClosestTheCorrectValueIfTheFrontHasTwoPointsCaseB ()
```

Case B: the front has two points and none of them is the point passed as a parameter. The dimensions of the points are ordered

shouldDistanceToNearestPointClosestTheCorrectValueIfTheFrontHasTwoPointsCaseC

```
public void shouldDistanceToNearestPointClosestTheCorrectValueIfTheFrontHasTwoPointsCaseC ()
```

Case B: the front has two points and none of them is the point passed as a parameter. The dimensions of the points are not ordered

shouldDistanceToNearestPointRaiseAnExceptionIfTheFrontIsEmpty

```
public void shouldDistanceToNearestPointRaiseAnExceptionIfTheFrontIsEmpty ()
```

shouldDistanceToNearestPointRaiseAnExceptionIfTheFrontIsNull

```
public void shouldDistanceToNearestPointRaiseAnExceptionIfTheFrontIsNull ()
```

shouldDistanceToNearestPointRaiseAnExceptionIfThePointIsNull

```
public void shouldDistanceToNearestPointRaiseAnExceptionIfThePointIsNull ()
```

shouldDistanceToNearestPointReturnMaxDoubleIfThePointIsTheOnlyPointInTheFront

```
public void shouldDistanceToNearestPointReturnMaxDoubleIfThePointIsTheOnlyPointInTheFront ()
```

shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasHasOnePoint

```
public void shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasHasOnePoint ()
```

shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasTwoPointsCaseA

```
public void shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasTwoPointsCaseA ()
```

Case A: the front has two points and one of them is the point passed as a parameter

shouldGetInvertedFrontRaiseAnExceptionIfTheFrontIsEmpty

```
public void shouldGetInvertedFrontRaiseAnExceptionIfTheFrontIsEmpty ()
```

shouldGetInvertedFrontRaiseAnExceptionIfTheFrontIsNull

```
public void shouldGetInvertedFrontRaiseAnExceptionIfTheFrontIsNull ()
```

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfFourPoints

```
public void shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfFourPoints ()  
    The front has the points [0.1, 0.9], [0.2, 0.8], [0.3, 0.7], [0.4, 0.6]. The inverted front is [0.9, 0.1], [0.8, 0.2],  
    [0.7, 0.3], [0.6, 0.4]
```

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCaseA

```
public void shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCaseA ()  
    Case A: the front has the point [0.5, 0.5]. The inverted front is the same
```

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCaseB

```
public void shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCaseB ()  
    Case B: the front has the point [0.0, 1.0]. The inverted front is [1.0, 0.0]
```

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCaseC

```
public void shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCaseC ()  
    Case C: the front has the point [3.0, -2.0]. The inverted front is [0.0, 1.0]
```

shouldGetMaximumValuesRaiseAnExceptionIfTheFrontIsEmpty

```
public void shouldGetMaximumValuesRaiseAnExceptionIfTheFrontIsEmpty ()
```

shouldGetMaximumValuesRaiseAnExceptionIfTheFrontIsNull

```
public void shouldGetMaximumValuesRaiseAnExceptionIfTheFrontIsNull ()
```

shouldGetMaximumValuesWithAFrontWithOnePointReturnTheCorrectValue

```
public void shouldGetMaximumValuesWithAFrontWithOnePointReturnTheCorrectValue ()
```

shouldGetMaximumValuesWithAFrontWithThreePointReturnTheCorrectValue

```
public void shouldGetMaximumValuesWithAFrontWithThreePointReturnTheCorrectValue ()
```

shouldGetMinimumValuesRaiseAnExceptionIfTheFrontIsEmpty

```
public void shouldGetMinimumValuesRaiseAnExceptionIfTheFrontIsEmpty ()
```

shouldGetMinimumValuesRaiseAnExceptionIfTheFrontIsNull

```
public void shouldGetMinimumValuesRaiseAnExceptionIfTheFrontIsNull ()
```

shouldGetMinimumValuesWithAFrontWithOnePointReturnTheCorrectValue

```
public void shouldGetMinimumValuesWithAFrontWithOnePointReturnTheCorrectValue ()
```

shouldGetMinimumValuesWithAFrontWithThreePointReturnTheCorrectValue

```
public void shouldGetMinimumValuesWithAFrontWithThreePointReturnTheCorrectValue ()
```

2.103 org.uma.jmetal.util.front.util

2.103.1 FrontNormalizer

```
public class FrontNormalizer
    Class for normalizing Front objects
```

Author Antonio J. Nebro

Constructors

FrontNormalizer

```
public FrontNormalizer (List<? extends Solution<?>> referenceFront)
    Constructor.
```

Parameters

- **referenceFront** –

FrontNormalizer

```
public FrontNormalizer (Front referenceFront)
    Constructor.
```

Parameters

- **referenceFront** –

FrontNormalizer

```
public FrontNormalizer (double[] minimumValues, double[] maximumValues)
    Constructor
```

Parameters

- **minimumValues** –
- **maximumValues** –

Methods

normalize

public [List](#)<? extends [Solution](#)<?>> **normalize** ([List](#)<? extends [Solution](#)<?>> *solutionList*)

Returns a normalized front

Parameters

- **solutionList** –

normalize

public [Front](#) **normalize** ([Front](#) *front*)

Returns a normalized front

Parameters

- **front** –

2.103.2 FrontNormalizerTest

public class **FrontNormalizerTest**

Author Antonio J. Nebro

Fields

exception

public ExpectedException **exception**

Methods

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheReferenceFrontIsNull

public void **shouldFrontNormalizerConstructorRaiseAnExceptionIsTheReferenceFrontIsNull** ()

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheReferenceSolutionListIsNull

public void **shouldFrontNormalizerConstructorRaiseAnExceptionIsTheReferenceSolutionListIsNull** ()

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheVectorOfMaximumValuesIsNull

public void **shouldFrontNormalizerConstructorRaiseAnExceptionIsTheVectorOfMaximumValuesIsNull** ()

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheVectorOfMinimumValuesIsNull

public void **shouldFrontNormalizerConstructorRaiseAnExceptionIsTheVectorOfMinimumValuesIsNull** ()

shouldFrontNormalizerConstructorRaiseAnExceptionTheDimensionOfTheMaximumAndMinimumArraysIsNotEqual

```
public void shouldFrontNormalizerConstructorRaiseAnExceptionTheDimensionOfTheMaximumAndMinimumA
```

shouldGetNormalizedFrontReturnTheCorrectFrontIfTheSolutionListContainsTwoPoints

```
public void shouldGetNormalizedFrontReturnTheCorrectFrontIfTheSolutionListContainsTwoPoints ()
    Points: [2,4], [-2, 3] Maximum values: [6, 8] Minimum values: [-10, 1] Result: [0.5, 1.0], []
```

shouldGetNormalizedFrontReturnTheCorrectFrontIfThisContainsTwoPoints

```
public void shouldGetNormalizedFrontReturnTheCorrectFrontIfThisContainsTwoPoints ()
    Points: [2,4], [-2, 3] Maximum values: [6, 8] Minimum values: [-10, 1] Result: [0.5, 1.0], []
```

shouldNormalizeRaiseAnExceptionIfTheFrontIsEmpty

```
public void shouldNormalizeRaiseAnExceptionIfTheFrontIsEmpty ()
```

shouldNormalizeRaiseAnExceptionIfTheMaxAndMinValuesAreTheSame

```
public void shouldNormalizeRaiseAnExceptionIfTheMaxAndMinValuesAreTheSame ()
    Point: [2,4] Maximum values: [2, 4] Minimum values: [2, 4] Result: [0.5, 1.0]
```

shouldNormalizeRaiseAnExceptionIfTheSolutionListIsEmpty

```
public void shouldNormalizeRaiseAnExceptionIfTheSolutionListIsEmpty ()
```

shouldNormalizeRaiseAnExceptionTheDimensionOfTheMaximumArrayPointsIsNotCorrect

```
public void shouldNormalizeRaiseAnExceptionTheDimensionOfTheMaximumArrayPointsIsNotCorrect ()
```

shouldNormalizeRaiseAnExceptionTheFrontIsNull

```
public void shouldNormalizeRaiseAnExceptionTheFrontIsNull ()
```

shouldNormalizeRaiseAnExceptionTheSolutionListIsNull

```
public void shouldNormalizeRaiseAnExceptionTheSolutionListIsNull ()
```

shouldNormalizeReturnTheCorrectFrontIfThisContainsOnePoint

```
public void shouldNormalizeReturnTheCorrectFrontIfThisContainsOnePoint ()
    Point: [2,4] Maximum values: [4, 4] Minimum values: [0, 0] Result: [0.5, 1.0]
```

2.103.3 FrontUtils

public class **FrontUtils**

A Front is a list of points. This class includes utilities to work with *Front* objects.

Author Antonio J. Nebro

Methods

convertFrontToArray

public static double[][] **convertFrontToArray** (*Front* front)

Given a front, converts it to an array of double values

Parameters

- **front** –

Returns A front as double[][] array

convertFrontToSolutionList

public static List<*PointSolution*> **convertFrontToSolutionList** (*Front* front)

Given a front, converts it to a Solution set of PointSolutions

Parameters

- **front** –

Returns A front as a List

distanceToClosestPoint

public static double **distanceToClosestPoint** (*Point* point, *Front* front)

Gets the distance between a point and the nearest one in a given front. The Euclidean distance is assumed

Parameters

- **point** – The point
- **front** – The front that contains the other points to calculate the distances

Returns The minimum distance between the point and the front

distanceToClosestPoint

public static double **distanceToClosestPoint** (*Point* point, *Front* front, *PointDistance* distance)

Gets the distance between a point and the nearest one in a given front

Parameters

- **point** – The point
- **front** – The front that contains the other points to calculate the distances

Returns The minimum distance between the point and the front

distanceToNearestPoint

public static double **distanceToNearestPoint** (*Point* point, *Front* front)

Gets the distance between a point and the nearest one in a front. If a distance equals to 0 is found, that means that the point is in the front, so it is excluded

Parameters

- **point** – The point
- **front** – The front that contains the other points to calculate the distances

Returns The minimum distance between the point and the front

distanceToNearestPoint

public static double **distanceToNearestPoint** (*Point* point, *Front* front, *PointDistance* distance)

Gets the distance between a point and the nearest one in a front. If a distance equals to 0 is found, that means that the point is in the front, so it is excluded

Parameters

- **point** – The point
- **front** – The front that contains the other points to calculate the distances

Returns The minimum distance between the point and the front

getInvertedFront

public static *Front* **getInvertedFront** (*Front* front)

This method receives a normalized pareto front and return the inverted one. This method is for minimization problems

Parameters

- **front** – The pareto front to inverse

Returns The inverted pareto front

getMaximumValues

public static double[] **getMaximumValues** (*Front* front)

Gets the maximum values for each objectives in a front

Parameters

- **front** – A front of objective values

Returns double [] An array with the maximum values for each objective

getMinimumValues

public static double[] **getMinimumValues** (*Front* front)

Gets the minimum values for each objectives in a given front

Parameters

- **front** – The front

Returns double [] An array with the minimum value for each objective

2.104 org.uma.jmetal.util.naming

2.104.1 DescribedEntity

public interface **DescribedEntity**

A *DescribedEntity* is identified through its name (*getName()*) and further detailed through its description (*getDescription()*).

Author Matthieu Vergne

Methods

getDescription

public *String* **getDescription()**

Returns the description of the *DescribedEntity*

getName

public *String* **getName()**

Returns the name of the *DescribedEntity*

2.105 org.uma.jmetal.util.naming.impl

2.105.1 DescribedEntitySet

public class **DescribedEntitySet**<Entity extends DescribedEntity> implements Set<Entity>

Methods

add

public boolean **add**(Entity *e*)

addAll

public boolean **addAll**(Collection<? extends Entity> *c*)

clear

public void **clear**()

contains

public boolean **contains** (*Object o*)

contains

public boolean **contains** (*String name*)

containsAll

public boolean **containsAll** (*Collection<?> c*)

get

public <E extends Entity> E **get** (*String name*)

isEmpty

public boolean **isEmpty** ()

iterator

public Iterator<Entity> **iterator** ()

remove

public boolean **remove** (*Object o*)

remove

public boolean **remove** (*String name*)

removeAll

public boolean **removeAll** (*Collection<?> c*)

retainAll

public boolean **retainAll** (*Collection<?> c*)

size

public int **size** ()

toArray

```
public Object[] toArray ()
```

toArray

```
public <T> T[] toArray (T[] a)
```

toString

```
public String toString ()
```

2.105.2 DescribedEntitySetTest

```
public class DescribedEntitySetTest
```

Methods**testAddingDifferentEntityWithDifferentNameProperlyAdds**

```
public void testAddingDifferentEntityWithDifferentNameProperlyAdds ()
```

testAddingDifferentEntityWithSameNameThrowsException

```
public void testAddingDifferentEntityWithSameNameThrowsException ()
```

testAddingSameEntityModifiesNothing

```
public void testAddingSameEntityModifiesNothing ()
```

testGetReturnsCorrectEntity

```
public void testGetReturnsCorrectEntity ()
```

testToStringInCaseInsensitiveOrder

```
public void testToStringInCaseInsensitiveOrder ()
```

2.105.3 SimpleDescribedEntity

public class **SimpleDescribedEntity** implements *DescribedEntity*
SimpleDescribedEntity is a basic implementation of *DescribedEntity*. It provides a basic support for the most generic properties required by this interface.

Author Matthieu Vergne

Constructors

SimpleDescribedEntity

public **SimpleDescribedEntity** (*String name*, *String description*)

Create a *SimpleDescribedEntity* with a given name and a given description.

Parameters

- **name** – the name of the *DescribedEntity*
- **description** – the description of the *DescribedEntity*

SimpleDescribedEntity

public **SimpleDescribedEntity** (*String name*)

Create a *SimpleDescribedEntity* with a given name and a null description.

Parameters

- **name** – the name of the *DescribedEntity*

SimpleDescribedEntity

public **SimpleDescribedEntity** ()

Create a *SimpleDescribedEntity* with the class name as its name and a null description.

Methods

getDescription

public *String* **getDescription** ()

getName

public *String* **getName** ()

setDescription

public void **setDescription** (*String description*)

Parameters

- **description** – the new description of this *DescribedEntity*

setName

public void **setName** (*String name*)

Parameters

- **name** – the new name of this *DescribedEntity*

toString

```
public String toString ()
```

2.105.4 SimpleDescribedEntityTest

```
public class SimpleDescribedEntityTest
```

Methods

testClassNameWhenNoName

```
public void testClassNameWhenNoName ()
```

testCorrectDescriptionWhenProvided

```
public void testCorrectDescriptionWhenProvided ()
```

testCorrectNameWhenProvided

```
public void testCorrectNameWhenProvided ()
```

testNullDescriptionWhenNoDescription

```
public void testNullDescriptionWhenNoDescription ()
```

testSetGetDescription

```
public void testSetGetDescription ()
```

testSetName

```
public void testSetName ()
```

2.105.5 SimpleDescribedEntityTest.TestedClass

```
class TestedClass extends SimpleDescribedEntity
```

2.106 org.uma.jmetal.util.neighborhood

2.106.1 Neighborhood

```
public interface Neighborhood<S> extends Serializable
```

Interface representing a neighborhood of a given solution in a list of solutions

Author Antonio J. Nebro

Methods

getNeighbors

```
public List<S> getNeighbors (List<S> solutionList, int solutionIndex)
```

2.107 org.uma.jmetal.util.neighborhood.impl

2.107.1 AdaptiveRandomNeighborhood

public class **AdaptiveRandomNeighborhood**<S> implements *Neighborhood*<S>

This class implements the adaptive random neighborhood (topology) defined by M. Clerc. Each solution in a solution list must have a neighborhood composed by it itself and K random selected neighbors (the same solution can be chosen several times).

Author Antonio J. Nebro

Constructors

AdaptiveRandomNeighborhood

```
public AdaptiveRandomNeighborhood (int solutionListSize, int numberOfRandomNeighbours)
```

Constructor

Parameters

- **solutionListSize** – The expected size of the list of solutions
- **numberOfRandomNeighbours** – The number of neighbors per solution

AdaptiveRandomNeighborhood

```
public AdaptiveRandomNeighborhood (int solutionListSize, int numberOfRandomNeighbours, BoundedRandomGenerator<Integer> randomGenerator)
```

Constructor

Parameters

- **solutionListSize** – The expected size of the list of solutions
- **numberOfRandomNeighbours** – The number of neighbors per solution
- **randomGenerator** – the *BoundedRandomGenerator* to use for the randomisation

Methods

getNeighbors

```
public List<S> getNeighbors (List<S> solutionList, int solutionIndex)
```

recompute

```
public void recompute ()  
    Recomputes the neighbors
```

2.107.2 AdaptiveRandomNeighborhoodTest

```
public class AdaptiveRandomNeighborhoodTest  
    Author Antonio J. Nebro
```

Fields

exception

```
public ExpectedException exception
```

Methods

shouldConstructorCreateAnInstanceIfTheParamtersAreValid

```
public void shouldConstructorCreateAnInstanceIfTheParamtersAreValid ()
```

shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsEqualThanTheListSize

```
public void shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsEqualThanTheListSize ()
```

shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsGreaterThanTheListSize

```
public void shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsGreaterThanTheListSize
```

shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsNegative

```
public void shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsNegative ()
```

shouldGetNeighborsReturnThreeNeighborsPlusTheCurrentSolution

```
public void shouldGetNeighborsReturnThreeNeighborsPlusTheCurrentSolution ()  
    Case 1 Solution list size: 3 Number of neighbors: 1 Neighbors: - solution 0: 0, 2 - solution 1: 1, 0 - solution 2:  
    2, 0
```

shouldGetNeighborsReturnTwoNeighborsPlusTheCurrentSolution

```
public void shouldGetNeighborsReturnTwoNeighborsPlusTheCurrentSolution ()  
    Case 1 Solution list size: 4 Number of neighbors: 2
```

shouldGetNeighborsThrowAnExceptionIfTheListSizelsNotCorrect

```
public void shouldGetNeighborsThrowAnExceptionIfTheListSizeIsNotCorrect ()
```

shouldGetNeighborsWithANegativeSolutionIndexThrowAnException

```
public void shouldGetNeighborsWithANegativeSolutionIndexThrowAnException ()
```

shouldGetNeighborsWithANullListOfSolutionsThrowAnException

```
public void shouldGetNeighborsWithANullListOfSolutionsThrowAnException ()
```

shouldGetNeighborsWithATooBigSolutionIndexThrowAnException

```
public void shouldGetNeighborsWithATooBigSolutionIndexThrowAnException ()
```

shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided

```
public void shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorProvided ()
```

2.107.3 C25

```
public class C25<S extends Solution<?>> extends TwoDimensionalMesh<S>
```

Class defining an C25 neighborhood of a solution belonging to a list of solutions which is structured as a bi-dimensional mesh. The neighbors are those solutions that are in 2-hop distance or less Shape: * * * * *

* * * 0 * * * * *

Author Esteban López Camacho

Constructors**C25**

```
public C25 (int rows, int columns)
```

Constructor. Defines a neighborhood for solutionSetSize

2.107.4 C49

```
public class C49<S extends Solution<?>> extends TwoDimensionalMesh<S>
```

Class defining an C49 neighborhood of a solution belonging to a list of solutions which is structured as a bi-dimensional mesh. The neighbors are those solutions that are in 3-hop distance or less Shape: * * * * *

* * * * * 0 * * * * *

Author Esteban López Camacho

Constructors

C49

public **C49** (int *rows*, int *columns*)
Constructor. Defines a neighborhood for solutionSetSize

2.107.5 C9

public class **C9**<S> extends *TwoDimensionalMesh*<S>
Class defining an L9 neighborhood of a solution belonging to a list of solutions which is structured as a bi-dimensional mesh. The neighbors are those solutions that are in 1-hop distance Shape: * * * * O * * * *
Author Antonio J. Nebro , Juan J. Durillo

Constructors

C9

public **C9** (int *rows*, int *columns*)
Constructor Defines a neighborhood for a solution set of rows x columns solutions

Parameters

- **rows** – the number of rows
- **columns** – the number of columns

2.107.6 C9Test

public class **C9Test**
Created by ajnebro on 26/5/15.

Methods

shouldGetNeighborsReturnFourNeighborsCase1

public void **shouldGetNeighborsReturnFourNeighborsCase1** ()
Case 1 Solution list: 0 The solution location is 0, the neighborhood is 0

shouldGetNeighborsReturnFourNeighborsCase10

public void **shouldGetNeighborsReturnFourNeighborsCase10** ()
Case 10 Solution list: 0 1 2 3 4 5 6 7 8 9 10 11 The solution location is 5, the neighborhood is 0, 1, 2, 4, 6, 8, 9

shouldGetNeighborsReturnFourNeighborsCase11

public void **shouldGetNeighborsReturnFourNeighborsCase11** ()
Case 11 Solution list: 0 1 2 3 4 5 6 7 8 9 10 11 The solution location is 11, the neighborhood is 6, 7, 10, 3, 2, 8

shouldGetNeighborsReturnFourNeighborsCase2

```
public void shouldGetNeighborsReturnFourNeighborsCase2 ()  
    Case 2 Solution list: 0 1 The solution location is 0, the neighborhood is 0, 1
```

shouldGetNeighborsReturnFourNeighborsCase3

```
public void shouldGetNeighborsReturnFourNeighborsCase3 ()  
    Case 3 Solution list: 0 1 The solution location is 1, the neighborhood is 0, 1
```

shouldGetNeighborsReturnFourNeighborsCase4

```
public void shouldGetNeighborsReturnFourNeighborsCase4 ()  
    Case 4 Solution list: 0 1 2 3 The solution location is 0, the neighborhood is 1, 2, 3
```

shouldGetNeighborsReturnFourNeighborsCase5

```
public void shouldGetNeighborsReturnFourNeighborsCase5 ()  
    Case 5 Solution list: 0 1 2 3 The solution location is 1, the neighborhood is 0, 2, 3
```

shouldGetNeighborsReturnFourNeighborsCase6

```
public void shouldGetNeighborsReturnFourNeighborsCase6 ()  
    Case 6 Solution list: 0 1 2 3 The solution location is 2, the neighborhood is 0, 1, 3
```

shouldGetNeighborsReturnFourNeighborsCase7

```
public void shouldGetNeighborsReturnFourNeighborsCase7 ()  
    Case 7 Solution list: 0 1 2 3 The solution location is 3, the neighborhood is 0, 1, 2
```

shouldGetNeighborsReturnFourNeighborsCase8

```
public void shouldGetNeighborsReturnFourNeighborsCase8 ()  
    Case 8 Solution list: 0 1 2 3 4 5 6 7 The solution location is 0, the neighborhood is 1, 4, 5, 3, 7
```

shouldGetNeighborsReturnFourNeighborsCase9

```
public void shouldGetNeighborsReturnFourNeighborsCase9 ()  
    Case 9 Solution list: 0 1 2 3 4 5 6 7 The solution location is 5, the neighborhood is 0, 1, 2, 4, 6
```

2.107.7 KNearestNeighborhood

```
public class KNearestNeighborhood<S extends Solution<?>> implements Neighborhood<S>
```

This class implements a neighborhood that select the k-nearest solutions according to a distance measure. By default, the Euclidean distance between objectives is used.

Parameters

- `<S>` –

Constructors

KNearestNeighborhood

```
public KNearestNeighborhood (int neighborSize)
```

KNearestNeighborhood

```
public KNearestNeighborhood (int neighborSize, Distance<S, S> distance)
```

Methods

getNeighbors

```
public List<S> getNeighbors (List<S> solutionList, int solutionIndex)
```

2.107.8 KNearestNeighborhoodTest

```
public class KNearestNeighborhoodTest
```

Methods

shouldGetNeighborsWorkProperlyCaseA

```
public void shouldGetNeighborsWorkProperlyCaseA ()  
    Case A: The solution list has two solutions and the neighbor size is 1
```

shouldGetNeighborsWorkProperlyCaseB

```
public void shouldGetNeighborsWorkProperlyCaseB ()  
    Case B: The solution list has three solutions, the index of the solution is 0, and the neighbor size is 2
```

shouldGetNeighborsWorkProperlyCaseC

```
public void shouldGetNeighborsWorkProperlyCaseC ()  
    Case C: The solution list has three solutions, the index of the solution is 1, and the neighbor size is 2
```

shouldGetNeighborsWorkProperlyCaseD

```
public void shouldGetNeighborsWorkProperlyCaseD ()  
    Case D: The solution list has three solutions, the index of the solution is 2, and the neighbor size is 2
```

shouldGetNeighborsWorkProperlyCaseE

```
public void shouldGetNeighborsWorkProperlyCaseE ()
```

Case E: The solution list has five solutions, the index of the solution is 0, and the neighbor size is 3

shouldGetNeighborsWorkProperlyCaseF

```
public void shouldGetNeighborsWorkProperlyCaseF ()
```

Case F: The solution list has five solutions, the index of the solution is 2, and the neighbor size is 3

2.107.9 L13

```
public class L13<S extends Solution<?>> extends TwoDimensionalMesh<S>
```

Class defining an L9 neighborhood of a solution belonging to a list of solutions which is structured as a bi-dimensional mesh. The neighbors is illustrated as follows: * * * * * O * * * * *

Author Esteban López Camacho

Constructors**L13**

```
public L13 (int rows, int columns)
```

Constructor. Defines a neighborhood for a solution set of rows x columns solutions

2.107.10 L13Test

```
public class L13Test
```

Created by ajnebro on 20/12/17.

Methods**shouldGetNeighborsReturnFourNeighborsCase1**

```
public void shouldGetNeighborsReturnFourNeighborsCase1 ()
```

Case 1 Solution list: 0 The solution location is 0, the neighborhood is 0

shouldGetNeighborsReturnFourNeighborsCase10

```
public void shouldGetNeighborsReturnFourNeighborsCase10 ()
```

Case 10 Solution list: 0 1 2 3 4 5 6 7 The solution location is 0, the neighborhood is 1, 2, 6

shouldGetNeighborsReturnFourNeighborsCase11

```
public void shouldGetNeighborsReturnFourNeighborsCase11 ()
```

Case 11 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 4, the neighborhood is 1, 3, 5, 7

shouldGetNeighborsReturnFourNeighborsCase12

public void **shouldGetNeighborsReturnFourNeighborsCase12** ()
Case 12 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 8, the neighborhood is 2, 6, 5, 7

shouldGetNeighborsReturnFourNeighborsCase2

public void **shouldGetNeighborsReturnFourNeighborsCase2** ()
Case 2 Solution list: 0 1 The solution location is 0, the neighborhood is 0, 1

shouldGetNeighborsReturnFourNeighborsCase3

public void **shouldGetNeighborsReturnFourNeighborsCase3** ()
Case 3 Solution list: 0 1 The solution location is 1, the neighborhood is 0, 1

shouldGetNeighborsReturnFourNeighborsCase4

public void **shouldGetNeighborsReturnFourNeighborsCase4** ()
Case 4 Solution list: 0 1 2 3 The solution location is 0, the neighborhood is 1, 2

shouldGetNeighborsReturnFourNeighborsCase5

public void **shouldGetNeighborsReturnFourNeighborsCase5** ()
Case 5 Solution list: 0 1 2 3 The solution location is 1, the neighborhood is 0, 3

shouldGetNeighborsReturnFourNeighborsCase6

public void **shouldGetNeighborsReturnFourNeighborsCase6** ()
Case 6 Solution list: 0 1 2 3 The solution location is 2, the neighborhood is 0, 3

shouldGetNeighborsReturnFourNeighborsCase7

public void **shouldGetNeighborsReturnFourNeighborsCase7** ()
Case 7 Solution list: 0 1 2 3 The solution location is 3, the neighborhood is 1, 2

shouldGetNeighborsReturnFourNeighborsCase8

public void **shouldGetNeighborsReturnFourNeighborsCase8** ()
Case 8 Solution list: 0 1 2 3 4 5 6 7 The solution location is 5, the neighborhood is 1, 1, 4, 6

shouldGetNeighborsReturnFourNeighborsCase9

public void **shouldGetNeighborsReturnFourNeighborsCase9** ()
Case 9 Solution list: 0 1 2 3 4 5 6 7 The solution location is 5, the neighborhood is 3, 4, 7

2.107.11 L25

public class **L25**<S extends Solution<?>> extends *TwoDimensionalMesh*<S>
Class representing neighborhoods for a solution into a list of solutions

Author Esteban López Camacho

Constructors

L25

public **L25** (int *rows*, int *columns*)
Constructor. Defines a neighborhood for solutionSetSize

2.107.12 L41

public class **L41**<S extends Solution<?>> extends *TwoDimensionalMesh*<S>
Class representing neighborhoods for a solution into a list of solutions

Author Esteban López Camacho

Constructors

L41

public **L41** (int *rows*, int *columns*)
Constructor. Defines a neighborhood for solutionSetSize

2.107.13 L5

public class **L5**<S> extends *TwoDimensionalMesh*<S>
Class defining an L5 neighborhood of a solution belonging to a list of solutions which is structured as a bi-dimensional mesh. The neighbors are those solutions that are in the positions North, South, East and West
Shape: **** o ****

Author Antonio J. Nebro , Juan J. Durillo

Constructors

L5

public **L5** (int *rows*, int *columns*)
Constructor. Defines a neighborhood for a solution set of rows x columns solutions

2.107.14 L5Test

public class **L5Test**
Created by ajnebro on 26/5/15.

Methods

shouldGetNeighborsReturnFourNeighborsCase1

public void **shouldGetNeighborsReturnFourNeighborsCase1** ()
Case 1 Solution list: 0 The solution location is 0, the neighborhood is 0

shouldGetNeighborsReturnFourNeighborsCase2

public void **shouldGetNeighborsReturnFourNeighborsCase2** ()
Case 2 Solution list: 0 1 The solution location is 0, the neighborhood is 0, 1

shouldGetNeighborsReturnFourNeighborsCase3

public void **shouldGetNeighborsReturnFourNeighborsCase3** ()
Case 3 Solution list: 0 1 The solution location is 1, the neighborhood is 0, 1

shouldGetNeighborsReturnFourNeighborsCase4

public void **shouldGetNeighborsReturnFourNeighborsCase4** ()
Case 4 Solution list: 0 1 2 3 The solution location is 0, the neighborhood is 1, 2

2.107.15 WeightVectorNeighborhood

public class **WeightVectorNeighborhood**<S> implements *Neighborhood*<S>
This class implements a neighborhood based on the weight vectors of MOEA/D

Author Antonio J. Nebro

Constructors

WeightVectorNeighborhood

public **WeightVectorNeighborhood** (int *numberOfWeightVectors*, int *neighborSize*)

WeightVectorNeighborhood

public **WeightVectorNeighborhood** (int *numberOfWeightVectors*, int *weightVectorSize*, int *neighborSize*,
String *vectorFileName*)

Methods

getNeighborSize

public int **getNeighborSize** ()

getNeighborhood

```
public int[][] getNeighborhood()
```

getNeighbors

```
public List<S> getNeighbors(List<S> solutionList, int solutionIndex)
```

getNumberOfWeightVectors

```
public int getNumberOfWeightVectors()
```

getWeightVector

```
public double[][] getWeightVector()
```

getWeightVectorSize

```
public int getWeightVectorSize()
```

2.107.16 WeightVectorNeighborhoodTest

```
public class WeightVectorNeighborhoodTest
```

Methods**shouldConstructorRaiseAnExceptionIfTheWeightFileDoesNotExist**

```
public void shouldConstructorRaiseAnExceptionIfTheWeightFileDoesNotExist()
```

shouldDefaultConstructorBeCorrectlyInitialized

```
public void shouldDefaultConstructorBeCorrectlyInitialized()
```

shouldGetNeighborsWorksProperlyWithTwoObjectives

```
public void shouldGetNeighborsWorksProperlyWithTwoObjectives()
```

2.108 org.uma.jmetal.util.neighborhood.util**2.108.1 TwoDimensionalMesh**

```
public class TwoDimensionalMesh<S> implements Neighborhood<S>
    Class defining a bi-dimensional mesh.
```

Constructors

TwoDimensionalMesh

public **TwoDimensionalMesh** (int *rows*, int *columns*, int[][] *neighborhood*)
Constructor. Defines a neighborhood for list of solutions

Methods

getNeighbors

public [List<S>](#) **getNeighbors** ([List<S>](#) *solutionList*, int *solutionPosition*)
Returns the north,south, east, and west solutions of a given solution

Parameters

- **solutionList** – the solution set from where the neighbors are taken
- **solutionPosition** – Represents the position of the solution

2.108.2 TwoDimensionalMeshTest

public class **TwoDimensionalMeshTest**
Created by ajnebro on 21/5/15.

Fields

exception

public ExpectedException **exception**

Methods

shouldGetNeighborsReturnFourNeighborsCase1

public void **shouldGetNeighborsReturnFourNeighborsCase1** ()
Case 1 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 4, the neighborhood is 1, 3, 5, 7

shouldGetNeighborsReturnFourNeighborsCase2

public void **shouldGetNeighborsReturnFourNeighborsCase2** ()
Case 2 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 1, the neighborhood is 7, 0, 2, 4

shouldGetNeighborsReturnFourNeighborsCase3

public void **shouldGetNeighborsReturnFourNeighborsCase3** ()
Case 3 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 0, the neighborhood is 1, 2, 3, 6

shouldGetNeighborsReturnFourNeighborsCase4

```
public void shouldGetNeighborsReturnFourNeighborsCase4 ()  
    Case 4 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 2, the neighborhood is 1, 0, 5, 8
```

shouldGetNeighborsReturnFourNeighborsCase5

```
public void shouldGetNeighborsReturnFourNeighborsCase5 ()  
    Case 5 Solution list: 0 1 2 3 4 5 6 7 8 The solution location is 2, the neighborhood is 2, 6, 7, 5
```

shouldGetNeighborsReturnFourNeighborsCase6

```
public void shouldGetNeighborsReturnFourNeighborsCase6 ()  
    Case 6 Solution list: 0 1 2 3 4 5 The solution location is 0, the neighborhood is 1, 3, 3, 2
```

shouldGetNeighborsReturnFourNeighborsCase7

```
public void shouldGetNeighborsReturnFourNeighborsCase7 ()  
    Case 7 Solution list: 0 1 2 3 4 5 The solution location is 3, the neighborhood is 0, 4, 5, 0
```

shouldGetNeighborsReturnFourNeighborsCase8

```
public void shouldGetNeighborsReturnFourNeighborsCase8 ()  
    Case 8 Solution list: 0 1 2 3 The solution location is 0, the neighborhood is 2, 1, 2, 1
```

shouldGetNeighborsWithANegativeSolutionIndexThrowAnException

```
public void shouldGetNeighborsWithANegativeSolutionIndexThrowAnException ()
```

shouldGetNeighborsWithANullListOfSolutionsThrowAnException

```
public void shouldGetNeighborsWithANullListOfSolutionsThrowAnException ()
```

shouldGetNeighborsWithASolutionIndexValueEqualToTheListSizeThrowAnException

```
public void shouldGetNeighborsWithASolutionIndexValueEqualToTheListSizeThrowAnException ()
```

shouldGetNeighborsWithASolutionIndexValueGreaterThanTheListSizeThrowAnException

```
public void shouldGetNeighborsWithASolutionIndexValueGreaterThanTheListSizeThrowAnException ()
```

shouldGetNeighborsWithAnEmptyListOfSolutionsThrowAnException

```
public void shouldGetNeighborsWithAnEmptyListOfSolutionsThrowAnException ()
```

2.109 org.uma.jmetal.util.point

2.109.1 Point

public interface **Point**

Interface representing a point

Author Antonio J. Nebro

Methods

getDimension

int **getDimension** ()

getValue

double **getValue** (int *index*)

getValues

double[] **getValues** ()

setValue

void **setValue** (int *index*, double *value*)

update

void **update** (double[] *point*)

2.109.2 PointSolution

public class **PointSolution** implements *Solution*<Double>

Solution used to wrap a *Point* object. Only objectives are used.

Author Antonio J. Nebro

Fields

attributes

protected *Map*<*Object*, *Object*> **attributes**

Constructors

PointSolution

public **PointSolution** (int *numberOfObjectives*)
Constructor

Parameters

- **numberOfObjectives** –

PointSolution

public **PointSolution** (*Point* *point*)
Constructor

Parameters

- **point** –

PointSolution

public **PointSolution** (*Solution*<?> *solution*)
Constructor

Parameters

- **solution** –

PointSolution

public **PointSolution** (*PointSolution* *point*)
Copy constructor

Parameters

- **point** –

Methods

copy

public *PointSolution* **copy** ()

equals

public boolean **equals** (Object *o*)

getAttribute

public Object **getAttribute** (Object *id*)

getNumberOfObjectives

```
public int getNumberOfObjectives ()
```

getNumberOfVariables

```
public int getNumberOfVariables ()
```

getObjective

```
public double getObjective (int index)
```

getObjectives

```
public double[] getObjectives ()
```

getVariableValue

```
public Double getVariableValue (int index)
```

getVariableValueString

```
public String getVariableValueString (int index)
```

hashCode

```
public int hashCode ()
```

setAttribute

```
public void setAttribute (Object id, Object value)
```

setObjective

```
public void setObjective (int index, double value)
```

setVariableValue

```
public void setVariableValue (int index, Double value)
```

toString

```
public String toString ()
```


2.110 org.uma.jmetal.util.point.impl

2.110.1 ArrayPoint

public class **ArrayPoint** implements *Point*

Class representing a point (i.e, an array of double values)

Author Antonio J. Nebro

Fields

point

protected double[] **point**

Constructors

ArrayPoint

public **ArrayPoint** ()

Default constructor

ArrayPoint

public **ArrayPoint** (int *dimension*)

Constructor

Parameters

- **dimension** – Dimension of the point

ArrayPoint

public **ArrayPoint** (*Point* *point*)

Copy constructor

Parameters

- **point** –

ArrayPoint

public **ArrayPoint** (double[] *point*)

Constructor from an array of double values

Parameters

- **point** –

ArrayPoint

public **ArrayPoint** (*String fileName*)
Constructor reading the values from a file

Parameters

- **fileName** –

Methods

equals

public boolean **equals** (*Object o*)

getDimension

public int **getDimension** ()

getValue

public double **getValue** (int *index*)

getValues

public double[] **getValues** ()

hashCode

public int **hashCode** ()

setValue

public void **setValue** (int *index*, double *value*)

toString

public *String* **toString** ()

update

public void **update** (double[] *point*)

2.110.2 ArrayPointTest

public class **ArrayPointTest**

Author Antonio J. Nebro

Methods

shouldConstructAPointFromANullPointRaiseAnException

public void **shouldConstructAPointFromANullPointRaiseAnException** ()

shouldConstructAPointFromOtherPointReturnAnIdenticalPoint

public void **shouldConstructAPointFromOtherPointReturnAnIdenticalPoint** ()

shouldConstructAPointOfAGivenDimension

public void **shouldConstructAPointOfAGivenDimension** ()

shouldConstructFromASolutionReturnTheCorrectPoint

public void **shouldConstructFromASolutionReturnTheCorrectPoint** ()

shouldConstructFromArrayReturnTheCorrectPoint

public void **shouldConstructFromArrayReturnTheCorrectPoint** ()

shouldConstructFromNullArrayRaiseAnException

public void **shouldConstructFromNullArrayRaiseAnException** ()

shouldEqualsReturnFalseIfTheClassIsNotAPoint

public void **shouldEqualsReturnFalseIfTheClassIsNotAPoint** ()

shouldEqualsReturnFalseIfThePointIsNull

public void **shouldEqualsReturnFalseIfThePointIsNull** ()

shouldEqualsReturnFalseIfThePointsAreNotIdentical

public void **shouldEqualsReturnFalseIfThePointsAreNotIdentical** ()

shouldEqualsReturnTrueIfThePointsAreIdentical

```
public void shouldEqualsReturnTrueIfThePointsAreIdentical()
```

shouldEqualsReturnTrueIfTheTwoPointsAreTheSame

```
public void shouldEqualsReturnTrueIfTheTwoPointsAreTheSame()
```

shouldGetDimensionValueReturnTheCorrectValue

```
public void shouldGetDimensionValueReturnTheCorrectValue()
```

shouldGetDimensionValueWithInvalidIndexesRaiseAnException

```
public void shouldGetDimensionValueWithInvalidIndexesRaiseAnException()
```

shouldGetNumberOfDimensionsReturnTheCorrectValue

```
public void shouldGetNumberOfDimensionsReturnTheCorrectValue()
```

shouldGetValuesReturnTheCorrectValues

```
public void shouldGetValuesReturnTheCorrectValues()
```

shouldHashCodeReturnTheCorrectValue

```
public void shouldHashCodeReturnTheCorrectValue()
```

shouldSetDimensionValueAssignTheCorrectValue

```
public void shouldSetDimensionValueAssignTheCorrectValue()
```

shouldSetDimensionValueWithInvalidIndexesRaiseAnException

```
public void shouldSetDimensionValueWithInvalidIndexesRaiseAnException()
```

2.110.3 IdealPoint

```
public class IdealPoint extends ArrayPoint  
    d Class representing an ideal point (minimization is assumed)
```

Author Antonio J.Nebro

Constructors

IdealPoint

```
public IdealPoint (int dimension)
```

Methods

update

```
public void update (double[] point)
```

update

```
public void update (List<? extends Solution>> solutionList)
```

2.110.4 IdealPointTest

```
public class IdealPointTest  
    Created by ajnebro on 12/2/16.
```

Methods

shouldConstructorCreateAnIdealPointWithAllObjectiveValuesCorrectlyInitialized

```
public void shouldConstructorCreateAnIdealPointWithAllObjectiveValuesCorrectlyInitialized ()
```

shouldUpdateWithOneSolutionMakeTheIdealPointHaveTheSolutionValues

```
public void shouldUpdateWithOneSolutionMakeTheIdealPointHaveTheSolutionValues ()
```

shouldUpdateWithThreeSolutionsLeadToTheCorrectIdealPoint

```
public void shouldUpdateWithThreeSolutionsLeadToTheCorrectIdealPoint ()
```

shouldUpdateWithTwoSolutionsLeadToTheCorrectIdealPoint

```
public void shouldUpdateWithTwoSolutionsLeadToTheCorrectIdealPoint ()
```

2.110.5 LexicographicalPointComparatorTest

```
public class LexicographicalPointComparatorTest  
    Author Antonio J. Nebro
```

Methods

shouldCompareDifferentLengthPointsReturnTheCorrectValue

```
public void shouldCompareDifferentLengthPointsReturnTheCorrectValue ()
```

shouldCompareEmptyPointsReturnZero

```
public void shouldCompareEmptyPointsReturnZero ()
```

shouldCompareIdenticalPointsButTheFirstValueReturnMinus1

```
public void shouldCompareIdenticalPointsButTheFirstValueReturnMinus1 ()
```

shouldCompareIdenticalPointsButTheFirstValueReturnPlus1

```
public void shouldCompareIdenticalPointsButTheFirstValueReturnPlus1 ()
```

shouldCompareIdenticalPointsButTheLastValueReturnMinus1

```
public void shouldCompareIdenticalPointsButTheLastValueReturnMinus1 ()
```

shouldCompareIdenticalPointsButTheLastValueReturnPlus1

```
public void shouldCompareIdenticalPointsButTheLastValueReturnPlus1 ()
```

shouldCompareIdenticalPointsReturnZero

```
public void shouldCompareIdenticalPointsReturnZero ()
```

shouldFirstPointToCompareEqualsToNullRaiseAnException

```
public void shouldFirstPointToCompareEqualsToNullRaiseAnException ()
```

shouldSecondPointToCompareEqualsToNullRaiseAnException

```
public void shouldSecondPointToCompareEqualsToNullRaiseAnException ()
```

startup

```
public void startup ()
```

2.110.6 NadirPoint

public class **NadirPoint** extends *ArrayPoint*
 Class representing a nadir point (minimization is assumed)

Author Antonio J.Nebro

Constructors

NadirPoint

public **NadirPoint** (int *dimension*)

Methods

update

public void **update** (double[] *point*)

update

public void **update** (List<? extends *Solution*>> *solutionList*)

2.110.7 NadirPointTest

public class **NadirPointTest**
 Created by ajnebro on 12/2/16.

Methods

shouldConstructorCreateANadirPointWithAllObjectiveValuesCorrectlyInitialized

public void **shouldConstructorCreateANadirPointWithAllObjectiveValuesCorrectlyInitialized** ()

shouldUpdateAListOfSolutionsLeadToTheCorrectNadirPoint

public void **shouldUpdateAListOfSolutionsLeadToTheCorrectNadirPoint** ()

shouldUpdateWithOneSolutionMakeTheNadirPointHaveTheSolutionValues

public void **shouldUpdateWithOneSolutionMakeTheNadirPointHaveTheSolutionValues** ()

shouldUpdateWithThreeSolutionsLeadToTheCorrectNadirPoint

public void **shouldUpdateWithThreeSolutionsLeadToTheCorrectNadirPoint** ()

shouldUpdateWithTwoSolutionsLeadToTheCorrectNadirPoint

```
public void shouldUpdateWithTwoSolutionsLeadToTheCorrectNadirPoint ()
```

2.110.8 PointComparatorTest

```
public class PointComparatorTest
```

Author Antonio J. Nebro

Methods

shouldCompareBetterReturnZeroIfBothPointsAreEqualWhenMaximizing

```
public void shouldCompareBetterReturnZeroIfBothPointsAreEqualWhenMaximizing ()
```

shouldCompareBetterReturnZeroIfBothPointsAreEqualWhenMinimizing

```
public void shouldCompareBetterReturnZeroIfBothPointsAreEqualWhenMinimizing ()
```

shouldCompareReturnMinusOneIfTheFirstPointIsBetterThanTheSecondOneWhenMaximizing

```
public void shouldCompareReturnMinusOneIfTheFirstPointIsBetterThanTheSecondOneWhenMaximizing ()
```

shouldCompareReturnOneIfTheSecondPointIsBetterThanTheFirstOneWhenMaximizing

```
public void shouldCompareReturnOneIfTheSecondPointIsBetterThanTheFirstOneWhenMaximizing ()
```

shouldComparingDifferentLengthPointsRaiseAnException

```
public void shouldComparingDifferentLengthPointsRaiseAnException ()
```

shouldFirstPointToCompareEqualsToNullRaiseAnException

```
public void shouldFirstPointToCompareEqualsToNullRaiseAnException ()
```

shouldSecondPointToCompareEqualsToNullRaiseAnException

```
public void shouldSecondPointToCompareEqualsToNullRaiseAnException ()
```

2.110.9 PointDimensionComparatorTest

```
public class PointDimensionComparatorTest
```

Author Antonio J. Nebro

Methods

clean

public void **clean** ()

setup

public void **setup** ()

shouldCompareReturnMinusOnelfTheFirstValuelsLower

public void **shouldCompareReturnMinusOneIfTheFirstValueIsLower** ()

shouldCompareReturnPlusOnelfTheFirstValuelsGreater

public void **shouldCompareReturnPlusOneIfTheFirstValueIsGreater** ()

shouldCompareReturnZeroIfTheComparedValuesAreEqual

public void **shouldCompareReturnZeroIfTheComparedValuesAreEqual** ()

shouldFirstPointToCompareEqualsToNullRaiseAnException

public void **shouldFirstPointToCompareEqualsToNullRaiseAnException** ()

shouldIndexLessThanZeroRaiseAnException

public void **shouldIndexLessThanZeroRaiseAnException** ()

shouldIndexValueGreaterThanFirstPointDimensionsRaiseAnException

public void **shouldIndexValueGreaterThanFirstPointDimensionsRaiseAnException** ()

shouldIndexValueGreaterThanSecondPointDimensionsRaiseAnException

public void **shouldIndexValueGreaterThanSecondPointDimensionsRaiseAnException** ()

shouldSecondPointToCompareEqualsToNullRaiseAnException

public void **shouldSecondPointToCompareEqualsToNullRaiseAnException** ()

2.110.10 PointSolutionTest

public class **PointSolutionTest**

Author Antonio J. Nebro

Methods

idleTestToCoverTheUnusedMethods

public void **idleTestToCoverTheUnusedMethods** ()

shouldCopyConstructorCreateAnIdenticalObject

public void **shouldCopyConstructorCreateAnIdenticalObject** ()

shouldCopyReturnACopyOfTheSolution

public void **shouldCopyReturnACopyOfTheSolution** ()

shouldDefaultConstructorCreateTheObjectCorrectly

public void **shouldDefaultConstructorCreateTheObjectCorrectly** ()

shouldEqualsReturnFalseIfTheClassIsNotAPoint

public void **shouldEqualsReturnFalseIfTheClassIsNotAPoint** ()

shouldEqualsReturnFalseIfThePointsAreNotIdentical

public void **shouldEqualsReturnFalseIfThePointsAreNotIdentical** ()

shouldEqualsReturnFalseIfTheSolutionIsNull

public void **shouldEqualsReturnFalseIfTheSolutionIsNull** ()

shouldEqualsReturnFalseIfTheTwoSolutionsHaveDifferentNumberOfObjectives

public void **shouldEqualsReturnFalseIfTheTwoSolutionsHaveDifferentNumberOfObjectives** ()

shouldEqualsReturnTrueIfTheSolutionsAreIdentical

public void **shouldEqualsReturnTrueIfTheSolutionsAreIdentical** ()

shouldEqualsReturnTrueIfTheTwoPointsAreTheSame

```
public void shouldEqualsReturnTrueIfTheTwoPointsAreTheSame ()
```

shouldGetNumberOfObjectivesReturnTheCorrectValue

```
public void shouldGetNumberOfObjectivesReturnTheCorrectValue ()
```

shouldGetObjectiveReturnTheCorrectValue

```
public void shouldGetObjectiveReturnTheCorrectValue ()
```

shouldHashCodeReturnTheCorrectValue

```
public void shouldHashCodeReturnTheCorrectValue ()
```

shouldSetObjectiveAssignTheTheCorrectValue

```
public void shouldSetObjectiveAssignTheTheCorrectValue ()
```

2.111 org.uma.jmetal.util.point.util

2.111.1 DominanceDistanceTest

```
public class DominanceDistanceTest
```

Author Antonio J. Nebro

Methods**setup**

```
public void setup ()
```

shouldCalculatingDistanceOfPointsWithOneDimensionReturnTheCorrectValue

```
public void shouldCalculatingDistanceOfPointsWithOneDimensionReturnTheCorrectValue ()
```

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseA

```
public void shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseA ()
```

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseB

```
public void shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseB ()
```

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseC

```
public void shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseC ()
```

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseD

```
public void shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseD ()
```

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseE

```
public void shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseE ()
```

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseF

```
public void shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseF ()
```

shouldCalculatingDistanceOfPointsWithZeroDimensionReturnZero

```
public void shouldCalculatingDistanceOfPointsWithZeroDimensionReturnZero ()
```

shouldFirstPointToCompareEqualsToNullRaiseAnException

```
public void shouldFirstPointToCompareEqualsToNullRaiseAnException ()
```

shouldPassingPointsWithDifferentDimensionsRaiseAnException

```
public void shouldPassingPointsWithDifferentDimensionsRaiseAnException ()
```

shouldSecondPointToCompareEqualsToNullRaiseAnException

```
public void shouldSecondPointToCompareEqualsToNullRaiseAnException ()
```

2.111.2 EuclideanDistanceTest

```
public class EuclideanDistanceTest
```

Author Antonio J. Nebro

Methods

setup

public void **setup** ()

shouldCalculatingDistanceOfPointsWithOneDimensionReturnTheCorrectValue

public void **shouldCalculatingDistanceOfPointsWithOneDimensionReturnTheCorrectValue** ()

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseA

public void **shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseA** ()

shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseB

public void **shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValueCaseB** ()

shouldCalculatingDistanceOfPointsWithZeroDimensionReturnZero

public void **shouldCalculatingDistanceOfPointsWithZeroDimensionReturnZero** ()

shouldFirstPointToCompareEqualsToNullRaiseAnException

public void **shouldFirstPointToCompareEqualsToNullRaiseAnException** ()

shouldPassingPointsWithDifferentDimensionsRaiseAnException

public void **shouldPassingPointsWithDifferentDimensionsRaiseAnException** ()

shouldSecondPointToCompareEqualsToNullRaiseAnException

public void **shouldSecondPointToCompareEqualsToNullRaiseAnException** ()

2.112 org.uma.jmetal.util.point.util.comparator

2.112.1 LexicographicalPointComparator

public class **LexicographicalPointComparator** implements [Comparator<Point>](#)

This class implements the Comparator interface for comparing tow points. The order used is lexicographical order.

Author Antonio J. Nebro , Juan J. Durillo

Methods

compare

```
public int compare (Point pointOne, Point pointTwo)
```

The compare method compare the objects o1 and o2.

Parameters

- **pointOne** – An object that reference a double[]
- **pointTwo** – An object that reference a double[]

Returns The following value: -1 if point1 < point2, 1 if point1 > point2 or 0 in other case.

2.112.2 PointComparator

```
public class PointComparator implements Comparator<Point>
```

Point comparator. Starts the comparison from front last point dimension to the first one

Author Antonio J. Nebro

Constructors

PointComparator

```
public PointComparator ()
```

Methods

compare

```
public int compare (Point pointOne, Point pointTwo)
```

Compares two Point objects

Parameters

- **pointOne** – An object that reference a Point
- **pointTwo** – An object that reference a Point

Returns -1 if o1 < o1, 1 if o1 > o2 or 0 in other case.

setMaximizing

```
public void setMaximizing ()
```

setMinimizing

```
public void setMinimizing ()
```

2.112.3 PointDimensionComparator

public class **PointDimensionComparator** implements *Comparator<Point>*

This class implements the *Comparator* interface. It is used to compare two points according the value of a particular dimension.

Author Antonio J. Nebro , Juan J. Durillo

Constructors

PointDimensionComparator

public **PointDimensionComparator** (int *index*)
Constructor

Methods

compare

public int **compare** (*Point pointOne*, *Point pointTwo*)
Compares the objects o1 and o2.

Parameters

- **pointOne** – An object that reference a double[]
- **pointTwo** – An object that reference a double[]

Returns -1 if o1 < o2, 1 if o1 > o2 or 0 in other case.

2.113 org.uma.jmetal.util.point.util.distance

2.113.1 DominanceDistance

public class **DominanceDistance** implements *PointDistance*

Computes the distance between two points a y b according to the dominance relationship. Point a is supposed to be point of the Pareto front

Author Antonio J. Nebro

Methods

compute

public double **compute** (*Point a*, *Point b*)

2.113.2 EuclideanDistance

public class **EuclideanDistance** implements *PointDistance*

Computes the Euclidean distance between two points

Author Antonio J. Nebro

Methods

compute

public double **compute** (*Point a*, *Point b*)

2.113.3 PointDistance

public interface **PointDistance**

Interface representing classes for computing a distance between two points

Author Antonio J. Nebro

Methods

compute

public double **compute** (*Point pointA*, *Point pointB*)

2.114 org.uma.jmetal.util.pseudorandom

2.114.1 BoundedRandomGenerator

public interface **BoundedRandomGenerator**<Value> extends Comparable<Value>>

A *BoundedRandomGenerator* aims to provide a random value within a specific range. The range is inclusive, such that the lower bound and upper bound can be generated. Because lower and upper bounds make no sense if values cannot be compared, only *Comparable* values can be generated through this kind of generator. A *BoundedRandomGenerator* is a *FunctionalInterface*. It is not intended to be directly implemented by a class, but instead to request a method for generating random values, usually by using lambda expressions.

Author Matthieu Vergne

Parameters

- **<Value>** – The type of value to generate

Methods

bound

static *BoundedRandomGenerator*<*Double*> **bound** (*RandomGenerator*<*Double*> *unboundedGenerator*)

Create a *BoundedRandomGenerator* from a *RandomGenerator* which generate *Double* values between 0 and 1 (inclusive or exclusive). The distribution is preserved.

Parameters

- **unboundedGenerator** – *RandomGenerator* which generates values between 0 and 1

Returns *BoundedRandomGenerator* which generates *Double* values based on the provided generator

fromDoubleToInteger

static *BoundedRandomGenerator*<Integer> **fromDoubleToInteger** (*BoundedRandomGenerator*<Double> *doubleGenerator*)

Create a *BoundedRandomGenerator* which generates *Integer* values from a *BoundedRandomGenerator* which generate *Double* values. The distribution is preserved.

Parameters

- **doubleGenerator** – *BoundedRandomGenerator* which generates *Double* values

Returns *BoundedRandomGenerator* which generates *Integer* values based on the provided generator

fromDoubleToInteger

static *BoundedRandomGenerator*<Integer> **fromDoubleToInteger** (*RandomGenerator*<Double> *doubleGenerator*)

Create a *BoundedRandomGenerator* which generates *Integer* values from a *BoundedRandomGenerator* which generate *Double* values between 0 and 1 (inclusive or exclusive). The distribution is preserved.

Parameters

- **doubleGenerator** – *RandomGenerator* which generates *Double* values

Returns *BoundedRandomGenerator* which generates *Integer* values based on the provided generator

getRandomValue

Value **getRandomValue** (Value *lowerBound*, Value *upperBound*)

Generate a random value within the provided range.

Parameters

- **lowerBound** – the minimal value which can be generated
- **upperBound** – the maximal value which can be generated

Returns the value generated

2.114.2 BoundedRandomGeneratorTest

public class **BoundedRandomGeneratorTest**

Methods

testBoundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectDistribution

public void **testBoundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectDistribution** ()

testBoundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectValues

```
public void testBoundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectValues ()
```

testBoundingFactoryMethodReturnsGeneratorWithCorrectValues

```
public void testBoundingFactoryMethodReturnsGeneratorWithCorrectValues ()
```

testUnboundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectDistribution

```
public void testUnboundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectDistribution ()
```

testUnboundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectValues

```
public void testUnboundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectValues ()
```

2.114.3 JMetalRandom

```
public class JMetalRandom implements Serializable
```

Author Antonio J. Nebro

Methods

getGeneratorName

```
public String getGeneratorName ()
```

getInstance

```
public static JMetalRandom getInstance ()
```

getRandomGenerator

```
public PseudoRandomGenerator getRandomGenerator ()
```

getSeed

```
public long getSeed ()
```

nextDouble

```
public double nextDouble ()
```

nextDouble

public double **nextDouble** (double *lowerBound*, double *upperBound*)

nextInt

public int **nextInt** (int *lowerBound*, int *upperBound*)

setRandomGenerator

public void **setRandomGenerator** (*PseudoRandomGenerator* *randomGenerator*)

setSeed

public void **setSeed** (long *seed*)

2.114.4 PseudoRandomGenerator

public interface **PseudoRandomGenerator** extends *Serializable*

Author Antonio J. Nebro

Methods**getName**

public *String* **getName** ()

getSeed

public long **getSeed** ()

nextDouble

public double **nextDouble** (double *lowerBound*, double *upperBound*)

nextDouble

public double **nextDouble** ()

nextInt

public int **nextInt** (int *lowerBound*, int *upperBound*)

setSeed

public void **setSeed** (long *seed*)

2.114.5 RandomGenerator

public interface **RandomGenerator**<Value>

A *RandomGenerator* aims to provide a random value of a given type. Any value of this type can be generated. A *RandomGenerator* is a `FunctionalInterface`. It is not intended to be directly implemented by a class, but instead to request a method for generating random values, usually by using lambda expressions.

Author Matthieu Vergne

Parameters

- **<Value>** – The type of value to generate

Methods

filter

static <T> *RandomGenerator*<T> **filter** (*RandomGenerator*<T> *generator*, *Predicate*<T> *filter*)

Reduce a *RandomGenerator* range. The returned *RandomGenerator* uses the provided one to generate random values, but regenerate them if they do not pass the filter. Consequently, the initial *RandomGenerator* may be called several times to generate a single value. The impact on performance depends on the part of the distribution which is filtered out: if a significant part of the distribution is rejected, it might be more interesting to create a dedicated *RandomGenerator*.

Parameters

- **generator** – the *RandomGenerator* to filter
- **filter** – the filter to pass to be an acceptable value

Returns a *RandomGenerator* which provides only acceptable values

forArray

static <T> *RandomGenerator*<T> **forArray** (*BoundedRandomGenerator*<Integer> *indexSelector*, T... *values*)

Create a *RandomGenerator* over an array based on a random selector.

Parameters

- **indexSelector** – the random selector
- **values** – the values to return

Returns a *RandomGenerator* on the provided values

forCollection

static <T> *RandomGenerator*<T> **forCollection** (*BoundedRandomGenerator*<Integer> *indexSelector*, *Collection*<T> *values*)

Create a *RandomGenerator* over a *Collection* based on a random selector.

Parameters

- **indexSelector** – the random selector
- **values** – the values to return

Returns a *RandomGenerator* on the provided values

forEnum

static <T extends Enum<T>> *RandomGenerator*<T> **forEnum** (*BoundedRandomGenerator*<Integer> *indexSelector*, *Class*<T> *enumClass*) *in-*

Create a *RandomGenerator* over *Enum* values based on a random selector.

Parameters

- **indexSelector** – the random selector
- **enumClass** – the *Enum* to cover

Returns a *RandomGenerator* on the *Enum* values

getRandomValue

public Value **getRandomValue** ()

Generate a random value.

Returns the value generated

2.114.6 RandomGeneratorTest

public class **RandomGeneratorTest**

Methods**testArrayGeneratorGeneratesAllValues**

public void **testArrayGeneratorGeneratesAllValues** ()

testCollectionGeneratorGeneratesAllValues

public void **testCollectionGeneratorGeneratesAllValues** ()

testEnumGeneratorGeneratesAllValues

public void **testEnumGeneratorGeneratesAllValues** ()

testFilteredGeneratorGeneratesCorrectValues

public void **testFilteredGeneratorGeneratesCorrectValues** ()

2.114.7 RandomGeneratorTest.EnumValues

enum **EnumValues**

Enum Constants

VAL1

public static final *RandomGeneratorTest.EnumValues* **VAL1**

VAL2

public static final *RandomGeneratorTest.EnumValues* **VAL2**

VAL3

public static final *RandomGeneratorTest.EnumValues* **VAL3**

2.115 org.uma.jmetal.util.pseudorandom.impl

2.115.1 AuditableRandomGenerator

public class **AuditableRandomGenerator** implements *PseudoRandomGenerator*

An *AuditableRandomGenerator* is a *PseudoRandomGenerator* which can be audited to know when a random generation method is called.

Author Matthieu Vergne

Constructors

AuditableRandomGenerator

public **AuditableRandomGenerator** (*PseudoRandomGenerator* generator)

Methods

addListener

public void **addListener** (*Consumer*<*Audit*> listener)

getName

public *String* **getName** ()

getSeed

```
public long getSeed ()
```

nextDouble

```
public double nextDouble (double lowerBound, double upperBound)
```

nextDouble

```
public double nextDouble ()
```

nextInt

```
public int nextInt (int lowerBound, int upperBound)
```

removeListener

```
public void removeListener (Consumer<Audit> listener)
```

setSeed

```
public void setSeed (long seed)
```

2.115.2 AuditableRandomGenerator.Audit

```
public static class Audit
```

Constructors

Audit

```
public Audit (RandomMethod method, Bounds bounds, Number result)
```

Methods

getBounds

```
public Optional<Bounds> getBounds ()
```

getMethod

```
public RandomMethod getMethod ()
```

getResult

```
public Number getResult ()
```

2.115.3 AuditableRandomGenerator.Bounds

```
public static class Bounds
```

Fields

lower

```
final Number lower
```

upper

```
final Number upper
```

Constructors

Bounds

```
public Bounds (Number lower, Number upper)
```

2.115.4 AuditableRandomGenerator.RandomMethod

```
public static enum RandomMethod
```

Enum Constants

BOUNDED_DOUBLE

```
public static final AuditableRandomGenerator.RandomMethod BOUNDED_DOUBLE
```

BOUNDED_INT

```
public static final AuditableRandomGenerator.RandomMethod BOUNDED_INT
```

DOUBLE

```
public static final AuditableRandomGenerator.RandomMethod DOUBLE
```

2.115.5 AuditableRandomGeneratorTest

```
public class AuditableRandomGeneratorTest
```


Methods

testAuditableRandomGeneratorProvidesCorrectAuditWhenGettingBoundedDouble

```
public void testAuditableRandomGeneratorProvidesCorrectAuditWhenGettingBoundedDouble ()
```

testAuditableRandomGeneratorProvidesCorrectAuditWhenGettingBoundedInteger

```
public void testAuditableRandomGeneratorProvidesCorrectAuditWhenGettingBoundedInteger ()
```

testAuditableRandomGeneratorProvidesCorrectAuditWhenGettingDouble

```
public void testAuditableRandomGeneratorProvidesCorrectAuditWhenGettingDouble ()
```

2.115.6 ExtendedPseudoRandomGenerator

public class **ExtendedPseudoRandomGenerator** implements *PseudoRandomGenerator*

Extended pseudo random number generator based on the decorator pattern. Two new methods are added: randNormal() and randSphere()

Author Antonio J. Nebro

Constructors

ExtendedPseudoRandomGenerator

```
public ExtendedPseudoRandomGenerator (PseudoRandomGenerator randomGenerator)
```

Methods

getName

```
public String getName ()
```

getSeed

```
public long getSeed ()
```

nextDouble

```
public double nextDouble (double lowerBound, double upperBound)
```

nextDouble

```
public double nextDouble ()
```

nextInt

public int **nextInt** (int *lowerBound*, int *upperBound*)

randNormal

public double **randNormal** (double *mean*, double *standardDeviation*)

Use the polar form of the Box-Muller transformation to obtain a pseudo random number from a Gaussian distribution Code taken from Maurice Clerc's implementation

Parameters

- **mean** –
- **standardDeviation** –

Returns A pseudo random number

randSphere

public double[] **randSphere** (int *dimension*)

Get a random point from an hypersphere (center = 0, radius = 1) Code taken from Maurice Clerc's implementation

Parameters

- **dimension** –

Returns A pseudo random point

randSphere

public double[] **randSphere** (int *dimension*, double *center*, double *radius*)

Ger a random point from an hypersphere Code taken from Maurice Clerc's implementation

Parameters

- **center** –
- **radius** –

Returns A pseudo random number

setSeed

public void **setSeed** (long *seed*)

2.115.7 JavaRandomGenerator

public class **JavaRandomGenerator** implements *PseudoRandomGenerator*

Author Antonio J. Nebro

Constructors

JavaRandomGenerator

```
public JavaRandomGenerator ()  
    Constructor
```

JavaRandomGenerator

```
public JavaRandomGenerator (long seed)  
    Constructor
```

Methods

getName

```
public String getName ()
```

getSeed

```
public long getSeed ()
```

nextDouble

```
public double nextDouble (double lowerBound, double upperBound)
```

nextDouble

```
public double nextDouble ()
```

nextInt

```
public int nextInt (int lowerBound, int upperBound)
```

setSeed

```
public void setSeed (long seed)
```

2.115.8 MersenneTwisterGenerator

```
public class MersenneTwisterGenerator implements PseudoRandomGenerator
```

Author Antonio J. Nebro

Constructors

MersenneTwisterGenerator

```
public MersenneTwisterGenerator ()  
    Constructor
```

MersenneTwisterGenerator

```
public MersenneTwisterGenerator (long seed)  
    Constructor
```

Methods

getName

```
public String getName ()
```

getSeed

```
public long getSeed ()
```

nextDouble

```
public double nextDouble (double lowerBound, double upperBound)
```

nextDouble

```
public double nextDouble ()
```

nextInt

```
public int nextInt (int lowerBound, int upperBound)
```

setSeed

```
public void setSeed (long seed)
```

2.115.9 Well44497bGenerator

```
public class Well44497bGenerator implements PseudoRandomGenerator
```

Author Antonio J. Nebro

Constructors

Well44497bGenerator

```
public Well44497bGenerator ()
    Constructor
```

Well44497bGenerator

```
public Well44497bGenerator (long seed)
    Constructor
```

Methods

getName

```
public String getName ()
```

getSeed

```
public long getSeed ()
```

nextDouble

```
public double nextDouble (double lowerBound, double upperBound)
```

nextDouble

```
public double nextDouble ()
```

nextInt

```
public int nextInt (int lowerBound, int upperBound)
```

setSeed

```
public void setSeed (long seed)
```

2.116 org.uma.jmetal.util.solutionattribute

2.116.1 DensityEstimator

```
public interface DensityEstimator<S> extends SolutionAttribute<S, Double>
    Interface representing implementations to compute the crowding distance
```

Author Antonio J. Nebro

Methods

computeDensityEstimator

public void **computeDensityEstimator** ([List](#)<[S](#)> *solutionSet*)

2.116.2 Ranking

public interface **Ranking**<[S](#)> extends [SolutionAttribute](#)<[S](#), [Integer](#)>
Ranks a list of solutions according to the dominance relationship

Author Antonio J. Nebro

Methods

computeRanking

public [Ranking](#)<[S](#)> **computeRanking** ([List](#)<[S](#)> *solutionList*)

getNumberOfSubfronts

public int **getNumberOfSubfronts** ()

getSubfront

public [List](#)<[S](#)> **getSubfront** (int *rank*)

2.116.3 SolutionAttribute

public interface **SolutionAttribute**<[S](#), [V](#)> extends [Serializable](#)
Attributes allows to extend the solution classes to incorporate data required by operators or algorithms manipulating them.

Author Antonio J. Nebro

Methods

getAttribute

public [V](#) **getAttribute** ([S](#) *solution*)

getAttributeIdentifier

public [Object](#) **getAttributeIdentifier** ()

setAttribute

```
public void setAttribute (S solution, V value)
```

2.117 org.uma.jmetal.util.solutionattribute.impl**2.117.1 CrowdingDistance**

```
public class CrowdingDistance<S extends Solution<?>> extends GenericSolutionAttribute<S, Double> implements DensityEstimator
```

This class implements the crowding distance

Author Antonio J. Nebro

Methods**computeDensityEstimator**

```
public void computeDensityEstimator (List<S> solutionList)
```

Assigns crowding distances to all solutions in a *SolutionSet*.

Parameters

- **solutionList** – The *SolutionSet*.

Throws

- *org.uma.jmetal.util.JMetalException* –

getAttributeIdentifier

```
public Object getAttributeIdentifier ()
```

2.117.2 CrowdingDistanceTest

```
public class CrowdingDistanceTest
```

Author Antonio J. Nebro

Methods**shouldTheCrowdingDistanceOfASingleSolutionBeInfinity**

```
public void shouldTheCrowdingDistanceOfASingleSolutionBeInfinity ()
```

shouldTheCrowdingDistanceOfAnEmptySetDoNothing

```
public void shouldTheCrowdingDistanceOfAnEmptySetDoNothing ()
```

shouldTheCrowdingDistanceOfThreeSolutionsCorrectlyAssigned

```
public void shouldTheCrowdingDistanceOfThreeSolutionsCorrectlyAssigned()
```

shouldTheCrowdingDistanceOfTwoSolutionsBeInfinity

```
public void shouldTheCrowdingDistanceOfTwoSolutionsBeInfinity()
```

2.117.3 DistanceToSolutionListAttribute

```
public class DistanceToSolutionListAttribute extends GenericSolutionAttribute<Solution<?>, Double>  
    Created by charba on 24/3/15.
```

2.117.4 DominanceRanking

```
public class DominanceRanking<S extends Solution<?>> extends GenericSolutionAttribute<S, Integer> implements Ranking<S>  
    This class implements some facilities for ranking set of solutions. Given a collection of solutions, they are  
    ranked according to scheme proposed in NSGA-II; as an output, a set of subsets are obtained. The subsets are  
    numbered starting from 0 (in NSGA-II, the numbering starts from 1); thus, subset 0 contains the non-dominated  
    solutions, subset 1 contains the non-dominated solutions after removing those belonging to subset 0, and so on.
```

Author Antonio J. Nebro , Juan J. Durillo

Constructors

DominanceRanking

```
public DominanceRanking (Comparator<S> comparator)  
    Constructor
```

DominanceRanking

```
public DominanceRanking ()  
    Constructor
```

DominanceRanking

```
public DominanceRanking (Object id)
```

Methods

computeRanking

```
public Ranking<S> computeRanking (List<S> solutionSet)
```


getNumberOfSubfronts

```
public int getNumberOfSubfronts ()
```

getSubfront

```
public List<S> getSubfront (int rank)
```

2.117.5 DominanceRankingTest

```
public class DominanceRankingTest
```

Author Antonio J. Nebro

Methods**shouldRankingOfAPopulationWithFiveSolutionsWorkProperly**

```
public void shouldRankingOfAPopulationWithFiveSolutionsWorkProperly ()
```

shouldRankingOfAPopulationWithThreeDominatedSolutionsReturnThreeSubfronts

```
public void shouldRankingOfAPopulationWithThreeDominatedSolutionsReturnThreeSubfronts ()
```

shouldRankingOfAPopulationWithTwoDominatedSolutionsReturnTwoSubfronts

```
public void shouldRankingOfAPopulationWithTwoDominatedSolutionsReturnTwoSubfronts ()
```

shouldRankingOfAPopulationWithTwoNonDominatedSolutionsReturnOneSubfront

```
public void shouldRankingOfAPopulationWithTwoNonDominatedSolutionsReturnOneSubfront ()
```

shouldTheRankingOfAnEmptyPopulationReturnOneSubfronts

```
public void shouldTheRankingOfAnEmptyPopulationReturnOneSubfronts ()
```

shouldTheRankingOfAnEmptyPopulationReturnZeroSubfronts

```
public void shouldTheRankingOfAnEmptyPopulationReturnZeroSubfronts ()
```

2.117.6 Fitness

```
public class Fitness<S extends Solution<?>> extends GenericSolutionAttribute<S, Double>
```

Author Antonio J. Nebro

2.117.7 GenericSolutionAttribute

public class **GenericSolutionAttribute**<S extends Solution<?>, V> implements *SolutionAttribute*<S, V>
Generic class for implementing *SolutionAttribute* classes. By default, the identifier of a *SolutionAttribute* is the class object, but it can be set to a different value when constructing an instance.

Author Antonio J. Nebro

Constructors

GenericSolutionAttribute

public **GenericSolutionAttribute** ()
Constructor

GenericSolutionAttribute

public **GenericSolutionAttribute** (*Object id*)
Constructor

Parameters

- **id** – Attribute identifier

Methods

getAttribute

public V **getAttribute** (S *solution*)

getAttributeIdentifier

public *Object* **getAttributeIdentifier** ()

setAttribute

public void **setAttribute** (S *solution*, V *value*)

2.117.8 GenericSolutionAttributeTest

public class **GenericSolutionAttributeTest**

Author Antonio J. Nebro

Methods

shouldConstructorCreateASolutionAttributedWithThePassedIdentifier

public void **shouldConstructorCreateASolutionAttributedWithThePassedIdentifier** ()

shouldDefaultConstructorCreateASolutionAttributedWithAnIdentifierEqualToTheClassObject

```
public void shouldDefaultConstructorCreateASolutionAttributedWithAnIdentifierEqualToTheClassObject()
```

shouldGetAttributeIdentifierReturnTheRightIdentifier

```
public void shouldGetAttributeIdentifierReturnTheRightIdentifier()
```

shouldGetAttributeReturnNullIfTheSolutionHasNoAttribute

```
public void shouldGetAttributeReturnNullIfTheSolutionHasNoAttribute()
```

shouldGetAttributeReturnTheAttributeValue

```
public void shouldGetAttributeReturnTheAttributeValue()
```

shouldSetAttributeAssignTheAttributeValueToTheSolution

```
public void shouldSetAttributeAssignTheAttributeValueToTheSolution()
```

2.117.9 HypervolumeContributionAttribute

```
public class HypervolumeContributionAttribute<S> extends Solution<?>> extends GenericSolutionAttribute<S, Double>
```

Author Antonio J. Nebro

2.117.10 LocationAttribute

```
public class LocationAttribute<S> extends Solution<?>> extends GenericSolutionAttribute<S, Integer>
```

Assign to each solution in a solution list an attribute containing the position of the solutions in the list.

Author Antonio J. Nebro

Parameters

- `<S>` –

Constructors**LocationAttribute**

```
public LocationAttribute(List<S> solutionList)
```

2.117.11 NumberOfViolatedConstraints

```
public class NumberOfViolatedConstraints<S> extends Solution<?>> extends GenericSolutionAttribute<S, Integer>
```

Author Antonio J. Nebro

2.117.12 OverallConstraintViolation

public class **OverallConstraintViolation**<S extends Solution<?>> extends *GenericSolutionAttribute*<S, Double>

Author Antonio J. Nebro

2.117.13 PreferenceDistance

public class **PreferenceDistance**<S extends Solution<?>> extends *GenericSolutionAttribute*<S, Double> implements *DensityEstimator*

Constructors

PreferenceDistance

public **PreferenceDistance** (*List*<Double> *interestPoint*, double *epsilon*)

Methods

computeDensityEstimator

public void **computeDensityEstimator** (*List*<S> *solutionList*)

epsilonClean

public *List*<S> **epsilonClean** (*List*<S> *solutionList*)

getSize

public int **getSize** ()

updatePointOfInterest

public void **updatePointOfInterest** (*List*<Double> *newInterestPoint*)

2.117.14 StrengthRawFitness

public class **StrengthRawFitness**<S extends Solution<?>> extends *GenericSolutionAttribute*<S, Double> implements *DensityEstimator*

Methods

computeDensityEstimator

public void **computeDensityEstimator** (*List*<S> *solutionSet*)

2.118 org.uma.jmetal.utility

2.118.1 GenerateReferenceFrontFromFile

public class **GenerateReferenceFrontFromFile**

This utility reads a file or the files in a directory and creates a reference front. The file(s) must contain only objective values. The program receives two parameters: 1. the name of the file or directory containing the data 2. the output file name which will contain the generated front

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] args)

2.119 org.uma.jmetal.workingTest

2.119.1 BLXAlphaCrossoverWorkingTest

public class **BLXAlphaCrossoverWorkingTest**

Author Antonio J. Nebro

Methods

main

public static void **main** (*String*[] args)

Program to generate data representing the distribution of points generated by a SBX crossover operator. The parameters to be introduced by the command line are: - numberOfSolutions: number of solutions to generate - granularity: number of subdivisions to be considered. - alpha: alpha value - outputFile: file containing the results

Parameters

- **args** – Command line arguments

2.119.2 BLXAlphaCrossoverWorkingTest.VariableComparator

public static class **VariableComparator** implements Comparator<*DoubleSolution*>

Methods

compare

public int **compare** (*DoubleSolution* solution1, *DoubleSolution* solution2)

Compares two solutions according to the first variable value

Parameters

- **solution1** – Object representing the first `Solution`.
- **solution2** – Object representing the second `Solution`.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.119.3 IntegerPolynomialMutationWorkingTest

public class **IntegerPolynomialMutationWorkingTest**

Author Antonio J. Nebro

Methods**main**

public static void **main** (`String[] args`)

Program to generate data representing the distribution of points generated by a polynomial mutation operator. The parameters to be introduced by the command line are: - numberOfSolutions: number of solutions to generate - granularity: number of subdivisions to be considered. - distributionIndex: distribution index of the polynomial mutation operator - outputFile: file containing the results

Parameters

- **args** – Command line arguments

2.119.4 IntegerPolynomialMutationWorkingTest.VariableComparator

public static class **VariableComparator** implements `Comparator<IntegerSolution>`

Methods**compare**

public int **compare** (`IntegerSolution solution1`, `IntegerSolution solution2`)

Compares two solutions according to the first variable value

Parameters

- **solution1** – Object representing the first `Solution`.
- **solution2** – Object representing the second `Solution`.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.119.5 IntegerSBXCrossoverWorkingTest

public class **IntegerSBXCrossoverWorkingTest**

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

Program to generate data representing the distribution of points generated by a SBX crossover operator. The parameters to be introduced by the command line are: - numberOfSolutions: number of solutions to generate - granularity: number of subdivisions to be considered. - distributionIndex: distribution index of the polynomial mutation operator - outputFile: file containing the results

Parameters

- **args** – Command line arguments

2.119.6 IntegerSBXCrossoverWorkingTest.VariableComparator

public static class **VariableComparator** implements Comparator<*IntegerSolution*>

Methods

compare

public int **compare** (*IntegerSolution solution1*, *IntegerSolution solution2*)

Compares two solutions according to the first variable value

Parameters

- **solution1** – Object representing the first *Solution*.
- **solution2** – Object representing the second *Solution*.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.119.7 PolynomialMutationWorkingTest

public class **PolynomialMutationWorkingTest**

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

Program to generate data representing the distribution of points generated by a polynomial mutation operator. The parameters to be introduced by the command line are: - numberOfSolutions: number of solutions to generate - granularity: number of subdivisions to be considered. - distributionIndex: distribution index of the polynomial mutation operator - outputFile: file containing the results

Parameters

- **args** – Command line arguments

2.119.8 PolynomialMutationWorkingTest.VariableComparator

public static class **VariableComparator** implements *Comparator<DoubleSolution>*

Methods

compare

public int **compare** (*DoubleSolution solution1*, *DoubleSolution solution2*)

Compares two solutions according to the first variable value

Parameters

- **solution1** – Object representing the first *Solution*.
- **solution2** – Object representing the second *Solution*.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

2.119.9 SBXCrossoverWorkingTest

public class **SBXCrossoverWorkingTest**

Author Antonio J. Nebro

Methods

main

public static void **main** (*String[] args*)

Program to generate data representing the distribution of points generated by a SBX crossover operator. The parameters to be introduced by the command line are: - numberOfSolutions: number of solutions to generate - granularity: number of subdivisions to be considered. - distributionIndex: distribution index of the polynomial mutation operator - outputFile: file containing the results

Parameters

- **args** – Command line arguments

2.119.10 SBXCrossoverWorkingTest.VariableComparator

public static class **VariableComparator** implements *Comparator<DoubleSolution>*

Methods

compare

public int **compare** (*DoubleSolution solution1*, *DoubleSolution solution2*)

Compares two solutions according to the first variable value

Parameters

- **solution1** – Object representing the first *Solution*.

- **solution2** – Object representing the second `Solution`.

Returns -1, or 0, or 1 if o1 is less than, equal, or greater than o2, respectively.

jMetal is being developed by [Antonio J. Nebro](#), associate professor at the University of Málaga.

3.1 References

1. J.J. Durillo, A.J. Nebro jMetal: a Java Framework for Multi-Objective Optimization. *Advances in Engineering Software* 42 (2011) 760-771.
2. A.J. Nebro, J.J. Durillo, M. Vergne Redesigning the jMetal Multi-Objective Optimization Framework. *GECCO (Companion)* 2015, pp: 1093-1100. July 2015.
3. Nebro A.J. et al. (2018) Extending the Speed-Constrained Multi-objective PSO (SMPSO) with Reference Point Based Preference Articulation. In: Auger A., Fonseca C., Lourenço N., Machado P., Paquete L., Whitley D. (eds) *Parallel Problem Solving from Nature – PPSN XV*. PPSN 2018. *Lecture Notes in Computer Science*, vol 11101. Springer, Cham

CHAPTER 4

Installation steps

A

- a (Java field), 406, 442
- AbstractAlgorithmRunner (Java class), 539
- AbstractBinaryProblem (Java class), 312
- AbstractBoundedArchive (Java class), 557
- AbstractBoundedArchive(int) (Java constructor), 557
- AbstractCDG (Java class), 37
- AbstractCDG(Problem, int, int, int, CrossoverOperator, double, double, int, int, int, int, int) (Java constructor), 41
- AbstractCoralReefsOptimization (Java class), 14
- AbstractCoralReefsOptimization(Comparator, SelectionOperator, CrossoverOperator, MutationOperator, int, int, double, double, double, double, int) (Java constructor), 15
- AbstractDifferentialEvolution (Java class), 19
- AbstractDoubleProblem (Java class), 313
- AbstractEvolutionaryAlgorithm (Java class), 20
- AbstractEvolutionStrategy (Java class), 19
- AbstractEvolutionStrategy(Problem) (Java constructor), 20
- AbstractGenericProblem (Java class), 313
- AbstractGenericSolution (Java class), 521
- AbstractGenericSolution(P) (Java constructor), 522
- AbstractGeneticAlgorithm (Java class), 22
- AbstractGeneticAlgorithm(Problem) (Java constructor), 22
- AbstractIntegerDoubleProblem (Java class), 314
- AbstractIntegerPermutationProblem (Java class), 315
- AbstractIntegerProblem (Java class), 315
- AbstractMOEAD (Java class), 99
- AbstractMOEAD(Problem, int, int, int, CrossoverOperator, MutationOperator, FunctionType, String, double, int, int) (Java constructor), 101
- AbstractMOMBI (Java class), 119
- AbstractMOMBI(Problem, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 120
- AbstractParticleSwarmOptimization (Java class), 24
- AbstractScatterSearch (Java class), 25
- AbstractUtilityFunctionsSet (Java class), 127
- AbstractUtilityFunctionsSet(double[][][]) (Java constructor), 127
- AbstractUtilityFunctionsSet(String) (Java constructor), 127
- ABYSS (Java class), 27
- ABYSS(DoubleProblem, int, int, int, int, int, Archive, LocalSearchOperator, CrossoverOperator, int) (Java constructor), 30
- ABYSSBuilder (Java class), 32
- ABYSSBuilder(DoubleProblem, Archive) (Java constructor), 32
- ABYSSIT (Java class), 34
- ABYSSRunner (Java class), 485
- ABYSSTest (Java class), 35
- AchievementScalarizationComparator (Java class), 67
- AchievementScalarizationComparator(int) (Java constructor), 67
- ackley(double[]) (Java method), 431
- AdaptiveGrid (Java class), 540
- AdaptiveGrid(int, int) (Java constructor), 540
- AdaptiveGridArchive (Java class), 558
- AdaptiveGridArchive(int, int, int) (Java constructor), 558
- AdaptiveGridArchiveTest (Java class), 559
- AdaptiveGridTest (Java class), 543
- AdaptiveRandomNeighborhood (Java class), 629
- AdaptiveRandomNeighborhood(int, int) (Java constructor), 629
- AdaptiveRandomNeighborhood(int, int, BoundedRandomGenerator) (Java constructor), 629
- AdaptiveRandomNeighborhoodTest (Java class), 630
- add(Entity) (Java method), 624
- add(List) (Java method), 129
- add(S) (Java method), 556–558, 561, 564
- addAll(Collection) (Java method), 624
- addIndicatorChart(String) (Java method), 573
- addLastRankedSolutionsToPopulation(R2Ranking, int, List) (Java method), 122
- addLastRankedSolutionsToPopulation(Ranking, int, List)

- (Java method), 76, 136, 196, 305, 307
 - `addListener(Consumer)` (Java method), 666
 - `AddMember()` (Java method), 152
 - `AddPotentialMember(S, double)` (Java method), 152
 - `addRankedSolutionsToPopulation(R2Ranking, int, List)` (Java method), 122
 - `addRankedSolutionsToPopulation(Ranking, int, List)` (Java method), 76, 136, 146, 196, 305, 307
 - `addSolution(int)` (Java method), 540
 - `AGG` (Java field), 54, 102
 - `algorithm` (Java field), 34, 37, 59, 80, 92, 115, 133, 141, 157, 178, 190, 544, 566
 - `Algorithm` (Java interface), 13
 - `AlgorithmBuilder` (Java interface), 544
 - `AlgorithmRunner` (Java class), 544
 - `allocateSolution()` (Java method), 41
 - `allReferencePoints` (Java field), 568
 - `alpha` (Java field), 123, 132
 - `alphaFunction(double[], List, int, int)` (Java method), 381
 - `ANGLE_UTILITY` (Java field), 74
 - `angleUtility(List)` (Java method), 70
 - `angleUtility(List, double[][])` (Java method), 70
 - `archive` (Java field), 27, 34, 35, 62, 82, 86, 89, 158, 190, 557
 - `Archive` (Java interface), 555
 - `ArchiveMutationLocalSearch` (Java class), 280
 - `ArchiveMutationLocalSearch(int, MutationOperator, Archive, Problem)` (Java constructor), 280
 - `archiveSize` (Java field), 28, 82, 158, 174
 - `ArchiveWithReferencePoint` (Java class), 563
 - `ArchiveWithReferencePoint(int, List, Comparator)` (Java constructor), 564
 - `AREA` (Java field), 345
 - `ArrayDoubleSolution` (Java class), 523
 - `ArrayDoubleSolution(ArrayDoubleSolution)` (Java constructor), 524
 - `ArrayDoubleSolution(DoubleProblem)` (Java constructor), 524
 - `ArrayDoubleSolutionTest` (Java class), 525
 - `ArrayFront` (Java class), 609
 - `ArrayFront()` (Java constructor), 610
 - `ArrayFront(Front)` (Java constructor), 610
 - `ArrayFront(int, int)` (Java constructor), 610
 - `ArrayFront(List)` (Java constructor), 610
 - `ArrayFront(String)` (Java constructor), 610
 - `ArrayFrontTest` (Java class), 611
 - `ArrayPoint` (Java class), 645
 - `ArrayPoint()` (Java constructor), 645
 - `ArrayPoint(double[])` (Java constructor), 645
 - `ArrayPoint(int)` (Java constructor), 645
 - `ArrayPoint(Point)` (Java constructor), 645
 - `ArrayPoint(String)` (Java constructor), 646
 - `ArrayPointTest` (Java class), 647
 - `ART_ART` (Java field), 345
 - `ART_RIG` (Java field), 345
 - `ArtificialDecisionMaker` (Java class), 566
 - `ArtificialDecisionMaker(Problem, InteractiveAlgorithm)` (Java constructor), 566
 - `ArtificialDecisionMakerDecisionTree` (Java class), 568
 - `ArtificialDecisionMakerDecisionTree(Problem, InteractiveAlgorithm, double, double, int, List, List)` (Java constructor), 570
 - `ArtificiallDecisionMakerBuilder` (Java class), 571
 - `ArtificiallDecisionMakerBuilder(Problem, InteractiveAlgorithm)` (Java constructor), 571
 - `ArtificiallDecisionMakerIT` (Java class), 37
 - `ASCENDING` (Java field), 583
 - `asexualReproduction(List)` (Java method), 15, 202
 - `ASFUtilityFunctionSet` (Java class), 125
 - `ASFUtilityFunctionSet(double[][])` (Java constructor), 125
 - `ASFUtilityFunctionSet(double[][][], List)` (Java constructor), 125
 - `ASFUtilityFunctionSet(String)` (Java constructor), 126
 - `ASFUtilityFunctionSet(String, List)` (Java constructor), 125
 - `ASFWASFGA` (Java class), 126
 - `ASFWASFGA(double[][])` (Java constructor), 126
 - `ASFWASFGA(double[][][], List)` (Java constructor), 126
 - `ASFWASFGA(String)` (Java constructor), 126
 - `ASFWASFGA(String, List)` (Java constructor), 126
 - `asp` (Java field), 568
 - `associate(List)` (Java method), 150
 - `attributes` (Java field), 521, 523, 642
 - `Audit` (Java class), 667
 - `Audit(RandomMethod, Bounds, Number)` (Java constructor), 667
 - `AuditableRandomGenerator` (Java class), 666
 - `AuditableRandomGenerator(PseudoRandomGenerator)` (Java constructor), 666
 - `AuditableRandomGeneratorTest` (Java class), 668
 - `averageDistanceToSolutionList(S, List)` (Java method), 553
 - `Ax(double[], double[][], double[])` (Java method), 430
 - `aX_` (Java field), 359
 - `AxialForcei_` (Java field), 346
 - `AxialForcei_(int)` (Java method), 363
 - `AxialForcej_` (Java field), 346
 - `AxialForcej_(int)` (Java method), 363
 - `Ay_` (Java field), 346
 - `aY_` (Java field), 359
 - `Az_` (Java field), 346
 - `aZ_` (Java field), 359
- ## B
- `b` (Java field), 442
 - `badPopulation` (Java field), 37
 - `badSolution` (Java field), 38

- badSolutionNum (Java field), 38
 - basicFunc(int, double[]) (Java method), 456
 - BasicLocalSearch (Java class), 281
 - BasicLocalSearch(int, MutationOperator, Comparator, Problem) (Java constructor), 281
 - BasicLocalSearch(int, MutationOperator, Comparator, Problem, RandomGenerator) (Java constructor), 282
 - BasicLocalSearchTest (Java class), 282
 - BasicMeasure (Java class), 238
 - BasicMeasure() (Java constructor), 238
 - Benchmark (Java class), 429
 - Benchmark() (Java constructor), 430
 - Benchmark(String) (Java constructor), 430
 - BEST_FEASIBLE_POSITION (Java field), 69
 - BestSolutionSelection (Java class), 297
 - BestSolutionSelection(Comparator) (Java constructor), 298
 - BETA (Java field), 346
 - betaFunction(List, int) (Java method), 381
 - bFlat(float, float, float, float) (Java method), 405
 - bias() (Java method), 457
 - biases (Java field), 455
 - BigOpt2015 (Java class), 339
 - BigOpt2015(String) (Java constructor), 340
 - BinaryProblem (Java interface), 308
 - BinaryProblemsStudy (Java class), 230
 - BinarySet (Java class), 572
 - BinarySet(int) (Java constructor), 572
 - BinarySolution (Java interface), 515
 - BinaryTournamentSelection (Java class), 298
 - BinaryTournamentSelection() (Java constructor), 298
 - BinaryTournamentSelection(Comparator) (Java constructor), 298
 - BinaryTournamentSelectionTest (Java class), 298
 - Binh2 (Java class), 316
 - Binh2() (Java constructor), 316
 - biSections (Java field), 158
 - BitFlipMutation (Java class), 283
 - BitFlipMutation(double) (Java constructor), 283
 - BitFlipMutation(double, RandomGenerator) (Java constructor), 283
 - BitFlipMutationTest (Java class), 284
 - BLijY_ (Java field), 346
 - BLijZ_ (Java field), 346
 - BLXAlphaCrossover (Java class), 261
 - BLXAlphaCrossover(double) (Java constructor), 262
 - BLXAlphaCrossover(double, double) (Java constructor), 262
 - BLXAlphaCrossover(double, double, RepairDoubleSolution) (Java constructor), 262
 - BLXAlphaCrossover(double, double, RepairDoubleSolution, RandomGenerator) (Java constructor), 262
 - BLXAlphaCrossoverTest (Java class), 263
 - BLXAlphaCrossoverWorkingTest (Java class), 681
 - border (Java field), 38
 - borderLength (Java field), 38
 - bound(RandomGenerator) (Java method), 660
 - BOUNDED_DOUBLE (Java field), 668
 - BOUNDED_INT (Java field), 668
 - BoundedArchive (Java interface), 556
 - BoundedRandomGenerator (Java interface), 660
 - BoundedRandomGeneratorTest (Java class), 661
 - Bounds (Java class), 668
 - Bounds(Number, Number) (Java constructor), 668
 - bParam(float, float, float, float, float) (Java method), 405
 - bPoly(float, float) (Java method), 405
 - BucklingOmega(double, double[], double[]) (Java method), 364
 - build() (Java method), 32, 48, 55, 64, 79, 85, 90, 97, 106, 138, 148, 151, 156, 160, 163, 166, 168, 174, 188, 193, 204, 209, 214, 216, 221, 519, 544, 571, 595
 - buildAlgorithm() (Java method), 140, 210
 - Builder (Java class), 150, 214
 - Builder(DoubleProblem) (Java constructor), 214
 - buildNewReferenceSet1() (Java method), 30
 - buildNewReferenceSet2() (Java method), 30
- ## C
- C (Java field), 455
 - C1_DTLZ1 (Java class), 334
 - C1_DTLZ1(int, int) (Java constructor), 335
 - C1_DTLZ3 (Java class), 335
 - C1_DTLZ3(int, int) (Java constructor), 336
 - C25 (Java class), 631
 - C25(int, int) (Java constructor), 631
 - C2_DTLZ2 (Java class), 336
 - C2_DTLZ2(int, int) (Java constructor), 336
 - C3_DTLZ1 (Java class), 337
 - C3_DTLZ1(int, int, int) (Java constructor), 337
 - C3_DTLZ4 (Java class), 337
 - C3_DTLZ4(int, int, int) (Java constructor), 338
 - C49 (Java class), 631
 - C49(int, int) (Java constructor), 632
 - C9 (Java class), 632
 - C9(int, int) (Java constructor), 632
 - C9Test (Java class), 632
 - calculateDistance(DoubleSolution, double[]) (Java method), 116
 - calculateDistance(S, double[], double[], double[]) (Java method), 110
 - calculateDistance2(DoubleSolution, double[]) (Java method), 117
 - calculateDistance2(S, double[], double[], double[]) (Java method), 110
 - calculateFitness(List) (Java method), 84

- calculateHypervolume(double[[]], int, int) (Java method), 479
- calculateHypervolumeIndicator(Solution, Solution, int, double[], double[]) (Java method), 84
- calculateOccupied() (Java method), 540
- calculateReferencePoints(List, List, List) (Java method), 570
- calculateReferencePoints(List, R, List) (Java method), 566
- calculateX(float[]) (Java method), 407
- CARGA__TRIANGULAR_J (Java field), 347
- CARGA_MOMENTO_DISTRIBUIDO (Java field), 346
- CARGA_MOMENTO_PUNTUAL (Java field), 346
- CARGA_PARABOLICA (Java field), 346
- CARGA_PUNTUAL (Java field), 347
- CARGA_TEMPERATURA (Java field), 347
- CARGA_TRIANGULAR_I (Java field), 347
- CARGA_UNIFORME_PARCIAL (Java field), 347
- CARGA_UNIFORME_TOTAL (Java field), 347
- cataclysmicMutation (Java field), 95
- cbi (Java field), 359
- cbj (Java field), 359
- CDG (Java class), 45
- CDG(Problem, int, int, int, CrossoverOperator, double, double, int, int, int, int, int) (Java constructor), 46
- CDGBuilder (Java class), 46
- CDGBuilder(Problem) (Java constructor), 48
- CDGMutation (Java class), 285
- CDGMutation() (Java constructor), 285
- CDGMutation(double, double) (Java constructor), 285
- CDGMutation(double, double, RepairDoubleSolution) (Java constructor), 285
- CDGMutation(DoubleProblem, double) (Java constructor), 285
- CDGRunner (Java class), 485
- CEC2005Problem (Java class), 424
- CEC2005Problem(int, int) (Java constructor), 425
- CEC2005SUPPORTDATADIRECTORY (Java field), 429
- CellIDE (Java class), 50
- CellIDE45 (Java class), 50
- CellIDE45(Problem, int, int, BoundedArchive, Neighborhood, SelectionOperator, DifferentialEvolutionCrossover, double, SolutionListEvaluator) (Java constructor), 50
- CellIDERunner (Java class), 486
- changeReferenceFrontTo(String) (Java method), 604
- changeReferencePoint(List) (Java method), 564
- changeReferencePoints(List) (Java method), 182
- ChartContainer (Java class), 573
- ChartContainer(String) (Java constructor), 573
- ChartContainer(String, int) (Java constructor), 573
- ChartContainerWithReferencePoints (Java class), 575
- ChartContainerWithReferencePoints(String) (Java constructor), 575
- ChartContainerWithReferencePoints(String, int) (Java constructor), 575
- CHEBYSHEV (Java field), 74
- chebyshev(List) (Java method), 70
- chebyshev(List, double[]) (Java method), 70
- checkDominance(S, S) (Java method), 110
- checkEigenSystem(int, double[[]], double[], double[[]]) (Java method), 219
- checkNumberOfParents(List, int) (Java method), 22
- checkWithJdkGeneralPath(Point2D.Double, List) (Java method), 393
- childGrid_ (Java field), 38, 46
- childGridNum_ (Java field), 38, 46
- chooseNeighborType() (Java method), 101
- chooseNeighborType(int) (Java method), 41
- chooseSolution() (Java method), 41
- chooseSpecialPopulation() (Java method), 41
- CIRCLE (Java field), 347
- clean() (Java method), 653
- cleanup() (Java method), 140, 210
- clear() (Java method), 153, 624
- CMAESUtils (Java class), 219
- CommandLineIndicatorRunner (Java class), 458
- comparator (Java field), 14, 158, 563
- compare(DoubleSolution, DoubleSolution) (Java method), 681, 684
- compare(IntegerSolution, IntegerSolution) (Java method), 682, 683
- compare(Point, Point) (Java method), 658, 659
- compare(S, S) (Java method), 67, 577, 579, 581–583, 585, 586, 588, 589
- compareDoubleSolutionImplementationsWhenCreatingSolutions() (Java method), 532
- compareDoubleSolutionImplementationsWhenEvaluatingSolutions() (Java method), 533
- compareTo(Coordinate) (Java method), 18
- COMPRESSION (Java field), 347
- compute(Point, Point) (Java method), 659, 660
- computeDensityEstimator() (Java method), 68, 556, 559, 560, 565
- computeDensityEstimator(List) (Java method), 674, 675, 680
- computeHypervolume(List, Point) (Java method), 481
- computeHypervolumeContribution(List, List) (Java method), 468, 479, 481
- computeIndicatorValuesHD(List, double[], double[]) (Java method), 84
- ComputeQualityIndicators (Java class), 597
- ComputeQualityIndicators(Experiment) (Java constructor), 598
- computeRanking(List) (Java method), 50, 76, 81, 110, 122, 124, 130, 131, 136, 146, 171, 186, 196,

- 200, 674, 676
- computingTime (Java field), 545
- concave(float[], int) (Java method), 404
- Config (Java class), 73
- configureAlgorithmList(List) (Java method), 230–235
- configureLoggers(File) (Java method), 546
- considerationProbability (Java field), 568
- const2 (Java field), 387
- const4 (Java field), 388
- const5 (Java field), 389
- const8 (Java field), 391
- ConstrainedProblem (Java interface), 309
- CONSTRAINT (Java field), 347
- ConstraintMOEAD (Java class), 103
- ConstraintMOEAD (Java field), 109
- ConstraintMOEAD(Problem, int, int, int, MutationOperator, CrossoverOperator, FunctionType, String, double, int, int) (Java constructor), 103
- ConstraintProblemsStudy (Java class), 230
- ConstraintViolationComparator (Java interface), 577
- ConstrEx (Java class), 317
- ConstrEx() (Java constructor), 317
- constrictionCoefficient(double, double) (Java method), 171
- constructHyperplane(List, List) (Java method), 150
- contains(Object) (Java method), 625
- contains(String) (Java method), 625
- containsAll(Collection) (Java method), 625
- convergenceValue (Java field), 95
- convertFrontToArray(Front) (Java method), 622
- convertFrontToSolutionList(Front) (Java method), 622
- convex(float[], int) (Java method), 404
- ConvexC2_DTLZ2 (Java class), 338
- ConvexC2_DTLZ2(int, int) (Java constructor), 339
- Coordinate (Java class), 18
- Coordinate(int, int) (Java constructor), 18
- coordinates (Java field), 14
- copy() (Java method), 518, 524, 526, 528–532, 643
- CoralReefsOptimization (Java class), 202
- CoralReefsOptimization(Problem, int, Comparator, SelectionOperator, CrossoverOperator, MutationOperator, int, int, double, double, double, double, int) (Java constructor), 202
- CoralReefsOptimizationBuilder (Java class), 203
- CoralReefsOptimizationBuilder(Problem, SelectionOperator, CrossoverOperator, MutationOperator) (Java constructor), 204
- CoralReefsOptimizationRunner (Java class), 511
- correctTo01(float) (Java method), 405, 407
- correlation(List, List) (Java method), 340
- CosineDistanceBetweenSolutionsInObjectiveSpace (Java class), 590
- CosineDistanceBetweenSolutionsInObjectiveSpace(S) (Java constructor), 590
- CosineDistanceBetweenSolutionsInObjectiveSpaceTest (Java class), 590
- costMatrix (Java field), 320
- count (Java field), 239
- CountingMeasure (Java class), 239
- CountingMeasure() (Java constructor), 240
- CountingMeasure(long) (Java constructor), 240
- CountingMeasure(String, String) (Java constructor), 240
- CountingMeasure(String, String, long) (Java constructor), 239
- CountingMeasureTest (Java class), 241
- countOnes(int) (Java method), 110
- countRankOnes(int) (Java method), 111
- countTest() (Java method), 111
- CovarianceMatrixAdaptationEvolutionStrategy (Java class), 213
- CovarianceMatrixAdaptationEvolutionStrategyRunner (Java class), 511
- createFromGetters(Class) (Java method), 533, 534
- createFromGettersAndConstructors(Class) (Java method), 535
- createFromGettersAndSetters(Class) (Java method), 535
- createFromGettersWithoutSetters(Class) (Java method), 533
- createInitialPopulation() (Java method), 15, 20, 23, 50, 76, 88, 93, 136, 158, 202, 207, 213, 215, 218
- createInitialSwarm() (Java method), 24, 53, 154, 172, 182, 225, 228
- createInputStream(String) (Java method), 611
- createPullFromPush(PushMeasure, Value) (Java method), 249
- createPullsFromFields(Object) (Java method), 249
- createPullsFromGetters(Object) (Java method), 249
- createPushFromPull(PullMeasure, long) (Java method), 250
- createSolution() (Java method), 311–313, 315, 321, 322, 370, 407, 426
- createUtilityFunction() (Java method), 196
- createUtilityFunction(String) (Java method), 122, 124
- crossover (Java field), 28, 35, 46, 92, 105
- crossoverOperator (Java field), 14, 19, 22, 38, 78, 82, 89, 96, 99, 135, 187, 192
- CrossoverOperator (Java interface), 260
- CrowdingDistance (Java class), 675
- crowdingDistance (Java field), 75
- CrowdingDistanceArchive (Java class), 559
- CrowdingDistanceArchive(int) (Java constructor), 560
- CrowdingDistanceArchiveWithReferencePoint (Java class), 564
- CrowdingDistanceArchiveWithReferencePoint(int, List) (Java constructor), 564
- CrowdingDistanceComparator (Java class), 577
- crowdingDistanceComparator (Java field), 28
- CrowdingDistanceComparatorTest (Java class), 578

crowdingDistanceSelection(Ranking) (Java method), 76, 136, 305

CrowdingDistanceTest (Java class), 675

currentIndividual (Java field), 86

currentIteration (Java field), 181

currentNeighbors (Java field), 87

currentReferencePoint (Java field), 568

D

d (Java field), 406

d_ (Java field), 38

dataDirectory (Java field), 51, 99, 105

DecisionTreeEstimator (Java class), 567

DecisionTreeEstimator(List) (Java constructor), 568

decreaseMark(int) (Java method), 129

DEFAULT_FILE_BIAS (Java field), 429

DEFAULT_FILE_DATA (Java field), 434–454

DEFAULT_FILE_MX_PREFIX (Java field), 435, 438–441, 444–454

DEFAULT_FILE_MX_SUFFIX (Java field), 435, 438, 439, 441, 444, 446–454

DefaultBinarySolution (Java class), 526

DefaultBinarySolution(BinaryProblem) (Java constructor), 526

DefaultBinarySolution(DefaultBinarySolution) (Java constructor), 526

DefaultBinarySolutionTest (Java class), 527

DefaultDoubleBinarySolution (Java class), 528

DefaultDoubleBinarySolution(DefaultDoubleBinarySolution) (Java constructor), 528

DefaultDoubleBinarySolution(DoubleBinaryProblem) (Java constructor), 528

DefaultDoubleSolution (Java class), 529

DefaultDoubleSolution(DefaultDoubleSolution) (Java constructor), 529

DefaultDoubleSolution(DoubleProblem) (Java constructor), 529

DefaultFileOutputContext (Java class), 608

DefaultFileOutputContext(String) (Java constructor), 608

DefaultIntegerDoubleSolution (Java class), 530

DefaultIntegerDoubleSolution(DefaultIntegerDoubleSolution) (Java constructor), 530

DefaultIntegerDoubleSolution(IntegerDoubleProblem) (Java constructor), 530

DefaultIntegerPermutationSolution (Java class), 531

DefaultIntegerPermutationSolution(DefaultIntegerPermutationSolution) (Java constructor), 531

DefaultIntegerPermutationSolution(PermutationProblem) (Java constructor), 531

DefaultIntegerPermutationSolutionTest (Java class), 531

DefaultIntegerSolution (Java class), 531

DefaultIntegerSolution(DefaultIntegerSolution) (Java constructor), 532

DefaultIntegerSolution(IntegerProblem) (Java constructor), 532

deleteCrowdIndiv_diff(int, int, int, S) (Java method), 111

deleteCrowdIndiv_same(int, int, double, S) (Java method), 111

deleteCrowdRegion1(S, int) (Java method), 111

deleteCrowdRegion2(S, int) (Java method), 111

deleteRankOne(S, int) (Java method), 111

deltaMax (Java field), 181

deltaMin (Java field), 181

DensityEstimator (Java interface), 673

depredation(List, List) (Java method), 15, 202

DESCENDING (Java field), 584

DescribedEntity (Java interface), 624

DescribedEntitySet (Java class), 624

DescribedEntitySetTest (Java class), 626

DESCRIPTION (Java field), 347

description (Java field), 474

diagonal1(List) (Java method), 340

diagonal2(List) (Java method), 340

DifferentialEvolution (Java class), 206

DifferentialEvolution(DoubleProblem, int, int, DifferentialEvolutionCrossover, DifferentialEvolutionSelection, SolutionListEvaluator) (Java constructor), 207

DifferentialEvolutionBuilder (Java class), 208

DifferentialEvolutionBuilder(DoubleProblem) (Java constructor), 208

DifferentialEvolutionBuilderTest (Java class), 210

DifferentialEvolutionCrossover (Java class), 264

differentialEvolutionCrossover (Java field), 104, 113, 116

DifferentialEvolutionCrossover() (Java constructor), 265

DifferentialEvolutionCrossover(double, double, double, String) (Java constructor), 266

DifferentialEvolutionCrossover(double, double, String) (Java constructor), 265

DifferentialEvolutionCrossover(double, double, String, BoundedRandomGenerator, BoundedRandomGenerator) (Java constructor), 265

DifferentialEvolutionCrossover(double, double, String, RandomGenerator) (Java constructor), 265

DifferentialEvolutionCrossoverTest (Java class), 267

DifferentialEvolutionRunner (Java class), 511

DifferentialEvolutionSelection (Java class), 299

DifferentialEvolutionSelection() (Java constructor), 299

DifferentialEvolutionSelection(BoundedRandomGenerator) (Java constructor), 299

DifferentialEvolutionSelectionTest (Java class), 300

DifferentialEvolutionTestIT (Java class), 211

dimension() (Java method), 457

disc(float[], int, float, float) (Java method), 404

DisplacementNodes(int, int) (Java method), 364

DisplacementNodes_ (Java field), 347

Distance (Java interface), 590

[illegible]

DurationMeasureTest (Java class), 243
 dY_ (Java field), 359

E

E_ (Java field), 348
 Ebes (Java class), 345
 Ebes() (Java constructor), 363
 Ebes(String, String[]) (Java constructor), 363
 EBEsAssignAxialForces(int) (Java method), 364
 EBEsCalculus() (Java method), 364
 EBEsEcuationSolution(int) (Java method), 364
 EBEsEffortsElements3D(int, int, double[][]) (Java method), 364
 EBEsEffortsTotal3D(int) (Java method), 364
 EBEsElementsTopology(DoubleSolution) (Java method), 364
 EBEsInitialize(String) (Java method), 364
 EBEsMat3DG(int) (Java method), 364
 EBEsMat3DGij() (Java method), 364
 EBEsMat3DL_iArt_jArt(int) (Java method), 365
 EBEsMat3DL_iArt_jRig(int) (Java method), 365
 EBEsMat3DL_iRig_jArt(int) (Java method), 365
 EBEsMat3DL_iRig_jRig(int) (Java method), 365
 EBEsMat3DL_SOG(int) (Java method), 365
 EBEsMatrixAdd(double[][], double[][]) (Java method), 365
 EBEsMatrixGlobalFactory(int) (Java method), 365
 EBEsMatrixGlobalPenalization() (Java method), 365
 EBEsMatrixSubtractions(double[][], double[][]) (Java method), 365
 EBEsMatrixWeight(int) (Java method), 366
 EBEsMatrizMultiplicar(double[][], double[][]) (Java method), 366
 EBEsMatrizTraspuesta(double[][]) (Java method), 366
 EBEsMatrizVectorMultiplicar(double[][], double[]) (Java method), 366
 EBEsMatRot3DLaG(int) (Java method), 365
 EBEsMatRot3DLpSaL(int) (Java method), 365
 EBEsNodesEquilibrium3D(int) (Java method), 366
 EBEsOverloadWeightElement() (Java method), 366
 EBEsPrintArchTxtDesp(int) (Java method), 366
 EBEsPrintArchTxtEfforts(int) (Java method), 366
 EBEsPrintArchTxtElements() (Java method), 366
 EBEsPrintArchTxtMKG(String, int) (Java method), 366
 EBEsPrintArchTxtMKLB(int) (Java method), 366
 EBEsPrintArchTxtReaction(int) (Java method), 367
 EBEsPrintArchTxtStrain() (Java method), 367
 EBEsReactions3D(int) (Java method), 367
 EBEsReadDataFile(String) (Java method), 367
 EBEsReadProblems() (Java method), 367
 EBEsSteelingResults(int) (Java method), 367
 EBEsStrainMaxWhitElement() (Java method), 367
 EBEsStrainMaxWhitGroup() (Java method), 367
 EBEsStrainMinWhitElement() (Java method), 367
 EBEsStrainMinWhitGroup() (Java method), 367
 EBEsStrainNode(double[][][]) (Java method), 367
 EBEsStrainResidualVerication() (Java method), 368
 EBEsTransversalSection_H_Double(int, double, double, double, double) (Java method), 368
 EBEsTransversalSection_H_Single(int, double, double, double, double) (Java method), 368
 EBEsTransversalSection_I_Double(int, double, double, double, double) (Java method), 368
 EBEsTransversalSection_I_Single(int, double, double, double, double) (Java method), 368
 EBEsTransversalSection_L_Double(int, double, double, double, double) (Java method), 368
 EBEsTransversalSection_L_Single(int, double, double, double, double) (Java method), 368
 EBEsTransversalSection_T_Double(int, double, double, double, double) (Java method), 369
 EBEsTransversalSection_T_Single(int, double, double, double, double) (Java method), 369
 EBEsTransversalSectionCircular(int, double) (Java method), 368
 EBEsTransversalSectionHoleCircular(int, double, double) (Java method), 368
 EBEsTransversalSectionHoleRectangle(int, double, double, double, double) (Java method), 368
 EBEsTransversalSectionRectangle(int, double, double) (Java method), 368
 EBEsWeightDistributedUniformly(int, double[]) (Java method), 369
 EBEsWeightNodes() (Java method), 369
 EBEsWeigthElement() (Java method), 369
 Effort_i(int, int, int) (Java method), 369
 Effort_i_ (Java field), 348
 Effort_j(int, int, int) (Java method), 369
 Effort_j_ (Java field), 348
 Ei_ (Java field), 348
 Ej_ (Java field), 348
 Element_ (Java field), 348
 elementsBetweenDiffGreat_ (Java field), 360
 ELITIST (Java field), 217
 ElitistEvolutionStrategy (Java class), 215
 ElitistEvolutionStrategy(Problem, int, int, int, Mutation-Operator) (Java constructor), 215
 ElitistEvolutionStrategyRunner (Java class), 512
 elliptic(double[]) (Java method), 431
 ELONGATION_NEG (Java field), 348
 ELONGATION_POS (Java field), 348
 EnergyArchive (Java class), 67
 EnergyArchive(int) (Java constructor), 67
 EnergyArchive(int, ScalarizationWrapper) (Java constructor), 68
 EnergyArchive(int, ScalarizationWrapper, boolean) (Java constructor), 68

- EnergyArchive(int, ScalarizationWrapper, boolean, ReplacementStrategy) (Java constructor), 68
- EnumValues (Java enum), 666
- EnvironmentalSelection (Java class), 149, 194
- environmentalSelection (Java field), 190
- EnvironmentalSelection(Builder) (Java constructor), 149
- EnvironmentalSelection(int) (Java constructor), 194
- EnvironmentalSelection(List, int, List, int) (Java constructor), 149
- Epsilon (Java class), 458
- epsilon (Java field), 123, 195, 331, 332, 334
- Epsilon() (Java constructor), 459
- Epsilon(Front) (Java constructor), 459
- Epsilon(String) (Java constructor), 459
- epsilonClean(List) (Java method), 680
- EpsilonTest (Java class), 460
- epsilonValue (Java field), 60
- equalComparator (Java field), 28
- equals(Object) (Java method), 18, 522, 524, 611, 643, 646
- EqualSolutionsComparator (Java class), 581
- ErrorRatio (Java class), 461
- ErrorRatio(Front) (Java constructor), 461
- ErrorRatio(String) (Java constructor), 461
- ErrorRatioTest (Java class), 462
- EScatterF6(double[]) (Java method), 430
- EScatterF6NonCont(double[]) (Java method), 430
- ESPEA (Java class), 61
- ESPEA(Problem, int, int, CrossoverOperator, CrossoverOperator, MutationOperator, SelectionOperator, ScalarizationWrapper, SolutionListEvaluator, boolean, ReplacementStrategy) (Java constructor), 62
- ESPEABuilder (Java class), 63
- ESPEABuilder(Problem, CrossoverOperator, MutationOperator) (Java constructor), 64
- ESPEARunner (Java class), 487
- EuclideanDistance (Java class), 659
- EuclideanDistanceBetweenSolutionAndASolutionListInObjectiveSpace (Java class), 591
- EuclideanDistanceBetweenSolutionAndASolutionListInObjectiveSpace (Java constructor), 591
- EuclideanDistanceBetweenSolutionsInObjectiveSpace (Java class), 591
- EuclideanDistanceBetweenSolutionsInSolutionSpace (Java class), 591
- EuclideanDistanceTest (Java class), 656
- evalG(BinarySolution) (Java method), 423
- evalG(DoubleSolution) (Java method), 422, 424
- evalH(double, double) (Java method), 419–424
- evaluate(BinarySolution) (Java method), 322, 423, 426
- evaluate(DoubleSolution) (Java method), 316–319, 323–334, 340–345, 370, 377–380, 382–393, 395–404, 408, 410–413, 415–422, 424, 425, 427, 428
- evaluate(Evaluate) (Java method), 458, 461, 474
- evaluate(float[]) (Java method), 407, 408, 410–414, 416–418
- evaluate(IntegerDoubleSolution) (Java method), 321
- evaluate(IntegerSolution) (Java method), 321, 426
- evaluate(List) (Java method), 459, 463, 465, 470, 471, 478, 479, 481
- evaluate(List, List) (Java method), 475
- evaluate(List, Problem) (Java method), 592, 593
- evaluate(Pair) (Java method), 475
- evaluate(PermutationSolution) (Java method), 320, 428
- evaluate(S) (Java method), 127, 170, 312
- evaluate(S, int) (Java method), 126, 127, 132
- evaluateConstraints(DoubleSolution) (Java method), 316, 317, 319, 323–325, 327, 328, 335–339, 371
- evaluateConstraints(S) (Java method), 309
- evaluatePopulation(List) (Java method), 15, 20, 51, 62, 76, 88, 93, 120, 134, 137, 144, 146, 158, 162, 186, 191, 202, 207, 213, 215, 218, 220, 223
- evaluateSimpleSolutions() (Java method), 322
- evaluateSwarm(List) (Java method), 24, 53, 154, 172, 183, 226, 228
- Evaluation (Java class), 245
- evaluations (Java field), 28, 38, 50, 62, 75, 87, 99, 133, 135, 142, 158, 185, 195, 569
- evaluator (Java field), 62, 75, 78, 87, 89, 96, 119, 133, 135, 145, 153, 155, 161, 174, 181, 190, 192
- evalV(double) (Java method), 423
- EvolutionStrategyBuilder (Java class), 216
- EvolutionStrategyBuilder(Problem, MutationOperator, EvolutionStrategyVariant) (Java constructor), 216
- EvolutionStrategyVariant (Java enum), 217
- exception (Java field), 300, 302, 303, 306, 308, 460, 462, 466, 468, 472, 476, 550, 579, 584, 612, 615, 620, 630, 640
- excludeBadSolution() (Java method), 41
- excludeBadSolution3() (Java method), 41
- execute() (Java method), 545
- execute(AdaptiveGridArchive) (Java method), 165
- execute(BinarySolution) (Java method), 283
- execute(DoubleSolution) (Java method), 285, 289, 293, 295, 297
- execute(IntegerSolution) (Java method), 287
- execute(List) (Java method), 73, 150, 195, 262, 266, 268, 269, 271, 273, 274, 277, 280, 298, 300, 301, 303–305, 307, 308
- execute(PermutationSolution) (Java method), 291
- execute(S) (Java method), 281, 282, 291
- execute(Source) (Java method), 261
- ExecuteAlgorithms (Java class), 598
- ExecuteAlgorithms(Experiment) (Java constructor), 598
- Executor (Java class), 544
- Executor(Algorithm) (Java constructor), 545

- Experiment (Java class), [593](#)
Experiment(ExperimentBuilder) (Java constructor), [593](#)
ExperimentAlgorithm (Java class), [602](#)
ExperimentAlgorithm(Algorithm, ExperimentProblem, int) (Java constructor), [603](#)
ExperimentAlgorithm(Algorithm, String, ExperimentProblem, int) (Java constructor), [602](#)
ExperimentBuilder (Java class), [595](#)
ExperimentBuilder(String) (Java constructor), [595](#)
ExperimentComponent (Java interface), [597](#)
ExperimentProblem (Java class), [603](#)
ExperimentProblem(Problem) (Java constructor), [604](#)
ExperimentProblem(Problem, String) (Java constructor), [603](#)
ExtendedPseudoRandomGenerator (Java class), [669](#)
ExtendedPseudoRandomGenerator(PseudoRandomGenerator) (Java constructor), [669](#)
ExtremeValuesFinder (Java interface), [604](#)
eY_ (Java field), [360](#)
eZ_ (Java field), [360](#)
- ## F
- f(double[]) (Java method), [434–455](#), [457](#)
F01ShiftedSphere (Java class), [434](#)
F01ShiftedSphere(int, double) (Java constructor), [434](#)
F01ShiftedSphere(int, double, String) (Java constructor), [434](#)
F02ShiftedSchwefel (Java class), [434](#)
F02ShiftedSchwefel(int, double) (Java constructor), [435](#)
F02ShiftedSchwefel(int, double, String) (Java constructor), [435](#)
F03ShiftedRotatedHighCondElliptic (Java class), [435](#)
F03ShiftedRotatedHighCondElliptic(int, double) (Java constructor), [435](#)
F03ShiftedRotatedHighCondElliptic(int, double, String, String) (Java constructor), [436](#)
F04ShiftedSchwefelNoise (Java class), [436](#)
F04ShiftedSchwefelNoise(int, double) (Java constructor), [436](#)
F04ShiftedSchwefelNoise(int, double, String) (Java constructor), [436](#)
F05SchwefelGlobalOptBound (Java class), [436](#)
F05SchwefelGlobalOptBound(int, double) (Java constructor), [437](#)
F05SchwefelGlobalOptBound(int, double, String) (Java constructor), [437](#)
F06ShiftedRosenbrock (Java class), [437](#)
F06ShiftedRosenbrock(int, double) (Java constructor), [437](#)
F06ShiftedRosenbrock(int, double, String) (Java constructor), [438](#)
F07ShiftedRotatedGriewank (Java class), [438](#)
F07ShiftedRotatedGriewank(int, double) (Java constructor), [438](#)
F07ShiftedRotatedGriewank(int, double, String, String) (Java constructor), [438](#)
F08ShiftedRotatedAckleyGlobalOptBound (Java class), [439](#)
F08ShiftedRotatedAckleyGlobalOptBound(int, double) (Java constructor), [439](#)
F08ShiftedRotatedAckleyGlobalOptBound(int, double, String, String) (Java constructor), [439](#)
F09ShiftedRastrigin (Java class), [440](#)
F09ShiftedRastrigin(int, double) (Java constructor), [440](#)
F09ShiftedRastrigin(int, double, String) (Java constructor), [440](#)
F10ShiftedRotatedRastrigin (Java class), [440](#)
F10ShiftedRotatedRastrigin(int, double) (Java constructor), [441](#)
F10ShiftedRotatedRastrigin(int, double, String, String) (Java constructor), [441](#)
F11ShiftedRotatedWeierstrass (Java class), [441](#)
F11ShiftedRotatedWeierstrass(int, double) (Java constructor), [442](#)
F11ShiftedRotatedWeierstrass(int, double, String, String) (Java constructor), [442](#)
F12Schwefel (Java class), [442](#)
F12Schwefel(int, double) (Java constructor), [443](#)
F12Schwefel(int, double, String) (Java constructor), [443](#)
F13ShiftedExpandedGriewankRosenbrock (Java class), [443](#)
F13ShiftedExpandedGriewankRosenbrock(int, double) (Java constructor), [443](#)
F13ShiftedExpandedGriewankRosenbrock(int, double, String) (Java constructor), [443](#)
F14ShiftedRotatedExpandedScaffer (Java class), [444](#)
F14ShiftedRotatedExpandedScaffer(int, double) (Java constructor), [444](#)
F14ShiftedRotatedExpandedScaffer(int, double, String, String) (Java constructor), [444](#)
F15HybridComposition1 (Java class), [444](#)
F15HybridComposition1(int, double) (Java constructor), [445](#)
F15HybridComposition1(int, double, String) (Java constructor), [445](#)
F16RotatedHybridComposition1 (Java class), [445](#)
F16RotatedHybridComposition1(int, double) (Java constructor), [446](#)
F16RotatedHybridComposition1(int, double, String, String) (Java constructor), [446](#)
F17RotatedHybridComposition1Noise (Java class), [446](#)
F17RotatedHybridComposition1Noise(int, double) (Java constructor), [447](#)
F17RotatedHybridComposition1Noise(int, double, String, String) (Java constructor), [447](#)
F18RotatedHybridComposition2 (Java class), [447](#)
F18RotatedHybridComposition2(int, double) (Java constructor), [448](#)

- F18RotatedHybridComposition2(int, double, String, String) (Java constructor), 448
- F19RotatedHybridComposition2NarrowBasinGlobalOpt (Java class), 448
- F19RotatedHybridComposition2NarrowBasinGlobalOpt(int, double) (Java constructor), 449
- F19RotatedHybridComposition2NarrowBasinGlobalOpt(int, double, String, String) (Java constructor), 449
- f1max (Java field), 339
- f1min (Java field), 339
- F2(double, double) (Java method), 430
- F20RotatedHybridComposition2GlobalOptBound (Java class), 449
- F20RotatedHybridComposition2GlobalOptBound(int, double) (Java constructor), 450
- F20RotatedHybridComposition2GlobalOptBound(int, double, String, String) (Java constructor), 450
- F21RotatedHybridComposition3 (Java class), 450
- F21RotatedHybridComposition3(int, double) (Java constructor), 451
- F21RotatedHybridComposition3(int, double, String, String) (Java constructor), 451
- F22RotatedHybridComposition3HighCondNumMatrix (Java class), 451
- F22RotatedHybridComposition3HighCondNumMatrix(int, double) (Java constructor), 452
- F22RotatedHybridComposition3HighCondNumMatrix(int, double, String, String) (Java constructor), 452
- F23NoncontinuousRotatedHybridComposition3 (Java class), 452
- F23NoncontinuousRotatedHybridComposition3(int, double) (Java constructor), 453
- F23NoncontinuousRotatedHybridComposition3(int, double, String, String) (Java constructor), 453
- F24RotatedHybridComposition4 (Java class), 453
- F24RotatedHybridComposition4(int, double) (Java constructor), 454
- F24RotatedHybridComposition4(int, double, String, String) (Java constructor), 454
- F25RotatedHybridComposition4Bound (Java class), 454
- F25RotatedHybridComposition4Bound(int, double) (Java constructor), 455
- F25RotatedHybridComposition4Bound(int, double, String, String) (Java constructor), 455
- f2max (Java field), 339
- f2min (Java field), 339
- F8(double) (Java method), 430
- F8F2(double[]) (Java method), 431
- feasibilityRatio(List) (Java method), 589
- fileName (Java field), 608
- FileOutputContext (Java interface), 606
- fillPopulationWithNewSolutions(List, Problem, int) (Java method), 547
- filter(RandomGenerator, Predicate) (Java method), 664
- finalize() (Java method), 240
- findBestIndicatorFronts(Experiment) (Java method), 598
- findBestSolution(List, Comparator) (Java method), 547
- FindClosestMember() (Java method), 152
- findHighestValues(Front) (Java method), 605
- findHighestValues(List) (Java method), 605
- findHighestValues(Source) (Java method), 604
- findIndexOfBestSolution(List, Comparator) (Java method), 547
- findIndexOfWorstSolution(List, Comparator) (Java method), 547
- findLowestValues(Front) (Java method), 605
- findLowestValues(List) (Java method), 605
- findLowestValues(Source) (Java method), 604
- FindNicheReferencePoint() (Java method), 149
- findPosition(S) (Java method), 111
- findRegion(int) (Java method), 112
- findWorstSolution(Collection, Comparator) (Java method), 548
- Fitness (Java class), 677
- fitness(List, int) (Java method), 84
- FitnessComparator (Java class), 581
- fitnessComparator (Java field), 28
- fitnessFunction(S, double[]) (Java method), 101
- fmax (Java field), 455
- Fonseca (Java class), 317
- Fonseca() (Java constructor), 317
- forArray(BoundedRandomGenerator, T) (Java method), 664
- forCollection(BoundedRandomGenerator, Collection) (Java method), 664
- forEnum(BoundedRandomGenerator, Class) (Java method), 665
- FourBarTruss (Java class), 318
- FourBarTruss() (Java constructor), 318
- frequency (Java field), 28, 113, 116
- fromDoubleToInteger(BoundedRandomGenerator) (Java method), 661
- fromDoubleToInteger(RandomGenerator) (Java method), 661
- Front (Java interface), 609
- FrontExtremeValues (Java class), 605
- FrontNormalizer (Java class), 619
- FrontNormalizer(double[], double[]) (Java constructor), 619
- FrontNormalizer(Front) (Java constructor), 619
- FrontNormalizer(List) (Java constructor), 619
- FrontNormalizerTest (Java class), 620
- FrontUtils (Java class), 622
- FrontUtilsTest (Java class), 615
- fs (Java field), 484
- fullArchiveCrossoverOperator (Java field), 62
- FUNCTION_NAME (Java field), 434–454
- FunctionENS(int) (Java method), 369

FunctionsMahalanobis_Distance_With_Variance(int)
(Java method), 369

FunctionType (Java enum), 54, 102

functionType (Java field), 52, 99, 105

Fyz_ (Java field), 348

G

G_ (Java field), 349

g_ (Java field), 360

GDE3 (Java class), 75

GDE3(DoubleProblem, int, int, DifferentialEvolutionS-
election, DifferentialEvolutionCrossover, Solu-
tionListEvaluator) (Java constructor), 76

GDE3BigDataRunner (Java class), 487

GDE3Builder (Java class), 78

GDE3Builder(DoubleProblem) (Java constructor), 79

GDE3Runner (Java class), 488

GDE3TestIT (Java class), 80

GDominanceComparator (Java class), 582

GDominanceComparator(List) (Java constructor), 582

GeneralizedSpread (Java class), 463

GeneralizedSpread() (Java constructor), 463

GeneralizedSpread(Front) (Java constructor), 463

generalizedSpread(Front, Front) (Java method), 464

GeneralizedSpread(String) (Java constructor), 463

GenerateBoxplotsWithR (Java class), 598

GenerateBoxplotsWithR(Experiment) (Java constructor),
599

generateCoordinates() (Java method), 16, 202

GenerateFriedmanTestTables (Java class), 599

GenerateFriedmanTestTables(Experiment) (Java con-
structor), 599

GenerateLatexTablesWithStatistics (Java class), 600

GenerateLatexTablesWithStatistics(Experiment) (Java
constructor), 600

generatePairsFromSolutionList(List) (Java method), 30

generatePreferenceInformation() (Java method), 566, 570

GenerateReferenceFrontFromFile (Java class), 681

GenerateReferenceParetoFront (Java class), 601

GenerateReferenceParetoFront(Experiment) (Java con-
structor), 601

GenerateReferenceParetoSetAndFrontFromDoubleSolutions
(Java class), 601

GenerateReferenceParetoSetAndFrontFromDoubleSolutions
(Java constructor), 601

generateReferencePoints(List, int, List) (Java method),
153

GenerateWilcoxonTestTablesWithR (Java class), 602

GenerateWilcoxonTestTablesWithR(Experiment) (Java
constructor), 602

GENERATIONAL (Java field), 223

GenerationalDistance (Java class), 464

GenerationalDistance() (Java constructor), 464

GenerationalDistance(Front) (Java constructor), 465

generationalDistance(Front, Front) (Java method), 465

GenerationalDistance(String) (Java constructor), 465

GenerationalDistance(String, double) (Java constructor),
464

GenerationalDistanceTest (Java class), 466

GenerationalGeneticAlgorithm (Java class), 220

GenerationalGeneticAlgorithm(Problem, int, int,
CrossoverOperator, MutationOperator, Se-
lectionOperator, SolutionListEvaluator) (Java
constructor), 220

GenerationalGeneticAlgorithmBinaryEncodingRunner
(Java class), 512

GenerationalGeneticAlgorithmDoubleEncodingRunner
(Java class), 512

GenerationalGeneticAlgorithmTestIT (Java class), 221

GenerationalGeneticAlgorithmTSPRunner (Java class),
512

GenericIndicator (Java class), 466

GenericIndicator() (Java constructor), 467

GenericIndicator(Front) (Java constructor), 467

GenericIndicator(String) (Java constructor), 467

GenericSolutionAttribute (Java class), 678

GenericSolutionAttribute() (Java constructor), 678

GenericSolutionAttribute(Object) (Java constructor), 678

GenericSolutionAttributeTest (Java class), 678

generV(double, double) (Java method), 393

GeneticAlgorithmBuilder (Java class), 221

GeneticAlgorithmBuilder(Problem, CrossoverOperator,
MutationOperator) (Java constructor), 221

GeneticAlgorithmVariant (Java enum), 223

geometryCheck(int, int) (Java method), 371

geometryCheck_ (Java field), 360

get() (Java method), 237, 238, 240, 243, 246, 253

get(int) (Java method), 556, 557, 561

get(Solution) (Java method), 520, 521

get(String) (Java method), 625

get2DHV(List) (Java method), 481

get2DHV(WfgHypervolumeFront) (Java method), 484

getAlgorithm() (Java method), 603

getAlgorithmList() (Java method), 593, 595

getAlgorithmTag() (Java method), 603

getAlpha() (Java method), 262

getArchive() (Java method), 90

getArchiveSize() (Java method), 32, 85, 156, 159, 160,
163, 174

getAttemptsToSettle() (Java method), 16, 204

getAttribute(Object) (Java method), 518, 522, 524, 643

getAttribute(S) (Java method), 130, 150, 674, 678

getAttributeIdentifier() (Java method), 130, 150, 674, 675,
678

getAverageOccupation() (Java method), 541

getBestSolution(S, S, Comparator) (Java method), 554

getBestSolution(S, S, Comparator, BinaryOperator) (Java
method), 554

getBestSolution(S, S, Comparator, RandomGenerator) (Java method), 554
 getBinarySetLength() (Java method), 572
 getBiSections() (Java method), 159, 160, 163
 getBisections() (Java method), 541
 getBitsPerVariable(int) (Java method), 312, 323, 423, 426
 getBorder() (Java method), 42
 getBounds() (Java method), 667
 getC1Max() (Java method), 55, 175
 getC1Min() (Java method), 55, 175
 getC2Max() (Java method), 55, 175
 getC2Min() (Java method), 55, 175
 getCataclysmicMutation() (Java method), 97
 getChangeVelocity1() (Java method), 55, 175
 getChangeVelocity2() (Java method), 55, 175
 getChart(String) (Java method), 573, 575
 getChildGrid() (Java method), 48
 getChildGridNum() (Java method), 48
 getComparator() (Java method), 68, 204, 556, 559–561, 565
 getComputingTime() (Java method), 544
 getConvergenceValue() (Java method), 97
 getCoordinates() (Java method), 16
 getCr() (Java method), 266
 getCrossover() (Java method), 48, 85, 97, 106
 getCrossoverOperator() (Java method), 23, 33, 64, 79, 90, 138, 148, 163, 168, 188, 193, 204, 209, 221
 getCrossoverProbability() (Java method), 262, 268, 269, 271, 273, 274, 277, 280
 getCurrentIteration() (Java method), 289
 getDataDirectory() (Java method), 55, 107
 getDelay() (Java method), 573, 575
 getDelta() (Java method), 285
 getDescription() (Java method), 30, 46, 51, 53, 61, 63, 77, 81, 84, 88, 93, 95, 103, 104, 112, 114, 117, 123, 124, 134, 137, 142, 144–146, 154, 159, 162, 165, 167, 172, 180, 183, 186, 191, 196, 199, 202, 207, 213, 215, 218, 220, 223, 226, 228, 253, 459, 464, 465, 470, 471, 478, 480, 481, 566, 624, 627
 getDimension() (Java method), 642, 646
 getDistance(E, J) (Java method), 590
 getDistance(S, L) (Java method), 591
 getDistance(S, S) (Java method), 590–592
 getDistances() (Java method), 567, 570
 getDistributionIndex() (Java method), 269, 274, 287, 293
 getElement(int, int) (Java method), 371
 getElementsBetweenDiffGreat() (Java method), 371
 getEvaluations() (Java method), 143, 207, 260, 281, 282
 getEvaluator() (Java method), 55, 64, 148, 175, 221
 getExclusiveHV(WfgHypervolumeFront, int) (Java method), 484
 getExperimentBaseDirectory() (Java method), 594, 595
 getExperimentName() (Java method), 594, 595
 getF() (Java method), 266
 getFa() (Java method), 16, 204
 getFbr() (Java method), 16, 204
 getFbs() (Java method), 16, 204
 getFd() (Java method), 16, 204
 getFileName() (Java method), 606, 608
 getFileWriter() (Java method), 606, 608
 getFrontChart() (Java method), 573, 575
 getFronts() (Java method), 151
 getFullArchiveCrossoverOperator() (Java method), 64
 getFunctionType() (Java method), 56, 107
 getG(S, int) (Java method), 42
 getGeneratorName() (Java method), 662
 getGrid() (Java method), 559
 getGridPos(int, double) (Java method), 42
 getGroups(int) (Java method), 371
 getGroupShape(int) (Java method), 371
 getHV(WfgHypervolumeFront) (Java method), 484
 getHypercubes() (Java method), 541
 getImprovementOperator() (Java method), 33
 getInclusiveHV(Point) (Java method), 484
 getIndependentRuns() (Java method), 594, 595
 getIndicatorList() (Java method), 594, 596
 getInitialConvergenceCount() (Java method), 97
 getInstance() (Java method), 662
 getInvertedFront(Front) (Java method), 623
 getInvertedFront(List) (Java method), 548
 getIterations() (Java method), 172
 getK() (Java method), 48, 266
 getLambda() (Java method), 213, 217
 getLeaders() (Java method), 175
 getLessContributorHV(List) (Java method), 484
 getLocalBest() (Java method), 226, 228
 getLocationDensity(int) (Java method), 541
 getLowerBound(int) (Java method), 309–311, 313–315, 516, 517, 524, 528–530, 532
 getM() (Java method), 16, 205
 getMatrixWidthBand() (Java method), 371
 getMax(List) (Java method), 125
 getMaxAge() (Java method), 56
 getMaxEvaluation() (Java method), 97
 getMaxEvaluations() (Java method), 33, 48, 64, 79, 85, 90, 107, 159, 160, 163, 165, 166, 188, 205, 209, 213, 217, 222
 getMaximumNumberOfReplacedSolutions() (Java method), 107
 getMaximumValues(Front) (Java method), 623
 getMaxIterations() (Java method), 56, 138, 148, 156, 168, 172, 175, 193, 290, 571
 getMaxObjective(int) (Java method), 129
 getMaxPopulationSize() (Java method), 23, 77, 93, 123
 getMaxSize() (Java method), 556, 557
 getMeasureKeys() (Java method), 236, 255

- getMeasureManager() (Java method), 61, 143, 180, 183, 199, 235
- getMethod() (Java method), 667
- getMinimumValues(Front) (Java method), 623
- getMostPopulatedHypercube() (Java method), 541
- getMu() (Java method), 217
- getMutation() (Java method), 85, 107, 175, 217
- getMutationOperator() (Java method), 20, 23, 33, 64, 90, 138, 148, 159, 161, 163, 168, 175, 188, 193, 205, 222
- getMutationProbability() (Java method), 283, 286, 287, 290, 291, 293, 296, 297
- getMxyMax(int, int) (Java method), 371
- getMxyMin(int, int) (Java method), 371
- getMxzMax(int, int) (Java method), 372
- getMxzMin(int, int) (Java method), 372
- getN() (Java method), 16, 205
- getNadirPoint() (Java method), 120
- getName() (Java method), 30, 46, 51, 53, 56, 63, 77, 81, 84, 88, 93, 95, 103, 104, 112, 114, 117, 123, 125, 134, 137, 143–146, 154, 159, 162, 165, 167, 172, 180, 183, 186, 191, 196, 199, 203, 207, 213, 215, 218, 220, 223, 226, 228, 253, 312, 313, 459, 461, 464, 466, 468, 470, 471, 474, 475, 478, 567, 573, 576, 624, 627, 663, 666, 669, 671–673
- getNeighborhood() (Java method), 639
- getNeighborhoodSelectionProbability() (Java method), 48, 107
- getNeighbors(List, int) (Java method), 629, 634, 639, 640
- getNeighborSize() (Java method), 107, 638
- getNewGenerationSelection() (Java method), 97
- getNode(int, int) (Java method), 372
- getNodeRestrict(int, int) (Java method), 372
- getNonDominatedSolutions(List) (Java method), 77, 134, 137, 147, 196
- getNondominatedSolutions(List) (Java method), 548
- getNonUniformMutation() (Java method), 156
- getNormalizedFront(List, List, List) (Java method), 548
- getNumberOfBits() (Java method), 309, 516, 528
- getNumberOfBits(int) (Java method), 309, 312, 516, 526
- getNumberOfConstraints() (Java method), 309, 312, 313
- getNumberOfConstraintsGeometric() (Java method), 375
- getNumberOfConstraintsNodes() (Java method), 372
- getNumberOfCores() (Java method), 594, 596
- getNumberOfDoubleVariables() (Java method), 310, 314, 516, 517, 528, 530
- getNumberOfElements() (Java method), 372
- getNumberOfGeneratedChildren() (Java method), 260, 263, 266, 268, 270, 271, 273, 275, 278, 280
- getNumberOfGroupElements() (Java method), 375
- getNumberOfImprovements() (Java method), 260, 281, 282
- getNumberOfIntegerVariables() (Java method), 310, 314, 517, 530
- getNumberOfNodes() (Java method), 372
- getNumberOfNodesRestricts() (Java method), 372
- getNumberOfNonComparableSolutions() (Java method), 260, 281, 282
- getNumberOfObjectives() (Java method), 151, 312, 313, 518, 522, 524, 644
- getNumberOfPoints() (Java method), 483, 609, 611
- getNumberOfRequiredParents() (Java method), 260, 263, 266, 268, 270, 271, 273, 275, 278, 280
- getNumberOfSolutionsToSelect() (Java method), 305, 307
- getNumberOfSubfronts() (Java method), 82, 131, 171, 200, 674, 677
- getNumberOfSubranges() (Java method), 33
- getNumberOfThreads() (Java method), 48, 107, 593
- getNumberOfVariables() (Java method), 312, 314, 518, 522, 524, 644
- getNumberOfWeightVectors() (Java method), 639
- getNumberOfWeigthHypothesis() (Java method), 372
- getNumberOfWeigthsElements() (Java method), 372
- getNumberOfWeigthsNodes() (Java method), 372
- getNxxMax(int, int) (Java method), 373
- getNxxMin(int, int) (Java method), 373
- getObjective(int) (Java method), 518, 522, 524, 644
- getObjectiveArrayFromSolutionList(List, int) (Java method), 548
- getObjectives() (Java method), 518, 521, 522, 525, 644
- getOffset() (Java method), 189, 468, 480, 481
- getOldStrainMax(int, int) (Java method), 373
- getOldStrainMin(int, int) (Java method), 373
- getOmegaMax(int, int) (Java method), 373
- getOperationType() (Java method), 64
- getOrder(S) (Java method), 42
- getOutputParetoFrontFileName() (Java method), 594, 596
- getOutputParetoSetFileName() (Java method), 594, 596
- getParentSelection() (Java method), 97
- getPd() (Java method), 16, 205
- getPermutationLength() (Java method), 311, 320, 428
- getPerturbation() (Java method), 290, 297
- getPoint(int) (Java method), 483, 609, 611
- getPointDimensions() (Java method), 609, 611
- getPopulation() (Java method), 16, 21, 26
- getPopulationSize() (Java method), 17, 26, 33, 48, 65, 79, 85, 90, 97, 107, 138, 148, 164, 168, 189, 193, 197, 209, 222
- getPos(int, int, int) (Java method), 42
- getPreservedPopulation() (Java method), 97
- getProblem() (Java method), 21, 56, 91, 98, 138, 140, 148, 161, 164, 168, 176, 189, 193, 205, 209, 210, 222, 571, 604
- getProblemList() (Java method), 594, 596
- getProblemTag() (Java method), 603

- getPullMeasure(Object) (Java method), 237, 255
- getPushMeasure(Object) (Java method), 237, 255
- getR1Max() (Java method), 56, 176
- getR1Min() (Java method), 56, 176
- getR2Max() (Java method), 56, 176
- getR2Min() (Java method), 56, 176
- getRandomGenerator() (Java method), 662
- getRandomValue() (Java method), 665
- getRandomValue(Value, Value) (Java method), 661
- getRank(S, int) (Java method), 42
- getReferenceFront() (Java method), 604
- getReferenceFrontDirectory() (Java method), 594, 596
- getReferenceParetoFront() (Java method), 603
- getReferencePoint() (Java method), 120
- getReferencePoints() (Java method), 151, 567, 570
- getRefSet1Size() (Java method), 33
- getRefSet2Size() (Java method), 33
- getResult() (Java method), 13, 17, 21, 24, 26, 31, 42, 51, 53, 63, 77, 84, 88, 94, 95, 101, 117, 120, 134, 137, 147, 154, 159, 162, 165, 167, 172, 183, 186, 191, 197, 203, 207, 213, 215, 218, 220, 224, 226, 228, 567, 668
- getResultPopulationSize() (Java method), 49, 107
- getRho() (Java method), 17, 205
- getRunId() (Java method), 603
- getScalarization() (Java method), 65
- getSeed() (Java method), 662, 663, 667, 669, 671–673
- getSelection() (Java method), 85
- getSelectionOperator() (Java method), 23, 65, 79, 91, 138, 148, 168, 189, 194, 205, 209, 222
- getSeparator() (Java method), 606, 608
- getSize() (Java method), 128, 170, 680
- getSolutionList() (Java method), 556, 557, 562
- getSolutionListEvaluator() (Java method), 91, 138, 164, 169, 194, 209
- getSolutionsToSelect() (Java method), 151
- getStrainAdmissibleCut() (Java method), 373
- getStrainCutMax(int, int) (Java method), 373
- getStrainj(int, int, int) (Java method), 374
- getStrainMax(int, int) (Java method), 373
- getStrainMin(int, int) (Java method), 373
- getStrainMxyMax(int, int) (Java method), 373
- getStrainMxyMin(int, int) (Java method), 373
- getStrainMxzMax(int, int) (Java method), 374
- getStrainMxzMin(int, int) (Java method), 374
- getStrainNxxMax(int, int) (Java method), 374
- getStrainNxxMin(int, int) (Java method), 374
- getStrainResidualCut(int) (Java method), 374
- getStrainResidualMax(int) (Java method), 374
- getStrainResidualMin(int) (Java method), 374
- getSubfront(int) (Java method), 82, 131, 171, 200, 674, 677
- getSubsetOfEvenlyDistributedSolutions(List, int) (Java method), 118
- getSwarm() (Java method), 24, 53
- getSwarmSize() (Java method), 56, 156, 172, 176
- getSwarmSpeedMatrix() (Java method), 226, 229
- getT() (Java method), 49
- getTag() (Java method), 604
- getTotalNumberOfBits() (Java method), 309, 312, 516, 527
- getUniformMutation() (Java method), 156
- getUpperBound(int) (Java method), 310, 311, 313–315, 516, 517, 525, 529, 530, 532
- getUtilityFunctions() (Java method), 123, 131, 200
- getValue(int) (Java method), 642, 646
- getValues() (Java method), 642, 646
- getVarChart() (Java method), 573, 576
- getVariablePosition(int) (Java method), 374
- getVariables() (Java method), 519
- getVariableValue(int) (Java method), 518, 522, 525, 644
- getVariableValueString(int) (Java method), 519, 525, 527, 529–532, 644
- getVariant() (Java method), 222, 266
- getVectorSize() (Java method), 128
- getWeightElement(int, int) (Java method), 374
- getWeightElementItself(int, int) (Java method), 374
- getWeightMax() (Java method), 56, 176
- getWeightMin() (Java method), 57, 176
- getWeightNode(int, int) (Java method), 375
- getWeightVector() (Java method), 639
- getWeightVector(int) (Java method), 128
- getWeightVectorSize() (Java method), 639
- getX() (Java method), 18
- getY() (Java method), 18
- GLT1 (Java class), 376
- GLT1() (Java constructor), 376
- GLT1(int) (Java constructor), 376
- GLT2 (Java class), 377
- GLT2() (Java constructor), 377
- GLT2(int) (Java constructor), 377
- GLT3 (Java class), 377
- GLT3() (Java constructor), 378
- GLT3(int) (Java constructor), 378
- GLT4 (Java class), 378
- GLT4() (Java constructor), 378
- GLT4(int) (Java constructor), 378
- GLT5 (Java class), 379
- GLT5() (Java constructor), 379
- GLT5(int) (Java constructor), 379
- GLT6 (Java class), 379
- GLT6() (Java constructor), 380
- GLT6(int) (Java constructor), 380
- GNSGAII Measures With Charts Runner (Java class), 488
- Golinski (Java class), 318
- Golinski() (Java constructor), 319
- GravitationalAxis_ (Java field), 349
- gridDetail_ (Java field), 38

[gridDetalSum_ \(Java field\)](#), 38
[gridSystemSetup\(\) \(Java method\)](#), 42
[gridSystemSetup3\(\) \(Java method\)](#), 42
[Griewank \(Java class\)](#), 425
[Griewank\(double\[\]\) \(Java method\)](#), 399
[griewank\(double\[\]\) \(Java method\)](#), 431
[Griewank\(Integer\) \(Java constructor\)](#), 425
[group2\(\) \(Java method\)](#), 42
[group3\(\) \(Java method\)](#), 42
[GROUP_ \(Java field\)](#), 348
[Groups_ \(Java field\)](#), 349
[guassianElimination\(List, List\) \(Java method\)](#), 150
[GWASFGA \(Java class\)](#), 80
[GWASFGA\(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, double\) \(Java constructor\)](#), 81
[GWASFGA\(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, double, String\) \(Java constructor\)](#), 80
[GWASFGARanking \(Java class\)](#), 81
[GWASFGARanking\(AbstractUtilityFunctionsSet, AbstractUtilityFunctionsSet\) \(Java constructor\)](#), 81
[GWASGFGARunner \(Java class\)](#), 489
[gX_ \(Java field\)](#), 360
[gY_ \(Java field\)](#), 360
[gZ_ \(Java field\)](#), 360

H

[H_DOUBLE \(Java field\)](#), 349
[H_SINGLE \(Java field\)](#), 349
[hashCode\(\) \(Java method\)](#), 523, 525, 611, 644, 646
[HasPotentialMember\(\) \(Java method\)](#), 152
[HCJob \(Java class\)](#), 455
[HCJob\(\) \(Java constructor\)](#), 456
[history \(Java field\)](#), 124
[HOLE_CIRCLE \(Java field\)](#), 349
[HOLE_RECTANGLE \(Java field\)](#), 349
[HUXCrossover \(Java class\)](#), 267
[HUXCrossover\(double\) \(Java constructor\)](#), 267
[HUXCrossover\(double, RandomGenerator\) \(Java constructor\)](#), 267
[HUXCrossoverTest \(Java class\)](#), 268
[hybrid_composition\(double\[\], HCJob\) \(Java method\)](#), 431
[Hypervolume \(Java class\)](#), 467
[hypervolume \(Java field\)](#), 560
[Hypervolume\(\) \(Java constructor\)](#), 467
[Hypervolume\(Front\) \(Java constructor\)](#), 468
[Hypervolume\(String\) \(Java constructor\)](#), 467
[HypervolumeArchive \(Java class\)](#), 560
[HypervolumeArchive\(int, Hypervolume\) \(Java constructor\)](#), 560

[HypervolumeArchiveWithReferencePoint \(Java class\)](#), 565
[HypervolumeArchiveWithReferencePoint\(int, List\) \(Java constructor\)](#), 565
[HypervolumeContributionAttribute \(Java class\)](#), 679
[HypervolumeContributionComparator \(Java class\)](#), 582
[hypervolumeImplementation \(Java field\)](#), 187
[HypervolumeTest \(Java class\)](#), 468
[hypervolumeValue \(Java field\)](#), 60, 142

I

[i_ \(Java field\)](#), 360
[I_DOUBLE \(Java field\)](#), 349
[I_SINGLE \(Java field\)](#), 349
[IBEA \(Java class\)](#), 82
[IBEA\(Problem, int, int, int, SelectionOperator, CrossoverOperator, MutationOperator\) \(Java constructor\)](#), 83
[IBEABuilder \(Java class\)](#), 85
[IBEABuilder\(Problem\) \(Java constructor\)](#), 85
[IBEARunner \(Java class\)](#), 489
[idealObjectiveVector \(Java field\)](#), 569
[IdealPoint \(Java class\)](#), 648
[idealPoint \(Java field\)](#), 39, 99
[IdealPoint\(int\) \(Java constructor\)](#), 649
[IdealPointTest \(Java class\)](#), 649
[idleTestToCoverTheUnusedMethods\(\) \(Java method\)](#), 654
[if_infeasible\(double\[\]\) \(Java method\)](#), 393
[if_inside_polygon\(double\[\], double\[\]\[\]\) \(Java method\)](#), 393
[improvement\(DoubleSolution\) \(Java method\)](#), 31
[improvement\(S\) \(Java method\)](#), 26
[improvementOperator \(Java field\)](#), 32
[increment\(\) \(Java method\)](#), 240
[increment\(long\) \(Java method\)](#), 240
[indArray \(Java field\)](#), 52, 99
[INDEX_ \(Java field\)](#), 349
[indexOfRelevantObjectiveFunctions \(Java field\)](#), 566
[indicatorValues \(Java field\)](#), 83
[individualObjRankSort\(\) \(Java method\)](#), 43
[initChart\(\) \(Java method\)](#), 574, 576
[initialCDGAttributes\(S\) \(Java method\)](#), 43
[initialConvergenceCount \(Java field\)](#), 96
[initialGridDetal\(\) \(Java method\)](#), 43
[initializationPhase\(\) \(Java method\)](#), 26
[initializeIdealPoint\(\) \(Java method\)](#), 43
[initializeLeader\(List\) \(Java method\)](#), 24, 154, 172, 183, 226, 229
[initializeLeaders\(List\) \(Java method\)](#), 53
[initializeNadirPoint\(\) \(Java method\)](#), 43
[initializeNeighborhood\(\) \(Java method\)](#), 43, 101
[initializeNeighborhoodGrid\(\) \(Java method\)](#), 43
[initializeObjectiveValues\(\) \(Java method\)](#), 523

- initializeParticlesMemory(List) (Java method), 24, 53, 154, 172, 183, 226, 229
- initializePopulation() (Java method), 46, 104, 105, 114, 117
- initializeSubP2() (Java method), 43
- initializeSubP3() (Java method), 43
- initializeUniformlyInTwoDimensions(double, int) (Java method), 200
- initializeUniformWeight() (Java method), 102
- initializeVelocity(List) (Java method), 24, 54, 154, 173, 183, 226, 229
- initPopulation() (Java method), 112
- initProgress() (Java method), 17, 21, 24, 51, 53, 61, 63, 77, 88, 94, 121, 125, 134, 143, 144, 147, 154, 159, 162, 167, 172, 180, 183, 186, 191, 199, 203, 208, 213, 216, 218, 220, 224, 226, 229, 567, 570
- innerproduct(double[], double[]) (Java method), 112, 117
- IntegerDoubleProblem (Java interface), 310
- IntegerDoubleSolution (Java interface), 517
- IntegerPolynomialMutation (Java class), 286
- IntegerPolynomialMutation() (Java constructor), 286
- IntegerPolynomialMutation(double, double) (Java constructor), 286
- IntegerPolynomialMutation(double, double, RepairDoubleSolution) (Java constructor), 286
- IntegerPolynomialMutation(double, double, RepairDoubleSolution, RandomGenerator) (Java constructor), 287
- IntegerPolynomialMutation(IntegerProblem, double) (Java constructor), 286
- IntegerPolynomialMutationTest (Java class), 287
- IntegerPolynomialMutationWorkingTest (Java class), 682
- IntegerProblem (Java interface), 311
- IntegerSBXCrossover (Java class), 269
- IntegerSBXCrossover(double, double) (Java constructor), 269
- IntegerSBXCrossover(double, double, RandomGenerator) (Java constructor), 269
- IntegerSBXCrossoverTest (Java class), 270
- IntegerSBXCrossoverWorkingTest (Java class), 682
- IntegerSolution (Java interface), 517
- InteractiveAlgorithm (Java interface), 13
- Interpolation_I_Single_ey_func_Y_(double) (Java method), 370
- Interpolation_I_Single_ez_func_Y_(double) (Java method), 370
- Interpolation_I_Single_Y_func_Area_(double) (Java method), 369
- Interpolation_I_Single_Y_func_Wxy_(double) (Java method), 369
- Interpolation_I_Single_Y_func_Wxz_(double) (Java method), 370
- Interpolation_I_Single_Z_func_Y_(double) (Java method), 370
- intersection(double[], double[]) (Java method), 394
- invert(double[][], boolean) (Java method), 201
- InvertedGenerationalDistance (Java class), 469
- InvertedGenerationalDistance() (Java constructor), 469
- InvertedGenerationalDistance(Front) (Java constructor), 469
- invertedGenerationalDistance(Front, Front) (Java method), 470
- InvertedGenerationalDistance(String) (Java constructor), 469
- InvertedGenerationalDistance(String, double) (Java constructor), 469
- InvertedGenerationalDistancePlus (Java class), 470
- InvertedGenerationalDistancePlus() (Java constructor), 471
- InvertedGenerationalDistancePlus(Front) (Java constructor), 471
- invertedGenerationalDistancePlus(Front, Front) (Java method), 472
- InvertedGenerationalDistancePlus(String) (Java constructor), 471
- InvertedGenerationalDistancePlusTest (Java class), 472
- isEmpty() (Java method), 625
- isFull() (Java method), 69
- isInner(S) (Java method), 43
- IsInterfaceException (Java class), 535
- IsInterfaceException(Class) (Java constructor), 536
- isNormalizeObjectives() (Java method), 65
- isSolutionDominatedBySolutionList(S, List) (Java method), 549
- isStoppingConditionReached() (Java method), 17, 21, 25, 26, 31, 51, 54, 61, 63, 77, 88, 94, 121, 135, 143, 144, 147, 154, 159, 162, 167, 173, 180, 183, 187, 191, 199, 203, 208, 214, 216, 218, 220, 224, 226, 229, 567, 570
- isTheLowerTheIndicatorValueTheBetter() (Java method), 459, 464, 466–468, 470, 472, 478
- isUnMarked(int) (Java method), 129
- It_ (Java field), 349
- iterations (Java field), 52, 60, 119, 146, 179, 181, 191, 198
- iterator() (Java method), 625
- Iw_ (Java field), 350
- Iy_ (Java field), 350
- Iz_ (Java field), 350
- ## J
- j_ (Java field), 360
- JavaRandomGenerator (Java class), 670
- JavaRandomGenerator() (Java constructor), 671
- JavaRandomGenerator(long) (Java constructor), 671
- JMetalException (Java class), 545
- JMetalException(Exception) (Java constructor), 545
- JMetalException(String) (Java constructor), 545

JMetalException(String, Exception) (Java constructor), 545
 JMetalLogger (Java class), 545
 JMetalRandom (Java class), 662
 join(Archive) (Java method), 558, 562
 jointPopulation (Java field), 99

K

k (Java field), 406
 K10 (Java field), 394
 K11 (Java field), 395
 K12 (Java field), 396
 k_ (Java field), 39, 47
 KGii (Java field), 350
 KGij (Java field), 350
 KGji (Java field), 350
 KGjj (Java field), 350
 Kii (Java field), 350
 KiiSOG (Java field), 350
 Kij (Java field), 350
 KijSOG (Java field), 350
 Kji (Java field), 351
 KjiSOG (Java field), 351
 Kjj (Java field), 351
 KjjSOG (Java field), 351
 Kmax (Java field), 442
 KNearestNeighborhood (Java class), 633
 KNearestNeighborhood(int) (Java constructor), 634
 KNearestNeighborhood(int, Distance) (Java constructor), 634
 KNearestNeighborhoodTest (Java class), 634
 Kursawe (Java class), 319
 Kursawe() (Java constructor), 319
 Kursawe(Integer) (Java constructor), 319

L

l (Java field), 406
 L13 (Java class), 635
 L13(int, int) (Java constructor), 635
 L13Test (Java class), 635
 L25 (Java class), 637
 L25(int, int) (Java constructor), 637
 L41 (Java class), 637
 L41(int, int) (Java constructor), 637
 L5 (Java class), 637
 L5(int, int) (Java constructor), 637
 L5Test (Java class), 637
 L_ (Java field), 351
 L_DOUBLE (Java field), 351
 L_SINGLE (Java field), 351
 lambda (Java field), 52, 100, 455
 LARGEST_DIFFERENCE (Java field), 69
 larvaeSettlementPhase(List, List, List) (Java method), 17, 203

LastEvaluationMeasure (Java class), 244
 LastEvaluationMeasure() (Java constructor), 244
 LastEvaluationMeasureTest (Java class), 245
 lBuckling (Java field), 360
 leaders (Java field), 174, 182
 LexicographicalPointComparator (Java class), 657
 LexicographicalPointComparatorTest (Java class), 649
 lexicographicSort() (Java method), 43
 Li_ (Java field), 351
 line_of_twoP(double[], double[]) (Java method), 394
 linear(float[], int) (Java method), 404
 linearTransformationMatrix (Java field), 455
 lines_of_polygon(double[][]) (Java method), 394
 link(PushMeasure) (Java method), 241
 ListenerTimeMeasure (Java class), 245
 ListenerTimeMeasureTest (Java class), 247
 Lj_ (Java field), 351
 lLoadsOwnWeight (Java field), 361
 loadColumnVector(BufferedReader, int, double[]) (Java method), 431
 loadColumnVectorFromFile(String, int, double[]) (Java method), 431
 loader (Java field), 429
 loadMatrix(BufferedReader, int, int, double[][]) (Java method), 431
 loadMatrixFromFile(String, int, int, double[][]) (Java method), 431
 loadNMatrixFromFile(String, int, int, int, double[][][]) (Java method), 431
 loadProblem(String) (Java method), 546
 loadRowVector(BufferedReader, int, double[]) (Java method), 432
 loadRowVectorFromFile(String, int, double[]) (Java method), 432
 loadTestDataFromFile(String, int, int, double[][][], double[]) (Java method), 432
 loadWeightsFromFile(String) (Java method), 128
 localSearch (Java field), 28, 35
 localSearchOperator (Java field), 35
 LocalSearchOperator (Java interface), 260
 LocalSearchRunner (Java class), 513
 location (Java field), 87
 location(S) (Java method), 541
 LocationAttribute (Java class), 679
 LocationAttribute(List) (Java constructor), 679
 logger (Java field), 546
 lower (Java field), 668
 LOWERLIMIT (Java field), 327
 lSecondOrderGeometric (Java field), 361
 ltype (Java field), 380
 LZ09 (Java class), 380
 LZ09(int, int, int, int, int) (Java constructor), 381
 LZ09F1 (Java class), 381
 LZ09F1() (Java constructor), 382

LZ09F1(Integer, Integer, Integer) (Java constructor), 382
 LZ09F2 (Java class), 382
 LZ09F2() (Java constructor), 382
 LZ09F2(Integer, Integer, Integer) (Java constructor), 382
 LZ09F3 (Java class), 382
 LZ09F3() (Java constructor), 383
 LZ09F3(Integer, Integer, Integer) (Java constructor), 383
 LZ09F4 (Java class), 383
 LZ09F4() (Java constructor), 383
 LZ09F4(Integer, Integer, Integer) (Java constructor), 383
 LZ09F5 (Java class), 383
 LZ09F5() (Java constructor), 384
 LZ09F5(Integer, Integer, Integer) (Java constructor), 384
 LZ09F6 (Java class), 384
 LZ09F6() (Java constructor), 384
 LZ09F6(Integer, Integer, Integer) (Java constructor), 384
 LZ09F7 (Java class), 384
 LZ09F7() (Java constructor), 385
 LZ09F7(Integer, Integer, Integer) (Java constructor), 385
 LZ09F8 (Java class), 385
 LZ09F8() (Java constructor), 385
 LZ09F8(Integer, Integer, Integer) (Java constructor), 385
 LZ09F9 (Java class), 385
 LZ09F9() (Java constructor), 386
 LZ09F9(Integer, Integer, Integer) (Java constructor), 386
 lZ_ (Java field), 361

M

m (Java field), 406
 M9 (Java field), 392
 MaF01 (Java class), 386
 MaF01() (Java constructor), 386
 MaF01(Integer, Integer) (Java constructor), 386
 MaF02 (Java class), 387
 MaF02() (Java constructor), 387
 MaF02(Integer, Integer) (Java constructor), 387
 MaF03 (Java class), 388
 MaF03() (Java constructor), 388
 MaF03(Integer, Integer) (Java constructor), 388
 MaF04 (Java class), 388
 MaF04() (Java constructor), 388
 MaF04(Integer, Integer) (Java constructor), 389
 MaF05 (Java class), 389
 MaF05() (Java constructor), 389
 MaF05(Integer, Integer) (Java constructor), 389
 MaF06 (Java class), 390
 MaF06() (Java constructor), 390
 MaF06(Integer, Integer) (Java constructor), 390
 MaF07 (Java class), 390
 MaF07() (Java constructor), 391
 MaF07(Integer, Integer) (Java constructor), 391
 MaF08 (Java class), 391
 MaF08() (Java constructor), 391
 MaF08(Integer, Integer) (Java constructor), 392

MaF09 (Java class), 392
 MaF09() (Java constructor), 393
 MaF09(Integer, Integer) (Java constructor), 393
 MaF10 (Java class), 394
 MaF10() (Java constructor), 394
 MaF10(Integer, Integer) (Java constructor), 394
 MaF11 (Java class), 395
 MaF11() (Java constructor), 395
 MaF11(Integer, Integer) (Java constructor), 395
 MaF12 (Java class), 396
 MaF12() (Java constructor), 396
 MaF12(Integer, Integer) (Java constructor), 396
 MaF13 (Java class), 396
 MaF13() (Java constructor), 396
 MaF13(Integer, Integer) (Java constructor), 397
 MaF14 (Java class), 397
 MaF14() (Java constructor), 397
 MaF14(Integer, Integer) (Java constructor), 397
 MaF15 (Java class), 398
 MaF15() (Java constructor), 398
 MaF15(Integer, Integer) (Java constructor), 398
 main(String[]) (Java method), 230–235, 432, 458, 485–515, 562, 681–684
 makeDominatedBit(WfgHypervolumeFront, int) (Java method), 485
 mark(int) (Java method), 129
 MarkAttribute (Java class), 37
 marked (Java field), 28
 matingSelection(int, int) (Java method), 112
 matingSelection(int, int, NeighborType) (Java method), 44, 102
 MatrixStiffness(int) (Java method), 370
 MatrixStiffness_ (Java field), 351
 matrixWidthBand_ (Java field), 361
 MAX_COLUMN (Java field), 351
 MAX_LENGTH (Java field), 128
 MAX_SUPPORT_DIM (Java field), 429
 maxAge (Java field), 52
 maxEvaluations (Java field), 29, 39, 47, 50, 62, 75, 78, 83, 87, 89, 96, 100, 105, 134, 135, 158, 186, 187, 195, 569
 maximizing (Java field), 484
 maximumNumberOfReplacedSolutions (Java field), 100, 105
 maxIndicatorValue (Java field), 83
 maxinter9 (Java field), 392
 maxIterations (Java field), 52, 119, 146, 182, 191, 192
 maxPopulationSize (Java field), 22
 maxs (Java field), 124
 maxSize (Java field), 557
 mBias (Java field), 457
 mDimension (Java field), 457
 mean() (Java method), 129
 meanOverallViolation(List) (Java method), 589

- Measurable (Java interface), 235
- Measure (Java interface), 236
- MeasureFactory (Java class), 249
- MeasureFactoryTest (Java class), 250
- measureGenerated(Value) (Java method), 236
- MeasureListener (Java interface), 236
- measureManager (Java field), 60, 142, 179, 182, 198
- MeasureManager (Java interface), 236
- Measures (Java field), 59, 92, 139, 169, 178
- MemberSize() (Java method), 152
- MersenneTwisterGenerator (Java class), 671
- MersenneTwisterGenerator() (Java constructor), 672
- MersenneTwisterGenerator(long) (Java constructor), 672
- mFuncName (Java field), 457
- minFastSort(double[], int[], int, int) (Java method), 119
- mixed(float[], int, float) (Java method), 404
- MOCeCell (Java class), 86
- MOCeCell (Java field), 92
- MOCeCell(Problem, int, int, BoundedArchive, Neighborhood, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 87
- MOCeCellBuilder (Java class), 89
- MOCeCellBuilder(Problem, CrossoverOperator, MutationOperator) (Java constructor), 90
- MOCeCellHVRRunner (Java class), 490
- MOCeCellIT (Java class), 92
- MOCeCellRunner (Java class), 491
- MOCeCellVariant (Java enum), 91
- MOCHC (Java class), 93
- MOCHC(BinaryProblem, int, int, int, double, double, CrossoverOperator, MutationOperator, SelectionOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 93
- MOCHC45 (Java class), 94
- MOCHC45(BinaryProblem, int, int, int, double, double, CrossoverOperator, MutationOperator, SelectionOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 95
- MOCHC45Runner (Java class), 490
- MOCHCBuilder (Java class), 95
- MOCHCBuilder(BinaryProblem) (Java constructor), 96
- MOCHCRunner (Java class), 490
- MOEAD (Java class), 104
- MOEAD (Java field), 109
- MOEAD(Problem, int, int, int, MutationOperator, CrossoverOperator, FunctionType, String, double, int, int) (Java constructor), 104
- MOEADBuilder (Java class), 105
- MOEADBuilder(Problem, Variant) (Java constructor), 106
- MOEADD (Java class), 109
- MOEADD (Java field), 109
- MOEADD(Problem, int, int, int, CrossoverOperator, MutationOperator, AbstractMOEAD.FunctionType, String, double, int, int) (Java constructor), 110
- MOEADDRA (Java class), 113
- MOEADDRA (Java field), 109
- MOEADDRA(Problem, int, int, int, MutationOperator, CrossoverOperator, FunctionType, String, double, int, int) (Java constructor), 114
- MOEADDRAIT (Java class), 115
- MOEADDRunner (Java class), 491
- MOEADIT (Java class), 115
- MOEADRunner (Java class), 491
- MOEADSTM (Java class), 116
- MOEADSTM (Java field), 109
- MOEADSTM(Problem, int, int, int, MutationOperator, CrossoverOperator, FunctionType, String, double, int, int) (Java constructor), 116
- MOEADSTMRunner (Java class), 492
- MOEADUtils (Java class), 118
- moeadVariant (Java field), 105
- MOMBI (Java class), 122
- MOMBI(Problem, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, String) (Java constructor), 122
- MOMBI2 (Java class), 123
- MOMBI2(Problem, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, String) (Java constructor), 124
- MOMBI2History (Java class), 128
- MOMBI2History(int) (Java constructor), 129
- MOMBI2IT (Java class), 133
- MOMBI2Runner (Java class), 492
- MOMBIRunner (Java class), 493
- MOP1 (Java class), 399
- MOP1() (Java constructor), 399
- MOP1(Integer) (Java constructor), 399
- MOP2 (Java class), 400
- MOP2() (Java constructor), 400
- MOP2(Integer) (Java constructor), 400
- MOP3 (Java class), 400
- MOP3() (Java constructor), 401
- MOP3(Integer) (Java constructor), 401
- MOP4 (Java class), 401
- MOP4() (Java constructor), 401
- MOP4(Integer) (Java constructor), 401
- MOP5 (Java class), 402
- MOP5() (Java constructor), 402
- MOP5(Integer) (Java constructor), 402
- MOP6 (Java class), 402
- MOP6() (Java constructor), 403
- MOP6(Integer) (Java constructor), 403
- MOP7 (Java class), 403
- MOP7() (Java constructor), 403
- MOP7(Integer) (Java constructor), 403

- MultiobjectiveTSP (Java class), 320
- MultiobjectiveTSP(String, String) (Java constructor), 320
- multiplyWithoutAMP(List, List) (Java method), 340
- MultithreadedSolutionListEvaluator (Java class), 592
- MultithreadedSolutionListEvaluator(int, Problem) (Java constructor), 592
- mutation (Java field), 35, 36, 92, 105
- mutationOperator (Java field), 14, 19, 22, 83, 89, 100, 135, 174, 188, 192
- MutationOperator (Java interface), 260
- MxyMax_ (Java field), 352
- MxyMin_ (Java field), 352
- MxzMax_ (Java field), 352
- MxzMin_ (Java field), 352
- myRound(double) (Java method), 432
- myXRound(double) (Java method), 432
- myXRound(double, double) (Java method), 432
- N**
- n (Java field), 331, 332
- nadirObjectiveVector (Java field), 569
- NadirPoint (Java class), 651
- nadirPoint (Java field), 39, 100, 120
- NadirPoint(int) (Java constructor), 651
- NadirPointTest (Java class), 651
- name (Java field), 474
- name() (Java method), 457
- NaryRandomSelection (Java class), 301
- NaryRandomSelection() (Java constructor), 301
- NaryRandomSelection(int) (Java constructor), 301
- NaryRandomSelectionTest (Java class), 301
- NaryTournamentSelection (Java class), 303
- NaryTournamentSelection() (Java constructor), 303
- NaryTournamentSelection(int, Comparator) (Java constructor), 303
- NaryTournamentSelectionTest (Java class), 303
- NASH (Java field), 74
- nash(List) (Java method), 71
- nash(List, double[]) (Java method), 71
- needToCompare(S, S) (Java method), 589
- NEIGHBOR (Java field), 45, 103
- neighborhood (Java field), 39, 87, 89, 100
- Neighborhood (Java interface), 628
- neighborhoodNum (Java field), 39
- neighborhoodSelectionProbability (Java field), 39, 47, 100, 106
- neighborSize (Java field), 100, 106
- NeighborType (Java enum), 45, 103
- newGenerationSelection (Java field), 96
- newMeanStandardDeviation(List) (Java method), 340
- nextDouble() (Java method), 662, 663, 667, 669, 671–673
- nextDouble(double, double) (Java method), 663, 667, 669, 671–673
- nextInt(int, int) (Java method), 663, 667, 670–673
- nextPoint(double, double[], double) (Java method), 392
- NIntegerMin (Java class), 425
- NIntegerMin() (Java constructor), 425
- NIntegerMin(int, int, int, int) (Java constructor), 426
- nk14 (Java field), 397
- nk15 (Java field), 398
- NMMin (Java class), 320
- NMMin() (Java constructor), 321
- NMMin(int, int, int, int, int) (Java constructor), 321
- NMMin2 (Java class), 321
- NMMin2() (Java constructor), 321
- NMMin2(int, int, int, int, int, int) (Java constructor), 321
- NMMinTest (Java class), 322
- nobj (Java field), 380
- Node_ (Java field), 352
- nodeCheck(int, int) (Java method), 375
- nodeCheck_ (Java field), 361
- NodeRestrict_ (Java field), 352
- NON_ELITIST (Java field), 217
- nondominated_sorting_add(S) (Java method), 112
- nondominated_sorting_delete(S) (Java method), 112
- nonDominatedArchive (Java field), 165
- NonDominatedSolutionListArchive (Java class), 561
- NonDominatedSolutionListArchive() (Java constructor), 561
- NonDominatedSolutionListArchive(DominanceComparator) (Java constructor), 561
- NonDominatedSolutionListArchiveTest (Java class), 562
- NonElitistEvolutionStrategy (Java class), 218
- NonElitistEvolutionStrategy(Problem, int, int, int, MutationOperator) (Java constructor), 218
- NonElitistEvolutionStrategyRunner (Java class), 513
- NonUniformMutation (Java class), 289
- NonUniformMutation(double, double, int) (Java constructor), 289
- NonUniformMutation(double, double, int, RandomGenerator) (Java constructor), 289
- NonUniformMutationTest (Java class), 290
- norm(double[]) (Java method), 219
- norm_vector(double[]) (Java method), 112, 117
- normalise(float[]) (Java method), 407
- normalize(Double, int) (Java method), 130
- normalize(Front) (Java method), 620
- normalize(List) (Java method), 620
- normalizeObjectives(List, List, List) (Java method), 150
- Normalizer (Java class), 130
- normalizer (Java field), 124
- Normalizer(List, List) (Java constructor), 130
- NPointCrossover (Java class), 270
- NPointCrossover(double, int) (Java constructor), 270
- NPointCrossover(int) (Java constructor), 270
- NSGAI (Java class), 133
- NSGAI (Java field), 139, 169

- NSGAI(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, Comparator, SolutionListEvaluator) (Java constructor), 134
 - NSGAI(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 134
 - NSGAI45 (Java class), 135
 - NSGAI45 (Java field), 139, 169
 - NSGAI45(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 136
 - NSGAI45Runner (Java class), 493
 - NSGAIBigDataRunner (Java class), 494
 - NSGAIBinaryRunner (Java class), 494
 - NSGAIBuilder (Java class), 137
 - NSGAIBuilder(Problem, CrossoverOperator, MutationOperator) (Java constructor), 138
 - NSGAIBuilderTest (Java class), 140
 - NSGAIEbesRunner (Java class), 495
 - NSGAIII (Java class), 145
 - NSGAIII(NSGAIIIBuilder) (Java constructor), 146
 - NSGAIIIBuilder (Java class), 147
 - NSGAIIIBuilder(Problem) (Java constructor), 147
 - NSGAIIIntegerRunner (Java class), 495
 - NSGAIIIRunner (Java class), 495
 - NSGAIIIT (Java class), 141
 - NSGAIIIMeasures (Java class), 141
 - NSGAIIIMeasures(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, Comparator, SolutionListEvaluator) (Java constructor), 142
 - NSGAIIIMeasuresRunner (Java class), 496
 - NSGAIIIMeasuresWithChartsRunner (Java class), 496
 - NSGAIIIMeasuresWithHypervolumeRunner (Java class), 497
 - NSGAIIIRunner (Java class), 497
 - NSGAIIStoppingByTime (Java class), 143
 - NSGAIIStoppingByTime(Problem, int, long, CrossoverOperator, MutationOperator, SelectionOperator, Comparator) (Java constructor), 144
 - NSGAIIStoppingByTimeRunner (Java class), 498
 - NSGAIIStudy (Java class), 231
 - NSGAIIStudy2 (Java class), 232
 - NSGAIIITSPRunner (Java class), 498
 - NSGAIIVariant (Java enum), 139, 169
 - NullCrossover (Java class), 271
 - NullCrossoverTest (Java class), 271
 - NullMutation (Java class), 290
 - NUM_FUNC (Java field), 445–454
 - NUM_TEST_FUNC (Java field), 429
 - numberFormatter (Java field), 429
 - numberOfBasicFunctions (Java field), 456
 - numberOfCities (Java field), 320
 - numberOfConstraintsGeometric_ (Java field), 361
 - numberOfConstraintsNodes_ (Java field), 361
 - numberOfDimensions (Java field), 456
 - numberOfDivisions (Java field), 146
 - numberOfElements_ (Java field), 361
 - numberOfEval_ (Java field), 361
 - numberOfGroupElements_ (Java field), 361
 - numberOfGroupsToCheckGeometry_ (Java field), 362
 - numberOfLibertyDegree_ (Java field), 362
 - numberOfNodes (Java field), 362
 - numberOfNodesRestricts(int) (Java method), 375
 - numberOfNodesRestricts_ (Java field), 362
 - numberOfNonDominatedSolutionsInPopulation (Java field), 142
 - numberOfObjectives (Java field), 569
 - numberOfPoints (Java field), 610
 - numberOfSubRanges (Java field), 29
 - numberOfThreads (Java field), 47, 106
 - NumberOfViolatedConstraints (Java class), 679
 - numberOfViolatedConstraints (Java field), 316–318, 323–327, 335–338
 - numberOfWeighthHypothesis_ (Java field), 362
 - numberOfWeightsElements_ (Java field), 362
 - numberOfWeightsNodes_ (Java field), 362
 - numRanks (Java field), 109
 - nvar (Java field), 381
 - NxxMax_ (Java field), 352
 - NxxMin_ (Java field), 352
- ## O
- Objective (Java interface), 521
 - objective(List, List) (Java method), 381
 - ObjectiveComparator (Java class), 583
 - ObjectiveComparator(int) (Java constructor), 583
 - ObjectiveComparator(int, Ordering) (Java constructor), 583
 - ObjectiveComparatorTest (Java class), 584
 - ObjectiveFactory (Java class), 533
 - ObjectiveFactoryTest (Java class), 534
 - occupiedHypercubes() (Java method), 541
 - OF_ (Java field), 352
 - offset (Java field), 186, 188
 - offspringPopulation (Java field), 100
 - OldStrainMax_ (Java field), 353
 - OldStrainMin_ (Java field), 353
 - omegaMax_ (Java field), 362
 - OMOPSO (Java class), 153
 - OMOPSO(DoubleProblem, SolutionListEvaluator, int, int, int, UniformMutation, NonUniformMutation) (Java constructor), 153
 - OMOPSOBuilder (Java class), 155
 - OMOPSOBuilder(DoubleProblem, SolutionListEvaluator) (Java constructor), 156
 - OMOPSOIT (Java class), 157
 - OMOPSORunner (Java class), 498

- OneMax (Java class), 426
- OneMax() (Java constructor), 426
- OneMax(Integer) (Java constructor), 426
- OneZeroMax (Java class), 322
- OneZeroMax() (Java constructor), 322
- OneZeroMax(Integer) (Java constructor), 322
- operator (Java field), 279
- Operator (Java interface), 261
- OPT (Java field), 483
- Ordering (Java enum), 583
- org.uma.jmetal.algorithm (package), 13
- org.uma.jmetal.algorithm.impl (package), 14
- org.uma.jmetal.algorithm.multiobjective.abys (package), 27
- org.uma.jmetal.algorithm.multiobjective.abys.util (package), 37
- org.uma.jmetal.algorithm.multiobjective.artificialdecisionmaker (package), 37
- org.uma.jmetal.algorithm.multiobjective.cdg (package), 37
- org.uma.jmetal.algorithm.multiobjective.cellde (package), 50
- org.uma.jmetal.algorithm.multiobjective.dmopso (package), 51
- org.uma.jmetal.algorithm.multiobjective.espea (package), 61
- org.uma.jmetal.algorithm.multiobjective.espea.util (package), 67
- org.uma.jmetal.algorithm.multiobjective.gde3 (package), 75
- org.uma.jmetal.algorithm.multiobjective.gwasfga (package), 80
- org.uma.jmetal.algorithm.multiobjective.gwasfga.util (package), 81
- org.uma.jmetal.algorithm.multiobjective.ibea (package), 82
- org.uma.jmetal.algorithm.multiobjective.mocell (package), 86
- org.uma.jmetal.algorithm.multiobjective.mochc (package), 93
- org.uma.jmetal.algorithm.multiobjective.moea (package), 99
- org.uma.jmetal.algorithm.multiobjective.moea.util (package), 118
- org.uma.jmetal.algorithm.multiobjective.mombi (package), 119
- org.uma.jmetal.algorithm.multiobjective.mombi.util (package), 125
- org.uma.jmetal.algorithm.multiobjective.mombi2 (package), 133
- org.uma.jmetal.algorithm.multiobjective.nsgaii (package), 133
- org.uma.jmetal.algorithm.multiobjective.nsgaiii (package), 145
- org.uma.jmetal.algorithm.multiobjective.nsgaiii.util (package), 149
- org.uma.jmetal.algorithm.multiobjective.omopso (package), 153
- org.uma.jmetal.algorithm.multiobjective.paes (package), 157
- org.uma.jmetal.algorithm.multiobjective.pesa2 (package), 161
- org.uma.jmetal.algorithm.multiobjective.pesa2.util (package), 164
- org.uma.jmetal.algorithm.multiobjective.randomsearch (package), 165
- org.uma.jmetal.algorithm.multiobjective.rnsgaii (package), 166
- org.uma.jmetal.algorithm.multiobjective.rnsgaii.util (package), 170
- org.uma.jmetal.algorithm.multiobjective.smpso (package), 171
- org.uma.jmetal.algorithm.multiobjective.smsema (package), 185
- org.uma.jmetal.algorithm.multiobjective.spea2 (package), 190
- org.uma.jmetal.algorithm.multiobjective.spea2.util (package), 194
- org.uma.jmetal.algorithm.multiobjective.wasfga (package), 195
- org.uma.jmetal.algorithm.multiobjective.wasfga.util (package), 199
- org.uma.jmetal.algorithm.singleobjective.coralreefsoptimization (package), 202
- org.uma.jmetal.algorithm.singleobjective.differentialevolution (package), 206
- org.uma.jmetal.algorithm.singleobjective.evolutionstrategy (package), 213
- org.uma.jmetal.algorithm.singleobjective.evolutionstrategy.util (package), 219
- org.uma.jmetal.algorithm.singleobjective.geneticalgorithm (package), 220
- org.uma.jmetal.algorithm.singleobjective.particleswarmoptimization (package), 225
- org.uma.jmetal.experiment (package), 230
- org.uma.jmetal.measure (package), 235
- org.uma.jmetal.measure.impl (package), 238
- org.uma.jmetal.operator (package), 260
- org.uma.jmetal.operator.impl.crossover (package), 261
- org.uma.jmetal.operator.impl.localsearch (package), 280
- org.uma.jmetal.operator.impl.mutation (package), 283
- org.uma.jmetal.operator.impl.selection (package), 297
- org.uma.jmetal.problem (package), 308
- org.uma.jmetal.problem.impl (package), 312
- org.uma.jmetal.problem.multiobjective (package), 316
- org.uma.jmetal.problem.multiobjective.cdtlz (package), 334
- org.uma.jmetal.problem.multiobjective.cec2015OptBigDataCompetition

- (package), 339
 - org.uma.jmetal.problem.multiobjective.dtlz (package), 341
 - org.uma.jmetal.problem.multiobjective.ebes (package), 345
 - org.uma.jmetal.problem.multiobjective.glt (package), 376
 - org.uma.jmetal.problem.multiobjective.lz09 (package), 380
 - org.uma.jmetal.problem.multiobjective.maf (package), 386
 - org.uma.jmetal.problem.multiobjective.mop (package), 399
 - org.uma.jmetal.problem.multiobjective.UF (package), 328
 - org.uma.jmetal.problem.multiobjective.wfg (package), 404
 - org.uma.jmetal.problem.multiobjective.zdt (package), 419
 - org.uma.jmetal.problem.singleobjective (package), 424
 - org.uma.jmetal.problem.singleobjective.cec2005competitionmaker (package), 429
 - org.uma.jmetal.qualityIndicator (package), 458
 - org.uma.jmetal.qualityindicator (package), 458
 - org.uma.jmetal.qualityindicator.impl (package), 458
 - org.uma.jmetal.qualityindicator.impl.hypervolume (package), 478
 - org.uma.jmetal.qualityindicator.impl.hypervolume.util (package), 482
 - org.uma.jmetal.runner.multiobjective (package), 485
 - org.uma.jmetal.runner.singleobjective (package), 511
 - org.uma.jmetal.solution (package), 515
 - org.uma.jmetal.solution.impl (package), 521
 - org.uma.jmetal.solution.util (package), 536
 - org.uma.jmetal.util (package), 539
 - org.uma.jmetal.util.archive (package), 555
 - org.uma.jmetal.util.archive.impl (package), 557
 - org.uma.jmetal.util.archivewithreferencepoint (package), 563
 - org.uma.jmetal.util.archivewithreferencepoint.impl (package), 564
 - org.uma.jmetal.util.artificialdecisionmaker (package), 566
 - org.uma.jmetal.util.artificialdecisionmaker.impl (package), 568
 - org.uma.jmetal.util.binarySet (package), 572
 - org.uma.jmetal.util.chartcontainer (package), 573
 - org.uma.jmetal.util.comparator (package), 577
 - org.uma.jmetal.util.comparator.impl (package), 588
 - org.uma.jmetal.util.distance (package), 590
 - org.uma.jmetal.util.distance.impl (package), 590
 - org.uma.jmetal.util.evaluator (package), 592
 - org.uma.jmetal.util.evaluator.impl (package), 592
 - org.uma.jmetal.util.experiment (package), 593
 - org.uma.jmetal.util.experiment.component (package), 597
 - org.uma.jmetal.util.experiment.util (package), 602
 - org.uma.jmetal.util.extremevalues (package), 604
 - org.uma.jmetal.util.extremevalues.impl (package), 605
 - org.uma.jmetal.util.fileinput.util (package), 605
 - org.uma.jmetal.util.fileoutput (package), 606
 - org.uma.jmetal.util.fileoutput.impl (package), 608
 - org.uma.jmetal.util.front (package), 609
 - org.uma.jmetal.util.front.impl (package), 609
 - org.uma.jmetal.util.front.util (package), 619
 - org.uma.jmetal.util.naming (package), 624
 - org.uma.jmetal.util.naming.impl (package), 624
 - org.uma.jmetal.util.neighborhood (package), 628
 - org.uma.jmetal.util.neighborhood.impl (package), 629
 - org.uma.jmetal.util.neighborhood.util (package), 639
 - org.uma.jmetal.util.point (package), 642
 - org.uma.jmetal.util.point.impl (package), 645
 - org.uma.jmetal.util.point.util (package), 655
 - org.uma.jmetal.util.point.util.comparator (package), 657
 - org.uma.jmetal.util.point.util.distance (package), 659
 - org.uma.jmetal.util.pseudorandom (package), 660
 - org.uma.jmetal.util.pseudorandom.impl (package), 666
 - org.uma.jmetal.util.solutionattribute (package), 673
 - org.uma.jmetal.util.solutionattribute.impl (package), 675
 - org.uma.jmetal.utility (package), 681
 - org.uma.jmetal.workingTest (package), 681
 - Osyczka2 (Java class), 323
 - Osyczka2() (Java constructor), 323
 - OverallConstraintViolation (Java class), 680
 - OverallConstraintViolationComparator (Java class), 588
 - OverallConstraintViolationComparator() (Java constructor), 588
 - overallConstraintViolationDegree (Java field), 316–318, 323–327, 335–338, 362
 - OverloadInElement_ (Java field), 353
- ## P
- PAES (Java class), 157
 - PAES(Problem, int, int, int, MutationOperator) (Java constructor), 158
 - PAESBuilder (Java class), 160
 - PAESBuilder(Problem) (Java constructor), 160
 - PAESRunner (Java class), 499
 - ParallelGDE3Runner (Java class), 500
 - ParallelGenerationalGeneticAlgorithmRunner (Java class), 513
 - ParallelNSGAIIRunner (Java class), 500
 - ParallelPESA2Runner (Java class), 501
 - ParallelSMPSORunner (Java class), 501
 - ParallelSPEA2Runner (Java class), 501
 - parentSelection (Java field), 96
 - parentSelection(int, NeighborType) (Java method), 44, 102

- paretoDom(S, int) (Java method), 44
- paretoFilter() (Java method), 44
- paretoOptimalSolutions (Java field), 566
- PBI (Java field), 54, 102
- percentageFormatter (Java field), 429
- PermutationProblem (Java interface), 311
- PermutationSolution (Java interface), 518
- PermutationSwapMutation (Java class), 291
- PermutationSwapMutation(double) (Java constructor), 291
- PermutationSwapMutation(double, RandomGenerator) (Java constructor), 291
- PermutationSwapMutation(double, RandomGenerator, BoundedRandomGenerator) (Java constructor), 291
- PermutationSwapMutationTest (Java class), 292
- perpendicularDistance(List, List) (Java method), 150
- perturbation(List) (Java method), 25, 155, 173, 183, 227, 229
- PESA2 (Java class), 161
- PESA2(Problem, int, int, int, int, CrossoverOperator, MutationOperator, SolutionListEvaluator) (Java constructor), 162
- PESA2Builder (Java class), 163
- PESA2Builder(Problem, CrossoverOperator, MutationOperator) (Java constructor), 163
- PESA2Runner (Java class), 499
- PESA2Selection (Java class), 164
- PESA2Selection() (Java constructor), 164
- pi (Java field), 362
- pindex9 (Java field), 392
- PISAHypervolume (Java class), 478
- PISAHypervolume() (Java constructor), 479
- PISAHypervolume(Front) (Java constructor), 479
- PISAHypervolume(String) (Java constructor), 479
- PISAHypervolumeTest (Java class), 480
- PIx2 (Java field), 429, 442
- pj (Java field), 363
- PMXCrossover (Java class), 272
- PMXCrossover(double) (Java constructor), 272
- PMXCrossover(double, RandomGenerator) (Java constructor), 272
- PMXCrossover(double, RandomGenerator, BoundedRandomGenerator) (Java constructor), 272
- PMXCrossoverTest (Java class), 273
- point (Java field), 645
- Point (Java interface), 642
- PointComparator (Java class), 658
- PointComparator() (Java constructor), 658
- PointComparatorTest (Java class), 652
- PointDimensionComparator (Java class), 659
- PointDimensionComparator(int) (Java constructor), 659
- PointDimensionComparatorTest (Java class), 652
- PointDistance (Java interface), 660
- points (Java field), 610
- points9 (Java field), 393
- PointSolution (Java class), 642
- PointSolution(int) (Java constructor), 643
- PointSolution(Point) (Java constructor), 643
- PointSolution(PointSolution) (Java constructor), 643
- PointSolution(Solution) (Java constructor), 643
- PointSolutionTest (Java class), 654
- polygonpoints(int, double) (Java method), 392, 394
- PolynomialMutation (Java class), 292
- PolynomialMutation() (Java constructor), 292
- PolynomialMutation(double, double) (Java constructor), 292
- PolynomialMutation(double, double, RandomGenerator) (Java constructor), 293
- PolynomialMutation(double, double, RepairDoubleSolution) (Java constructor), 293
- PolynomialMutation(double, double, RepairDoubleSolution, RandomGenerator) (Java constructor), 293
- PolynomialMutation(DoubleProblem, double) (Java constructor), 292
- PolynomialMutation(DoubleProblem, double, RandomGenerator) (Java constructor), 292
- PolynomialMutationTest (Java class), 293
- PolynomialMutationWorkingTest (Java class), 683
- POPULATION (Java field), 45, 103
- population (Java field), 14, 20, 39, 100, 135
- populationIsNotFull(List) (Java method), 77, 121, 137
- populationSize (Java field), 39, 47, 78, 83, 89, 96, 101, 106, 136, 188, 192
- pos() (Java method), 153
- position (Java field), 152
- PQ (Java field), 353
- PreferenceDistance (Java class), 680
- PreferenceDistance(List, double) (Java constructor), 680
- preferenceDistanceSelection(Ranking, int) (Java method), 307
- PreferenceNSGAI (Java class), 170
- PreferenceNSGAI(List) (Java constructor), 170
- prefers(int, int, int[], int) (Java method), 117
- prepare(Variable, Value) (Java method), 520
- prepareFileOutputContents(double[]) (Java method), 600
- preservedPopulation (Java field), 96
- print() (Java method), 607
- printEndLatexCommands(String) (Java method), 600
- printFinalSolutionSet(List) (Java method), 539
- printHeaderLatexCommands(String) (Java method), 600
- printObjectivesToFile(FileOutputContext, List) (Java method), 607
- printObjectivesToFile(FileOutputContext, List, List) (Java method), 607
- printObjectivesToFile(String) (Java method), 607
- printObjectivesToFile(String, List) (Java method), 607
- printQualityIndicators(List, String) (Java method), 540

- printVariablesToFile(FileOutputContext, List) (Java method), 607
- printVariablesToFile(String) (Java method), 607
- problem (Java field), 20, 29, 35, 36, 39, 47, 83, 90, 92, 96, 101, 106, 136, 156, 188, 193, 322, 522, 523, 527, 566
- Problem (Java interface), 311
- ProblemUtils (Java class), 546
- PRODUCT_OF_OBJECTIVES (Java field), 74
- productOfObjectives(List) (Java method), 71
- prune() (Java method), 69, 558–561, 564
- PseudoRandomGenerator (Java interface), 663
- psfunc2(double, double, int, int, int) (Java method), 381
- psfunc3(double, double, double, int, int) (Java method), 381
- ptype (Java field), 381
- PullMeasure (Java interface), 237
- PullPushMeasure (Java class), 252
- PullPushMeasure(PullMeasure, PushMeasure, DescribedEntity) (Java constructor), 252
- PullPushMeasure(PullMeasure, PushMeasure, String, String) (Java constructor), 252
- PullPushMeasure(PushMeasure, Value) (Java constructor), 253
- PullPushMeasure(String, String) (Java constructor), 253
- push(Solution, Value) (Java method), 244
- push(T) (Java method), 239
- push(Value) (Java method), 259
- PushMeasure (Java interface), 237
- ## Q
- Qa_ (Java field), 354
- QAx_ (Java field), 353
- QAy_ (Java field), 353
- QAz_ (Java field), 353
- Qb_ (Java field), 354
- QE_ (Java field), 353
- QH_ (Java field), 353
- Qi (Java field), 354
- Qj (Java field), 354
- QT_ (Java field), 353
- QualityIndicator (Java interface), 458
- quickSort(double[], int[], int, int) (Java method), 119
- ## R
- R2 (Java class), 473
- R2() (Java constructor), 473
- R2(Front) (Java constructor), 473
- r2(Front) (Java method), 474
- R2(int) (Java constructor), 473
- R2(int, Front) (Java constructor), 473
- R2(String) (Java constructor), 473
- R2(String, Front) (Java constructor), 473
- R2Ranking (Java class), 130
- R2Ranking(AbstractUtilityFunctionsSet) (Java constructor), 130
- R2RankingAttribute (Java class), 131
- R2RankingNormalized (Java class), 131
- R2RankingNormalized(AbstractUtilityFunctionsSet, Normalizer) (Java constructor), 131
- R2SolutionData (Java class), 132
- R2Test (Java class), 474
- randNormal(double, double) (Java method), 670
- random (Java field), 406, 429, 569
- randomGenerator (Java field), 29, 40, 101, 113, 116, 522, 523
- RandomGenerator (Java interface), 664
- RandomGeneratorTest (Java class), 665
- RandomMember() (Java method), 153
- RandomMethod (Java enum), 668
- randomOccupiedHypercube() (Java method), 542
- randomOccupiedHypercube(BoundedRandomGenerator) (Java method), 542
- randomPermutation(int[], int) (Java method), 119
- RandomSearch (Java class), 165
- RandomSearch(Problem, int) (Java constructor), 165
- RandomSearchBuilder (Java class), 166
- RandomSearchBuilder(Problem) (Java constructor), 166
- RandomSearchRunner (Java class), 503
- RandomSelection (Java class), 304
- RandomSelectionTest (Java class), 304
- randSphere(int) (Java method), 670
- randSphere(int, double, double) (Java method), 670
- rank (Java field), 132
- rankBasedSelection() (Java method), 44
- rankIdx (Java field), 109
- ranking (Java field), 76, 109
- Ranking (Java interface), 674
- RankingAndCrowdingDistanceComparator (Java class), 585
- RankingAndCrowdingDistanceComparatorTest (Java class), 585
- RankingAndCrowdingSelection (Java class), 305
- RankingAndCrowdingSelection(int) (Java constructor), 305
- RankingAndCrowdingSelection(int, Comparator) (Java constructor), 305
- RankingAndCrowdingSelectionTest (Java class), 306
- RankingAndPreferenceSelection (Java class), 306
- RankingAndPreferenceSelection(int, List, double) (Java constructor), 306
- rankingCoefficient (Java field), 569
- RankingComparator (Java class), 586
- RankingComparatorTest (Java class), 587
- rankUnfeasibleSolutions(List) (Java method), 82, 200
- Rastrigin (Java class), 427
- Rastrigin(double[]) (Java method), 398
- rastrigin(double[]) (Java method), 432

- Rastrigin(Integer) (Java constructor), 427
 rastriginNonCont(double[]) (Java method), 432
 RATIO_YZ (Java field), 354
 Reaction_ (Java field), 354
 ReadDoubleDataFile (Java class), 605
 readFile(String) (Java method), 606
 readFromFile(String) (Java method), 201
 readFromResourcesInJMetal(String) (Java method), 201
 recompute() (Java method), 630
 RECTANGLE (Java field), 354
 referenceFront (Java field), 60, 142
 referenceParetoFront (Java field), 466
 ReferencePoint (Java class), 151
 referencePoint (Java field), 120, 564
 ReferencePoint() (Java constructor), 152
 ReferencePoint(int) (Java constructor), 152
 ReferencePoint(ReferencePoint) (Java constructor), 152
 referencePoints (Java field), 146, 182
 referencePointSolution (Java field), 564
 referenceSet1 (Java field), 29
 referenceSet1Size (Java field), 29
 referenceSet2 (Java field), 29
 referenceSet2Size (Java field), 29
 referenceSetUpdate() (Java method), 26, 31
 referenceSetUpdate(DoubleSolution) (Java method), 31
 referenceSetUpdate(S) (Java method), 26
 refreshCharts() (Java method), 574, 576
 refreshCharts(int) (Java method), 574, 576
 refSet1Test(DoubleSolution) (Java method), 31
 refSet2Test(DoubleSolution) (Java method), 31
 register(MeasureListener) (Java method), 238, 253, 259
 relevantObjectiveFunctions(List) (Java method), 570
 relevantObjectiveFunctions(R) (Java method), 567
 remove(Object) (Java method), 625
 remove(String) (Java method), 625
 removeAll(Collection) (Java method), 625
 removeAllMeasures(Iterable) (Java method), 255
 removeDominatedSolutionsInArchives() (Java method), 184
 removeDuplicatedAlgorithms() (Java method), 594
 removeIndicator(String) (Java method), 574
 removeListener(Consumer) (Java method), 667
 removeMeasure(Object) (Java method), 255
 RemovePotentialMember(S) (Java method), 153
 removePullMeasure(Object) (Java method), 255
 removePushMeasure(Object) (Java method), 255
 removeSolution(int) (Java method), 542
 removeSolutionsFromList(List, int) (Java method), 549
 removeWorst(List) (Java method), 84
 repaint() (Java method), 574, 576
 RepairDoubleSolution (Java interface), 536
 RepairDoubleSolutionAtBounds (Java class), 537
 RepairDoubleSolutionAtBoundsTest (Java class), 537
 RepairDoubleSolutionAtRandom (Java class), 538
 RepairDoubleSolutionAtRandom() (Java constructor), 538
 RepairDoubleSolutionAtRandom(BoundedRandomGenerator) (Java constructor), 538
 RepairDoubleSolutionAtRandomTest (Java class), 539
 repairSolutionVariableValue(double, double, double) (Java method), 537, 538
 replace(int, S) (Java method), 113
 replacement(List, List) (Java method), 21, 63, 77, 88, 94, 123, 135, 143, 147, 159, 162, 167, 187, 192, 197, 208, 214, 216, 219, 221, 224
 ReplacementStrategy (Java enum), 69
 reproduction(List) (Java method), 21, 23, 63, 77, 88, 94, 121, 145, 147, 159, 162, 187, 192, 208, 214, 216, 219, 224
 reset() (Java method), 241, 243, 246
 reset(long) (Java method), 241
 restart() (Java method), 27, 32
 restart(List, Problem, int) (Java method), 549
 restartConditionIsFulfilled(List) (Java method), 27, 32
 resultPopulationSize (Java field), 40, 47, 101, 106
 retainAll(Collection) (Java method), 625
 reverseFrequency (Java field), 29
 RIG_ART (Java field), 354
 RIG_RIG (Java field), 354
 Rij (Java field), 355
 Rji (Java field), 355
 rNonsep(float[], int) (Java method), 405
 RNSGAII (Java class), 166
 RNSGAII(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, List, double) (Java constructor), 167
 RNSGAIIBuilder (Java class), 168
 RNSGAIIBuilder(Problem, CrossoverOperator, MutationOperator, List, double) (Java constructor), 168
 RNSGAIIConstraintRunner (Java class), 502
 RNSGAIIRanking (Java class), 170
 RNSGAIIRanking(PreferenceNSGAII, double, List) (Java constructor), 171
 RNSGAIIRunner (Java class), 502
 Rosenbrock (Java class), 427
 Rosenbrock(double[]) (Java method), 398
 rosenbrock(double[]) (Java method), 432
 Rosenbrock(Integer) (Java constructor), 427
 rotate(double[], double[], double[][]) (Java method), 432
 rouletteWheel() (Java method), 542
 rouletteWheel(BoundedRandomGenerator) (Java method), 542
 Rpij (Java field), 355
 Rpij (Java field), 355
 RpTij (Java field), 355
 RpTji (Java field), 355
 rSum(float[], float[]) (Java method), 405

RTij (Java field), 354
 RTji (Java field), 354
 run() (Java method), 13, 17, 21, 25, 27, 46, 51, 54, 61, 84, 95, 104, 105, 113, 114, 117, 121, 137, 143, 166, 181, 199, 567, 597–602
 runAlgorithm(Experiment) (Java method), 603
 runTest() (Java method), 433
 runTest(int) (Java method), 433
 rZ_ (Java field), 363

S

s (Java field), 407
 saveChart(String, BitmapFormat) (Java method), 574, 576
 savedValues (Java field), 114, 116
 SBXCrossover (Java class), 273
 SBXCrossover(double, double) (Java constructor), 274
 SBXCrossover(double, double, RandomGenerator) (Java constructor), 274
 SBXCrossover(double, double, RepairDoubleSolution) (Java constructor), 274
 SBXCrossover(double, double, RepairDoubleSolution, RandomGenerator) (Java constructor), 274
 SBXCrossoverTest (Java class), 275
 SBXCrossoverWorkingTest (Java class), 684
 SafferF6(double, double) (Java method), 431
 ScalarizationType (Java enum), 74
 ScalarizationUtils (Java class), 70
 ScalarizationValue (Java class), 72
 ScalarizationWrapper (Java class), 73
 ScalarizationWrapper(Config) (Java constructor), 73
 ScalarizationWrapper(ScalarizationType) (Java constructor), 73
 scaling (Java field), 340
 Schaffer (Java class), 323
 Schaffer() (Java constructor), 324
 schwefel_102(double[]) (Java method), 433
 scientificFormatter (Java field), 430
 sDecept(float, float, float, float) (Java method), 405
 selectBest(R2Ranking) (Java method), 123
 selectBest(Ranking) (Java method), 197
 selectBroadcastSpawners(List) (Java method), 17, 203
 SelectClusterMember(ReferencePoint) (Java method), 149
 selectedOF (Java field), 363
 selectGlobalBest() (Java method), 173, 184
 selection(List) (Java method), 21, 23, 63, 77, 89, 94, 121, 145, 147, 160, 162, 187, 192, 208, 214, 216, 219, 224
 selectionOperator (Java field), 14, 19, 22, 78, 83, 90, 136, 161, 188, 193
 SelectionOperator (Java interface), 261
 selectNRandomDifferentSolutions(int, List) (Java method), 549

selectNRandomDifferentSolutions(int, List, BoundedRandomGenerator) (Java method), 549
 separator (Java field), 608
 SequentialSolutionListEvaluator (Java class), 593
 set(T) (Java method), 239
 setAlgorithm(InteractiveAlgorithm) (Java method), 571
 setAlgorithmList(List) (Java method), 595, 596
 setAllMeasures(Map) (Java method), 256
 setAlpha(double) (Java method), 263
 setArchive(BoundedArchive) (Java method), 91
 setArchiveSize(int) (Java method), 33, 86, 156, 161, 164
 setAsp(List) (Java method), 571
 setAttemptsToSettle(int) (Java method), 205
 setAttribute(Object, Object) (Java method), 519, 523, 525, 644
 setAttribute(S, List) (Java method), 150
 setAttribute(S, R2SolutionData) (Java method), 131
 setAttribute(S, V) (Java method), 675, 678
 setBiSections(int) (Java method), 161
 setBisections(int) (Java method), 164
 setC1Max(double) (Java method), 57, 176
 setC1Min(double) (Java method), 57, 176
 setC2Max(double) (Java method), 57, 176
 setC2Min(double) (Java method), 57, 177
 setCataclysmicMutation(MutationOperator) (Java method), 98
 setChangeVelocity1(double) (Java method), 57, 177
 setChangeVelocity2(double) (Java method), 57, 177
 setChildGrid(int) (Java method), 49
 setChildGridNum(int) (Java method), 49
 setColumns(int) (Java method), 599
 setComparator(Comparator) (Java method), 205
 setConsiderationProbability(double) (Java method), 571
 setConvergenceValue(int) (Java method), 98
 setCoordinates(List) (Java method), 17
 SetCoverage (Java class), 475
 SetCoverage() (Java constructor), 475
 SetCoverageTest (Java class), 476
 setCr(double) (Java method), 266
 setCrossover(CrossoverOperator) (Java method), 49, 86, 98, 107
 setCrossover(DifferentialEvolutionCrossover) (Java method), 79, 209
 setCrossoverOperator(CrossoverOperator) (Java method), 33, 65, 148, 189
 setCrossoverProbability(double) (Java method), 263, 268, 270, 273, 275, 278
 setCurrentIteration(int) (Java method), 290
 setCurrentSolution(DoubleSolution) (Java method), 267
 setDataDirectory(String) (Java method), 57, 108
 setDelay(int) (Java method), 574, 576
 setDelta(double) (Java method), 286
 setDescription(String) (Java method), 627
 setDisplayNotch() (Java method), 599

- [setDistributionIndex\(double\)](#) (Java method), [270](#), [275](#), [287](#), [293](#)
[setDominanceComparator\(Comparator\)](#) (Java method), [138](#), [189](#)
[setElementsBetweenDiffGreat\(int\)](#) (Java method), [375](#)
[setEvaluations\(int\)](#) (Java method), [208](#)
[setEvaluator\(SolutionListEvaluator\)](#) (Java method), [65](#), [98](#)
[setExperimentBaseDirectory\(String\)](#) (Java method), [596](#)
[setF\(double\)](#) (Java method), [267](#)
[setFa\(double\)](#) (Java method), [205](#)
[setFbr\(double\)](#) (Java method), [206](#)
[setFbs\(double\)](#) (Java method), [206](#)
[setFd\(double\)](#) (Java method), [206](#)
[setFrontChart\(int, int\)](#) (Java method), [574](#), [576](#)
[setFrontChart\(int, int, String\)](#) (Java method), [574](#), [576](#)
[setFronts\(List\)](#) (Java method), [151](#)
[setFullArchiveCrossoverOperator\(CrossoverOperator\)](#) (Java method), [65](#)
[setFunctionType\(DMOPSO.FunctionType\)](#) (Java method), [57](#)
[setFunctionType\(MOEA.D.FunctionType\)](#) (Java method), [108](#)
[setFunFileOutputContext\(FileOutputContext\)](#) (Java method), [607](#)
[setG\(S, int, int\)](#) (Java method), [44](#)
[setHypervolumeImplementation\(Hypervolume\)](#) (Java method), [189](#)
[setImprovementOperator\(ArchiveMutationLocalSearch\)](#) (Java method), [34](#)
[setIndependentRuns\(int\)](#) (Java method), [596](#)
[setIndex\(int\)](#) (Java method), [300](#)
[setIndicatorList\(List\)](#) (Java method), [596](#)
[setIndividualObjRank\(\)](#) (Java method), [44](#)
[setInitialConvergenceCount\(double\)](#) (Java method), [98](#)
[setIterations\(int\)](#) (Java method), [173](#)
[setK\(double\)](#) (Java method), [267](#)
[setK\(int\)](#) (Java method), [49](#)
[setLambda\(int\)](#) (Java method), [214](#), [217](#)
[setLocation\(S, double\[\], double\[\]\)](#) (Java method), [113](#)
[setLowerBounds\(List\)](#) (Java method), [170](#)
[setLowerLimit\(List\)](#) (Java method), [313](#), [315](#), [316](#)
[setM\(int\)](#) (Java method), [206](#)
[setMatrixWidthBand\(int\)](#) (Java method), [375](#)
[setMaxAge\(int\)](#) (Java method), [57](#)
[setMaxEvaluations\(int\)](#) (Java method), [34](#), [49](#), [66](#), [79](#), [86](#), [91](#), [98](#), [108](#), [139](#), [161](#), [164](#), [166](#), [169](#), [189](#), [206](#), [209](#), [214](#), [217](#), [222](#), [572](#)
[setMaximizing\(\)](#) (Java method), [658](#)
[setMaximumNumberOfReplacedSolutions\(int\)](#) (Java method), [108](#)
[setMaxIterations\(int\)](#) (Java method), [57](#), [148](#), [156](#), [177](#), [194](#), [290](#)
[setMaxPopulationSize\(int\)](#) (Java method), [24](#), [78](#), [94](#)
[setMeasure\(Object, Measure\)](#) (Java method), [256](#)
[setMinimizing\(\)](#) (Java method), [658](#)
[setMu\(int\)](#) (Java method), [217](#)
[setMutation\(MutationOperator\)](#) (Java method), [86](#), [108](#), [177](#)
[setMutationOperator\(MutationOperator\)](#) (Java method), [34](#), [66](#), [149](#), [161](#), [189](#)
[setMutationProbability\(Double\)](#) (Java method), [297](#)
[setMutationProbability\(double\)](#) (Java method), [284](#), [286](#), [287](#), [290](#), [292](#), [293](#), [296](#)
[setN\(int\)](#) (Java method), [206](#)
[setNadir\(List\)](#) (Java method), [127](#)
[setName\(String\)](#) (Java method), [58](#), [314](#), [574](#), [576](#), [627](#)
[setNegativeMaxNumberOfEvaluations\(\)](#) (Java method), [210](#)
[setNegativeMaxNumberOfIterations\(\)](#) (Java method), [140](#)
[setNegativePopulationSize\(\)](#) (Java method), [140](#), [210](#)
[setNeighborhood\(Neighborhood\)](#) (Java method), [91](#)
[setNeighborhoodSelectionProbability\(double\)](#) (Java method), [49](#), [108](#)
[setNeighborSize\(int\)](#) (Java method), [108](#)
[setNewCrossoverOperator\(\)](#) (Java method), [210](#)
[setNewEvaluator\(\)](#) (Java method), [140](#), [210](#)
[setNewGenerationSelection\(SelectionOperator\)](#) (Java method), [98](#)
[setNewSelectionOperator\(\)](#) (Java method), [140](#), [211](#)
[setNonUniformMutation\(MutationOperator\)](#) (Java method), [157](#)
[setNormalizeObjectives\(boolean\)](#) (Java method), [66](#)
[setNormalizer\(Normalizer\)](#) (Java method), [126](#)
[setNullEvaluator\(\)](#) (Java method), [140](#)
[setNullSelectionOperator\(\)](#) (Java method), [140](#)
[setNumberOfConstraints\(int\)](#) (Java method), [314](#)
[setnumberOfConstraintsGeometric\(int\)](#) (Java method), [376](#)
[setNumberOfConstraintsNodes\(int\)](#) (Java method), [375](#)
[setNumberOfCores\(int\)](#) (Java method), [597](#)
[setNumberOfDoubleVariables\(int\)](#) (Java method), [315](#)
[setNumberOfElements\(int\)](#) (Java method), [375](#)
[setnumberOfGroupElements\(int\)](#) (Java method), [376](#)
[setNumberOfIntegerVariables\(int\)](#) (Java method), [315](#)
[setNumberOfNodes\(int\)](#) (Java method), [375](#)
[setNumberOfObjectives\(int\)](#) (Java method), [151](#), [314](#)
[setNumberOfPoints\(int\)](#) (Java method), [483](#)
[setNumberOfSubranges\(int\)](#) (Java method), [34](#)
[setNumberOfThreads\(int\)](#) (Java method), [49](#), [108](#)
[setNumberOfVariables\(int\)](#) (Java method), [314](#)
[setNumberOfWeigthHypothesis\(int\)](#) (Java method), [375](#)
[setNumberOfWeigthsElements\(int\)](#) (Java method), [376](#)
[setNumberOfWeigthsNodes\(int\)](#) (Java method), [376](#)
[setNumberReferencePoints\(int\)](#) (Java method), [572](#)
[setObjective\(int, double\)](#) (Java method), [519](#), [523](#), [525](#), [644](#)

- setObjectiveMinimizingObjectiveList(List) (Java method), 607
- setOffset(double) (Java method), 189, 468, 480, 481
- setOrder(S, int) (Java method), 44
- setOutputParetoFrontFileName(String) (Java method), 597
- setOutputParetoSetFileName(String) (Java method), 597
- setParentSelection(SelectionOperator) (Java method), 98
- setPd(double) (Java method), 206
- setPerturbation(Double) (Java method), 297
- setPerturbation(double) (Java method), 290
- setPoint(int, Point) (Java method), 483, 609, 611
- setPopulation(List) (Java method), 17, 21, 27
- setPopulationSize(int) (Java method), 27, 34, 49, 66, 79, 86, 91, 98, 108, 139, 149, 164, 169, 189, 194, 209, 222
- setPositiveMaxNumberOfEvaluations() (Java method), 211
- setPositiveMaxNumberOfIterations() (Java method), 141
- setPreservedPopulation(double) (Java method), 99
- setProblem(Problem) (Java method), 21
- setProblemList(List) (Java method), 597
- setPullMeasure(Object, PullMeasure) (Java method), 256
- setPushMeasure(Object, PushMeasure) (Java method), 256
- setR1Max(double) (Java method), 58, 177
- setR1Min(double) (Java method), 58, 177
- setR2Max(double) (Java method), 58, 177
- setR2Min(double) (Java method), 58, 177
- setRandomGenerator(PseudoRandomGenerator) (Java method), 58, 177, 663
- setRank(S, int, int) (Java method), 44
- setRankingCoefficient(List) (Java method), 572
- setReferenceFront(Front) (Java method), 61, 143
- setReferenceFrontDirectory(String) (Java method), 595, 597
- setReferenceParetoFront(Front) (Java method), 467
- setReferenceParetoFront(String) (Java method), 467
- setReferencePoint(List) (Java method), 574, 577
- setReferencePoints(List) (Java method), 151
- setReferencePointValue(Double, int) (Java method), 121
- setRefSet1Size(int) (Java method), 34
- setRefSet2Size(int) (Java method), 34
- setReplacementStrategy(ReplacementStrategy) (Java method), 66
- setResultPopulationSize(int) (Java method), 49, 108
- setRho(double) (Java method), 206
- setRows(int) (Java method), 599
- setScalarization(ScalarizationWrapper) (Java method), 66
- setSeed(long) (Java method), 663, 664, 667, 670–673
- setSelection(DifferentialEvolutionSelection) (Java method), 79, 210
- setSelection(SelectionOperator) (Java method), 86
- setSelectionOperator(SelectionOperator) (Java method), 66, 91, 139, 149, 169, 190, 194, 222
- setSeparator(String) (Java method), 606, 607, 609
- setSigma(double) (Java method), 215
- setSolutionListEvaluator(SolutionListEvaluator) (Java method), 58, 91, 139, 149, 164, 169, 177, 194, 210, 222
- setSolutionSetEvaluator(SolutionListEvaluator) (Java method), 80
- setSolutionsToSelect(int) (Java method), 151
- setSpIndividualRank() (Java method), 44
- setStrainAdmissibleCut(int) (Java method), 376
- setSwarm(List) (Java method), 25
- setSwarmSize(int) (Java method), 58, 157, 178
- setT(int) (Java method), 50
- setTolerance(double) (Java method), 572
- setTypicalX(double[]) (Java method), 215
- setUniformMutation(MutationOperator) (Java method), 157
- setUp() (Java method), 527
- setup() (Java method), 35, 36, 92, 184, 185, 476, 482, 525, 537, 539, 578, 585, 587, 653, 655, 657
- setUpperBounds(List) (Java method), 170
- setUpperLimit(List) (Java method), 313, 315, 316
- setUtopia(List) (Java method), 127
- setValidPopulationSize() (Java method), 141, 211
- setValue(int, double) (Java method), 642, 646
- setVarChart(int, int) (Java method), 575
- setVarFileOutputContext(FileOutputContext) (Java method), 608
- setVariableValue(int, Double) (Java method), 525, 644
- setVariableValue(int, T) (Java method), 519, 523
- setVariant(DMOPSOVariant) (Java method), 58
- setVariant(GeneticAlgorithmVariant) (Java method), 223
- setVariant(NSGAIIVariant) (Java method), 139, 169
- setVariant(SMPISOVariant) (Java method), 178
- setWeightMax(double) (Java method), 58, 178
- setWeightMin(double) (Java method), 58, 178
- setX(int) (Java method), 19
- setY(int) (Java method), 19
- sexualReproduction(List) (Java method), 17, 203
- SHAPE (Java field), 355
- Shapes (Java class), 404
- shift(double[], double[], double[]) (Java method), 433
- shiftGlobalOptimum (Java field), 456
- shouldAddADominantSolutionInAnArchiveOfSize1DiscardTheExistingSolution (Java method), 562
- shouldAddADominantSolutionInAnArchiveOfSize3DiscardTheRestOfSolutions (Java method), 562
- shouldAddADominatedSolutionInAnArchiveOfSize1DiscardTheNewSolution (Java method), 562
- shouldAddANonDominantSolutionInAnArchiveOfSize1IncorporateTheNewSolution (Java method), 562

shouldAddASolutionEqualsToOneAlreadyInTheArchiveDoesNotAdd() (Java method), 562
 shouldAddOnAnEmptyListHaveSizeOne() (Java method), 563
 shouldAddOnAnEmptyListInsertTheElement() (Java method), 563
 shouldAverageDistanceToSolutionListWorkProperlyCaseA() (Java method), 555
 shouldAverageDistanceToSolutionListWorkProperlyCaseB() (Java method), 555
 shouldAverageDistanceToSolutionListWorkProperlyCaseC() (Java method), 555
 shouldCalculatingDistanceOfPointsWithOneDimensionReturnTheCorrectValue() (Java method), 655, 657
 shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValue() (Java method), 655, 657
 shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValue() (Java method), 656, 657
 shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValue() (Java method), 656
 shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValue() (Java method), 656
 shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValue() (Java method), 656
 shouldCalculatingDistanceOfPointsWithTwoDimensionsReturnTheCorrectValue() (Java method), 656
 shouldCalculatingDistanceOfPointsWithZeroDimensionReturnZero() (Java method), 656, 657
 shouldCompareBetterReturnZeroIfBothPointsAreEqualWhenMaximizing() (Java method), 652
 shouldCompareBetterReturnZeroIfBothPointsAreEqualWhenMinimizing() (Java method), 652
 shouldCompareDifferentLengthPointsReturnTheCorrectValue() (Java method), 650
 shouldCompareEmptyPointsReturnZero() (Java method), 650
 shouldCompareIdenticalPointsButTheFirstValueReturnMinusOne() (Java method), 650
 shouldCompareIdenticalPointsButTheFirstValueReturnPlusOne() (Java method), 650
 shouldCompareIdenticalPointsButTheLastValueReturnMinusOne() (Java method), 650
 shouldCompareIdenticalPointsButTheLastValueReturnPlusOne() (Java method), 650
 shouldCompareIdenticalPointsReturnZero() (Java method), 650
 shouldCompareRaiseAnExceptionIfSolution1HasLessObjectivesThanTheOneRequested() (Java method), 584
 shouldCompareRaiseAnExceptionIfSolution2HasLessObjectivesThanTheOneRequested() (Java method), 584
 shouldCompareRaiseAnExceptionIfTheFirstSolutionIsNull() (Java method), 580
 shouldCompareRaiseAnExceptionIfTheSecondSolutionIsNull() (Java method), 580
 shouldCompareRaiseAnExceptionIfTheSolutionsHaveNotTheSameNumberOfObjectives() (Java method), 580
 shouldCompareReturnMinusOneIfSolutionAHasLessRanking() (Java method), 587
 shouldCompareReturnMinusOneIfSolutionBHasHigherDistance() (Java method), 578
 shouldCompareReturnMinusOneIfTheFirstPointIsBetterThanTheSecondOne() (Java method), 652
 shouldCompareReturnMinusOneIfTheFirstSolutionDominatesTheSecondOne() (Java method), 580
 shouldCompareReturnMinusOneIfTheFirstSolutionDominatesTheSecondOne() (Java method), 580
 shouldCompareReturnMinusOneIfTheFirstValueIsLower() (Java method), 653
 shouldCompareReturnMinusOneIfTheObjectiveOfSolution1IsGreaterInDescending() (Java method), 584
 shouldCompareReturnMinusOneIfTheObjectiveOfSolution1IsLower() (Java method), 584
 shouldCompareReturnMinusOneIfTheSecondSolutionIsNull() (Java method), 578, 584, 587
 shouldCompareReturnMinusOneIfTheTwoSolutionsHasOneObjectiveAndTheSecondIsBetter() (Java method), 580
 shouldCompareReturnMinusOneIfSolutionAHasLessDistance() (Java method), 578
 shouldCompareReturnMinusOneIfSolutionBHasLessRanking() (Java method), 587
 shouldCompareReturnOneIfTheFirstSolutionIsNull() (Java method), 578, 584, 587
 shouldCompareReturnOneIfTheObjectiveOfSolution2IsGreaterInDescending() (Java method), 584
 shouldCompareReturnOneIfTheObjectiveOfSolution2IsLower() (Java method), 585
 shouldCompareReturnOneIfTheSecondPointIsBetterThanTheFirstOneWhenMaximizing() (Java method), 652
 shouldCompareReturnOneIfTheSecondSolutionDominatesTheFirstOneCaseA() (Java method), 580
 shouldCompareReturnOneIfTheSecondSolutionDominatesTheFirstOneCaseB() (Java method), 580
 shouldCompareReturnOneIfTheTwoSolutionsHasOneObjectiveAndTheSecondIsBetter() (Java method), 580
 shouldCompareReturnPlusOneIfTheFirstValueIsGreater() (Java method), 653
 shouldCompareReturnTheValueReturnedByTheConstraintViolationComparator() (Java method), 580
 shouldCompareReturnZeroIfBothSolutionsAreNull() (Java method), 578, 585, 587
 shouldCompareReturnZeroIfBothSolutionsHaveNoCrowdingDistanceAttribute() (Java method), 578
 shouldCompareReturnZeroIfBothSolutionsHaveNoRankingAttribute() (Java method), 587
 shouldCompareReturnZeroIfBothSolutionsHaveTheSameDistance() (Java method), 578
 shouldCompareReturnZeroIfBothSolutionsHaveTheSameRanking() (Java method), 587

`shouldCompareReturnZeroIfTheComparedValuesAreEqual()` (Java method), 653
`shouldCompareReturnZeroIfTheObjectiveOfTheSolutionsIsTheSame()` (Java method), 585
`shouldCompareReturnZeroIfTheTwoSolutionsHaveOneObjectiveWithTheSameValue()` (Java method), 581
`shouldCompareTwoNullSolutionsReturnZero()` (Java method), 585
`shouldCompareWhenRankingYieldingAZeroReturnTheCrowdingDistanceValue()` (Java method), 586
`shouldCompareWithANullSolutionAsFirstArgumentReturnOne()` (Java method), 586
`shouldCompareWithANullSolutionAsSecondArgumentReturnMinusOne()` (Java method), 586
`shouldCompareWithNullRankingAttributeSolutionAsFirstArgumentReturnOne()` (Java method), 586
`shouldCompareWithRankingYieldingANonZeroValueReturnThatValue()` (Java method), 586
`shouldComparingDifferentLengthPointsRaiseAnException()` (Java method), 652
`shouldConstructAPointFromANullPointRaiseAnException()` (Java method), 647
`shouldConstructAPointFromOtherPointReturnAnIdenticalPoint()` (Java method), 647
`shouldConstructAPointOfAGivenDimension()` (Java method), 647
`shouldConstructFromArrayReturnTheCorrectPoint()` (Java method), 647
`shouldConstructFromASolutionReturnTheCorrectPoint()` (Java method), 647
`shouldConstructFromNullArrayRaiseAnException()` (Java method), 647
`shouldConstructorAssignTheCorrectDistributionIndex()` (Java method), 263, 275, 287, 294
`shouldConstructorAssignTheCorrectProbabilityValue()` (Java method), 263, 275, 278, 284, 287, 294
`shouldConstructorAssignTheCorrectValueSToTheNumberOfTournamentsAndTheComparator()` (Java method), 308
`shouldConstructorAssignTheCorrectValueToTheNumberOfTournaments()` (Java method), 308
`shouldConstructorAssignThePassedComparator()` (Java method), 563
`shouldConstructorCreateANadirPointWithAllObjectiveValuesCorrectlyInitialized()` (Java method), 651
`shouldConstructorCreateAnArchiveWithTheRightCapacity()` (Java method), 559
`shouldConstructorCreateAnArranFrontFromAFileContainingA2DFront()` (Java method), 612
`shouldConstructorCreateAnArranFrontFromAFileContainingA3DFront()` (Java method), 612
`shouldConstructorCreateAnEmptyArchive()` (Java method), 559, 563
`shouldConstructorCreateAnIdealPointWithAllObjectiveValuesCorrectlyInitialized()` (Java method), 649
`shouldConstructorCreateAnInstanceIfTheParamtersAreValid()` (Java method), 630
`shouldConstructorCreateAnObject()` (Java method), 526
`shouldConstructorCreateASolutionAttributedWithThePassedIdentifier()` (Java method), 678
`shouldConstructorCreateAValidInstance()` (Java method), 543
`shouldConstructorCreateAValidSolution()` (Java method), 543
`shouldConstructorFailWhenPassedANegativeAlphaValue()` (Java method), 263
`shouldConstructorFailWhenPassedANegativeDistributionIndex()` (Java method), 275, 288, 294
`shouldConstructorFailWhenPassedANegativeProbabilityValue()` (Java method), 263, 275, 278, 284, 288, 294
`shouldConstructorRaiseAnExceptionIfFileNameIsNull()` (Java method), 472
`shouldConstructorRaiseAnExceptionIfTheParetoFrontIsNull()` (Java method), 472
`shouldConstructorRaiseAnExceptionIfTheWeightFileDoesNotExist()` (Java method), 639
`shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsEqual()` (Java method), 630
`shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsGreater()` (Java method), 630
`shouldConstructorThrowAnExceptionWhenTheNumberOfNeighboursIsNegative()` (Java method), 630
`shouldConstructorWithoutParameterAssignTheDefaultValues()` (Java method), 288, 294
`shouldConstructorWithProblemAndDistributionIndexParametersAssignTheDefaultValues()` (Java method), 288, 294
`shouldConvertFrontToArrayRaiseAnExceptionIfTheFrontIsNull()` (Java method), 615
`shouldConvertFrontToArrayReturnAnEmptyArrayIfTheFrontIsEmpty()` (Java method), 615
`shouldConvertFrontToArrayReturnTheCorrectArrayCaseA()` (Java method), 615
`shouldConvertFrontToArrayReturnTheCorrectArrayCaseB()` (Java method), 615
`shouldConvertFrontToSolutionListRaiseAnExceptionIfTheFrontIsNull()` (Java method), 616
`shouldConvertFrontToSolutionListReturnAnEmptyListIfTheFrontIsEmpty()` (Java method), 616
`shouldConvertFrontToSolutionListReturnTheCorrectListCaseA()` (Java method), 616
`shouldConvertFrontToSolutionListReturnTheCorrectListCaseB()` (Java method), 616
`shouldCopyConstructorCreateAnIdenticalObject()` (Java method), 526
`shouldCopyConstructorCreateAnIdenticalSolution()` (Java method), 526
`shouldCopyReturnACopyOfTheSolution()` (Java method), 526
`shouldCopyReturnAnIdenticalVariable()` (Java method), 526

shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneSolution() (Java method), 679	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOnePoint() (Java method), 654
shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneSolutionAndRaiseAnExceptionIfTheFrontIsNotCorrectlyCreated() (Java method), 612	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOnePointAndReturnASingleSolution() (Java method), 302, 306
shouldCreateAnArrayFrontFromAListOfSolutionsHavingOneSolutionAndRaiseAnExceptionIfTheFrontIsNotCorrectlyCreated() (Java method), 612	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOnePointAndSetTheNumberOfSolutionsToBeReturnedEqualsToTheNumberInTheList() (Java method), 303
shouldCreateAnArrayFrontFromAListOfSolutionsHavingTwoSolutions() (Java method), 612	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOnePointAndShouldRaiseAnExceptionIfTheFrontIsEmpty() (Java method), 555
shouldCreateAnArrayFrontFromAnEmptyFrontRaiseAnException() (Java method), 612	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithOnePointAndShouldRaiseAnExceptionIfTheFrontIsNull() (Java method), 555
shouldCreateAnArrayFrontFromAnEmptyListRaiseAnException() (Java method), 613	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoPoints() (Java method), 555
shouldCreateAnArrayFrontFromAnotherFrontResultInTwoEqualDistances() (Java method), 613	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoPointsAndShouldRaiseAnExceptionIfTheFrontIsEmpty() (Java method), 555
shouldCreateAnArrayFrontFromANullFrontRaiseAnException() (Java method), 612	shouldDistanceBetweenObjectivesWorkProperlyWithTwoSolutionsWithTwoPointsAndShouldRaiseAnExceptionIfTheFrontIsNull() (Java method), 555
shouldCreateAnArrayFrontFromANullListRaiseAnException() (Java method), 612	shouldDistanceToClosestPointRaiseAnExceptionIfTheFrontIsEmpty() (Java method), 616
shouldCreateAnArrayFrontFromASolutionListResultInTwoEqualDistances() (Java method), 612	shouldDistanceToClosestPointRaiseAnExceptionIfTheFrontIsNull() (Java method), 616
shouldCreateInitialPopulationWhenPopulationSizeIsBiggerThanZero() (Java method), 211	shouldDistanceToClosestPointRaiseAnExceptionIfThePointIsNull() (Java method), 616
shouldCreateInitialPopulationWhenPopulationSizeIsZero() (Java method), 211	shouldDistanceToClosestPointReturnMaxZeroIfThePointIsTheOnlyPointInTheList() (Java method), 616
shouldCreateInputStreamThrownAnExceptionIfFileDoesNotExist() (Java method), 613	shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasHasOnePoint() (Java method), 616
shouldCrossingTheBitInTheMiddleOfSecondVariableReturnsTheCorrectValue() (Java method), 278	shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasTwoPoints() (Java method), 616
shouldCrossingTheBitInTheMiddleOfTwoSingleVariableSolutionsReturnsTheCorrectValue() (Java method), 278	shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasTwoPointsAndShouldRaiseAnExceptionIfTheFrontIsEmpty() (Java method), 617
shouldCrossingTheFirstBitOfSecondVariableReturnTheCorrectValue() (Java method), 278	shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasTwoPointsAndShouldRaiseAnExceptionIfTheFrontIsNull() (Java method), 617
shouldCrossingTheFirstBitOfTwoSingleVariableSolutionsReturnsTheCorrectValue() (Java method), 278	shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasTwoPointsAndShouldRaiseAnExceptionIfTheFrontIsNull() (Java method), 617
shouldCrossingTheLastBitOfTwoSingleVariableSolutionsReturnsTheCorrectValue() (Java method), 279	shouldDistanceToClosestPointReturnTheCorrectValueIfTheFrontHasTwoPointsAndShouldRaiseAnExceptionIfTheFrontIsNull() (Java method), 617
shouldCrossingTheSecondVariableReturnTheOtherVariables() (Java method), 275	shouldDistanceToNearestPointRaiseAnExceptionIfThePointIsNull() (Java method), 617
shouldCrossingTwoDoubleVariableSolutionsReturnValidSolutions() (Java method), 263, 276	shouldDistanceToNearestPointReturnMaxDoubleIfThePointIsTheOnlyPointInTheList() (Java method), 617
shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutions() (Java method), 264, 276	shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasHasOnePoint() (Java method), 617
shouldCrossingTwoSingleVariableSolutionsReturnTheSameSolutions() (Java method), 264, 276	shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasTwoPoints() (Java method), 617
shouldCrossingTwoSingleVariableSolutionsReturnValidSolutions() (Java method), 264, 276	shouldDistanceToNearestPointReturnTheCorrectValueIfTheFrontHasTwoPointsAndShouldRaiseAnExceptionIfTheListSizeIsTooSmall() (Java method), 553
shouldCrossingTwoSingleVariableSolutionsWithSimilarValuesReturnTheSameSolutions() (Java method), 264, 276	shouldDimensionsOfTheFrontsIsDifferent() (Java method), 613
shouldCrossingTwoVariableSolutionsReturnTheSameSolutions() (Java method), 279	shouldDimensionsOfTheFrontsIsDifferent() (Java method), 613
shouldDefaultConstructorBeCorrectlyInitialized() (Java method), 639	shouldEqualsReturnFalseIfTheArgumentIsNull() (Java method), 613
shouldDefaultConstructorCreateAnEmptyArrayFront() (Java method), 613	shouldEqualsReturnFalseIfTheClassIsNotAPoint() (Java method), 647, 654
shouldDefaultConstructorCreateASolutionAttributedWithASolution() (Java method), 613	shouldEqualsReturnTrueIfTheClassIsAPoint() (Java method), 647, 654
shouldDefaultConstructorCreateASolutionAttributedWithASolution() (Java method), 613	shouldFrontHasADifferentNumberOfPoints() (Java method), 613

(Java method), 613

shouldEqualsReturnFalseIfTheFrontsAreDifferent() (Java method), 613

shouldEqualsReturnFalseIfThePointIsNull() (Java method), 647

shouldEqualsReturnFalseIfThePointsAreNotIdentical() (Java method), 647, 654

shouldEqualsReturnFalseIfTheSolutionIsNull() (Java method), 654

shouldEqualsReturnFalseIfTheTwoSolutionsHaveDifferentNumberOfObjectives() (Java method), 654

shouldEqualsReturnTrueIfTheArgumentIsEqual() (Java method), 613

shouldEqualsReturnTrueIfTheArgumentIsTheSameObject() (Java method), 613

shouldEqualsReturnTrueIfThePointsAreIdentical() (Java method), 648

shouldEqualsReturnTrueIfTheSolutionsAreIdentical() (Java method), 654

shouldEqualsReturnTrueIfTheTwoPointsAreTheSame() (Java method), 648, 655

shouldEvaluatePopulation() (Java method), 211

shouldEvaluateRaiseAnExceptionIfTheFrontApproximationIsNull() (Java method), 472

shouldEvaluateReturnTheCorrectValueCaseA() (Java method), 472

shouldEvaluateReturnTheCorrectValueCaseB() (Java method), 472

shouldEvaluateReturnZeroIfTheFrontAndTheReferenceFrontContainsTheSamePoints() (Java method), 473

shouldEvaluateWorkProperlyCase1() (Java method), 480, 482

shouldEvaluateWorkProperlyCase2() (Java method), 482

shouldEvaluateWorkProperlyCase3() (Java method), 482

shouldEvaluateWorkProperlyCase4() (Java method), 482

shouldExecuteFailIfTheListContainsMoreThanTwoSolutions() (Java method), 279

shouldExecuteFailIfTheListContainsOnlyOneSolution() (Java method), 279

shouldExecuteRaiseAnExceptionIfTheFirstFrontIsNull() (Java method), 476

shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNull() (Java method), 460, 462, 466, 468

shouldExecuteRaiseAnExceptionIfTheFrontApproximationIsNotValid() (Java method), 460

shouldExecuteRaiseAnExceptionIfTheIndexIsHigherThanTheSolutionListLength() (Java method), 300

shouldExecuteRaiseAnExceptionIfTheIndexIsNegative() (Java method), 300

shouldExecuteRaiseAnExceptionIfTheIndexIsNotIndicated() (Java method), 300

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsHasOneSolution() (Java method), 300

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsEmpty() (Java method), 298, 300, 304

shouldExecuteRaiseAnExceptionIfTheListOfSolutionsIsNull() (Java method), 299, 301, 304

shouldExecuteRaiseAnExceptionIfTheListSizeIsOneAndTwoSolutionsAreEqual() (Java method), 302, 304

shouldExecuteRaiseAnExceptionIfTheListSizeIsTwoAndFourSolutionsAreEqual() (Java method), 302

shouldExecuteRaiseAnExceptionIfTheParetoFrontApproximationListIsNull() (Java method), 462

shouldExecuteRaiseAnExceptionIfTheParetoFrontIsNull() (Java method), 466, 469, 476

shouldExecuteRaiseAnExceptionIfTheSolutionListIsEmpty() (Java method), 302, 306, 308

shouldExecuteRaiseAnExceptionIfTheSolutionListIsNull() (Java method), 302, 306, 308

shouldExecuteReturnAnElementIfTheListHasOneElement() (Java method), 308

shouldExecuteReturnAValidSolutionIsWithCorrectParameters() (Java method), 299, 304

shouldExecuteReturnOneIfTheFrontsContainADifferentPoint() (Java method), 462

shouldExecuteReturnOneIfTheSecondFrontIsEmpty() (Java method), 476

shouldExecuteReturnTheCorrectNumberOfSolutions() (Java method), 302

shouldExecuteReturnTheCorrectValueCaseA() (Java method), 460, 462, 476

shouldExecuteReturnTheCorrectValueCaseB() (Java method), 460, 476

shouldExecuteReturnTheRightValueIfTheFrontsContainOnePointWhichIsTheSame() (Java method), 460, 476

shouldExecuteReturnTheSameSolutionIfTheListContainsOneSolution() (Java method), 299, 304

shouldExecuteReturnTheSolutionInTheListIfTheListContainsASolution() (Java method), 302, 550

shouldExecuteReturnTheSolutionSInTheListIfTheListContainsTwoSolutions() (Java method), 302

shouldExecuteReturnThreeDifferentSolutionsIfTheListHasFourElements() (Java method), 301

shouldExecuteReturnTwoDifferentObjectsWhichAreEquals() (Java method), 272

shouldExecuteReturnTwoSolutionsIfTheListContainsTwoSolutions() (Java method), 299, 304

shouldExecuteReturnZeroIfBothFrontsAreEmpty() (Java method), 477

shouldExecuteReturnZeroIfTheFrontsContainOnePointWhichIsTheSame() (Java method), 460, 462, 477

shouldExecuteWithInvalidSolutionListSizeThrowAnException() (Java method), 264, 276

shouldExecuteWithNullParameterThrowAnException() (Java method), 264, 276, 279, 284, 288, 294

shouldExecuteWorkProperlyIfTheTwoSolutionsInTheListAreNondominated() (Java method), 299

shouldFillPopulationWithNewSolutionsDoNothingIfTheMaxSizeIsLowerThan() (Java method), 299

(Java method), 550

shouldFillPopulationWithNewSolutionsIncreaseTheListLengthToTheDimensionValueWithInvalidIndexesRaiseAnException()
(Java method), 550

shouldFindBestSolutionRaiseAnExceptionIfTheComparatorIsNull()
(Java method), 551

shouldFindBestSolutionRaiseAnExceptionIfTheSolutionListIsEmpty()
(Java method), 551

shouldFindBestSolutionRaiseAnExceptionIfTheSolutionListIsNull()
(Java method), 551

shouldFindBestSolutionReturnTheLastOneIfThisIsTheBestSolutionInTheListWithFiveSolutions()
(Java method), 551

shouldFindBestSolutionReturnTheSecondSolutionInTheListIfIsTheBestOneOfTheSolutions()
(Java method), 551

shouldFindBestSolutionReturnTheSolutionInTheListWhenItContainsOneSolution()
(Java method), 551

shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheComparatorIsNull()
(Java method), 551

shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheSolutionListIsEmpty()
(Java method), 551

shouldFindIndexOfBestSolutionRaiseAnExceptionIfTheSolutionListIsNull()
(Java method), 551

shouldFindIndexOfBestSolutionReturn4IfTheBestSolutionIsTheLastInAListWithFiveSolutions()
(Java method), 551

shouldFindIndexOfBestSolutionReturnOneIfTheSecondSolutionIsTheBestOneOfTheSolutionsInTheList()
(Java method), 552

shouldFindIndexOfBestSolutionReturnZeroIfTheFirstSolutionIsTheBestOneOfTheSolutionsInTheList()
(Java method), 552

shouldFindIndexOfBestSolutionReturnZeroIfTheListWhenItContainsOneSolution()
(Java method), 552

shouldFirstPointToCompareEqualsToNullRaiseAnException()
(Java method), 650, 652, 653, 656, 657

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheReferenceFrontIsNull()
(Java method), 620

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheReferenceSolutionInListIsNull()
(Java method), 620

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheVectorOfMaximumValuesIsNull()
(Java method), 620

shouldFrontNormalizerConstructorRaiseAnExceptionIsTheVectorOfMinimumValuesIsNull()
(Java method), 620

shouldFrontNormalizerConstructorRaiseAnExceptionTheDimensionOfTheMaximumAndMinimumArrayIsNotEqual()
(Java method), 621

shouldGetAlphaReturnTheRightValue()
(Java method), 264

shouldGetAttributeIdentifierReturnTheRightIdentifier()
(Java method), 679

shouldGetAttributeReturnNullIfTheSolutionHasNoAttribute()
(Java method), 679

shouldGetAttributeReturnTheAttributeValue()
(Java method), 679

shouldGetAverageOccupationReturnTheRightValue()
(Java method), 543

shouldGetAverageOccupationReturnZeroIfThereAreNoOccupiedHyperplanes()
(Java method), 543

shouldGetDimensionValueReturnTheCorrectValue()
(Java method), 648

shouldGetDimensionValueWithInvalidIndexesRaiseAnException()
(Java method), 648

shouldGetDistributionIndexReturnTheRightValue()
(Java method), 276, 288, 294

shouldGetEvaluations()
(Java method), 211

shouldGetInvertedFrontRaiseAnExceptionIfTheFrontIsEmpty()
(Java method), 618

shouldGetInvertedFrontRaiseAnExceptionIfTheFrontIsNull()
(Java method), 618

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfFourPoints()
(Java method), 618

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCase1()
(Java method), 618

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCase2()
(Java method), 618

shouldGetInvertedFrontReturnTheCorrectFrontIfItComposedOfOnePointCase3()
(Java method), 618

shouldGetLowerBoundReturnTheRightValue()
(Java method), 526

shouldGetMaximumValuesRaiseAnExceptionIfTheFrontIsEmpty()
(Java method), 618

shouldGetMaximumValuesRaiseAnExceptionIfTheFrontIsNull()
(Java method), 618

shouldGetMaximumValuesWithAFrontWithOnePointReturnTheCorrectValue()
(Java method), 618

shouldGetMaximumValuesWithAFrontWithThreePointReturnTheCorrectValue()
(Java method), 618

shouldGetMinimumValuesRaiseAnExceptionIfTheFrontIsEmpty()
(Java method), 618

shouldGetMinimumValuesRaiseAnExceptionIfTheFrontIsNull()
(Java method), 618

shouldGetMinimumValuesWithAFrontWithOnePointReturnTheCorrectValue()
(Java method), 618

shouldGetMinimumValuesWithAFrontWithThreePointReturnTheCorrectValue()
(Java method), 618

shouldGetMutationProbabilityReturnTheRightValue()
(Java method), 284, 288, 294

shouldGetNameReturnTheCorrectValue()
(Java method), 466, 467

shouldGetNeighborsReturnFourNeighborsCase1()
(Java method), 632, 635, 638, 640

shouldGetNeighborsReturnFourNeighborsCase10()
(Java method), 632, 635

shouldGetNeighborsReturnFourNeighborsCase11()
(Java method), 632, 635

shouldGetNeighborsReturnFourNeighborsCase12()
(Java method), 636

shouldGetNeighborsReturnFourNeighborsCase2()
(Java method), 633, 636, 638, 640

shouldGetNeighborsReturnFourNeighborsCase3()
(Java method), 633, 636, 638, 640

shouldGetNeighborsReturnFourNeighborsCase4()
(Java method), 633, 636, 638, 641

shouldGetNeighborsReturnFourNeighborsCase5() (Java method), 633, 636, 641	shouldGetPointRaiseAnExceptionWhenTheIndexIsGreaterThanTheFrontSize() (Java method), 614
shouldGetNeighborsReturnFourNeighborsCase6() (Java method), 633, 636, 641	shouldGetPointRaiseAnExceptionWhenTheIndexIsNegative() (Java method), 614
shouldGetNeighborsReturnFourNeighborsCase7() (Java method), 633, 636, 641	shouldGetPointReturnTheCorrectObject() (Java method), 614
shouldGetNeighborsReturnFourNeighborsCase8() (Java method), 633, 636, 641	shouldGetProbabilityReturnTheRightValue() (Java method), 264, 276
shouldGetNeighborsReturnFourNeighborsCase9() (Java method), 633, 636	shouldGetResultReturnsThenReturnTheBestIndividual() (Java method), 212
shouldGetNeighborsReturnThreeNeighborsPlusTheCurrentSolution() (Java method), 630	shouldGetTotalNumberOfBitsBeEqualToTheSumOfBitsPerVariable() (Java method), 527
shouldGetNeighborsReturnTwoNeighborsPlusTheCurrentSolution() (Java method), 630	shouldGetUpperBoundReturnTheRightValue() (Java method), 526
shouldGetNeighborsThrowAnExceptionIfTheListSizeIsNotCorrect() (Java method), 631	shouldGetValuesReturnTheCorrectValues() (Java method), 648
shouldGetNeighborsWithANegativeSolutionIndexThrowAnException() (Java method), 631, 641	shouldGetVariableValueStringReturnARightStringRepresentation() (Java method), 527
shouldGetNeighborsWithAnEmptyListOfSolutionsThrowAnException() (Java method), 641	shouldHasACodeReturnTheCorrectValue() (Java method), 648, 655
shouldGetNeighborsWithANullListOfSolutionsThrowAnException() (Java method), 631, 641	shouldIdenticalPointsHaveADistanceOfOne() (Java method), 590
shouldGetNeighborsWithASolutionIndexValueEqualToTheListSizeThrowAnException() (Java method), 641	shouldInitThrowsAnExceptionRaiseAnException() (Java method), 653
shouldGetNeighborsWithASolutionIndexValueGreaterThanTheListSizeThrowAnException() (Java method), 641	shouldInitThrowsAnExceptionFirstPointDimensionsRaiseAnException() (Java method), 653
shouldGetNeighborsWithATooBigSolutionIndexThrowAnException() (Java method), 631	shouldInitThrowsAnExceptionSecondPointDimensionsRaiseAnException() (Java method), 653
shouldGetNeighborsWorkProperlyCaseA() (Java method), 634	shouldInitializationPhaseLeadToAPopulationFilledWithEvaluatedSolutions() (Java method), 36
shouldGetNeighborsWorkProperlyCaseB() (Java method), 634	shouldInitProgress() (Java method), 212
shouldGetNeighborsWorkProperlyCaseC() (Java method), 634	shouldIsStoppingConditionReachedReturnFalseIfTheConditionDoesNotFulfill() (Java method), 36
shouldGetNeighborsWorkProperlyCaseD() (Java method), 634	shouldIsStoppingConditionReachedReturnTrueIfTheConditionFulfills() (Java method), 36
shouldGetNeighborsWorkProperlyCaseE() (Java method), 635	shouldIsStoppingConditionReachedWhenEvaluationsBiggerThanMaxEvaluation() (Java method), 212
shouldGetNeighborsWorkProperlyCaseF() (Java method), 635	shouldIsStoppingConditionReachedWhenEvaluationsEqualToMaxEvaluation() (Java method), 212
shouldGetNeighborsWorksProperlyWithTwoObjectives() (Java method), 639	shouldIsStoppingConditionReachedWhenEvaluationsLesserThanMaxEvaluation() (Java method), 212
shouldGetNormalizedFrontReturnTheCorrectFrontIfTheSolutionListContainsThePoints() (Java method), 621	shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorNotBeUsed() (Java method), 543
shouldGetNormalizedFrontReturnTheCorrectFrontIfThisCostFunctionIsNotUsed() (Java method), 621	shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorNotBeUsed() (Java method), 543
shouldGetNumberOfBitsBeEqualToTheNumberOfOfBitsPerVariable() (Java method), 527	shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorNotBeUsed() (Java method), 543
shouldGetNumberOfDimensionsReturnTheCorrectValue() (Java method), 648	shouldJMetalRandomGeneratorNotBeUsedWhenCustomRandomGeneratorNotBeUsed() (Java method), 552
shouldGetNumberOfObjectivesReturnTheCorrectValue() (Java method), 655	shouldJoinAnEmptyArchiveProduceAnArchiveWithTheSameSolutions() (Java method), 563
shouldGetObjectiveReturnTheCorrectValue() (Java method), 655	shouldJoinTwoEmptyArchivesReturnAnEmptyArchive() (Java method), 563

shouldJoinWithAnEmptyArchivesRemainTheArchiveWithTheSameNumberOfSolutions() (Java method), 563
 shouldMutateASingleVariableSolutionReturnAnotherValidSolution() (Java method), 288, 295
 shouldMutateASingleVariableSolutionReturnAValidSolution() (Java method), 288, 295
 shouldMutateASingleVariableSolutionReturnTheSameSolutionIfNotModified() (Java method), 288, 295
 shouldMutateASingleVariableSolutionReturnTheSameSolutionIfNotModifiedWithALineContainingWrongDataRaiseAnException() (Java method), 288, 295
 shouldMutateASingleVariableSolutionReturnTheSameSolutionIfNotModifiedWithALineWithALineMissingDataRaiseAnException() (Java method), 284
 shouldMutateASingleVariableSolutionReturnTheSameSolutionIfNotModifiedWithALineWithALineMissingDataRaiseAnException() (Java method), 289, 295
 shouldMutateASingleVariableSolutionWhenASingleBitIsMutated() (Java method), 284
 shouldMutateASingleVariableSolutionWithSameLowerAndUpperBounds() (Java method), 289, 295
 shouldMutateATwoVariableSolutionReturnTheSameSolutionIfNotModified() (Java method), 284
 shouldMutateATwoVariableSolutionWhenTwoBitsAreMutated() (Java method), 285
 shouldNonDefaultConstructorReturnTheCorrectNumberOfSolutions() (Java method), 302, 306
 shouldNormalizeRaiseAnExceptionIfTheFrontIsEmpty() (Java method), 621
 shouldNormalizeRaiseAnExceptionIfTheMaxAndMinValuesAreTheSame() (Java method), 621
 shouldNormalizeRaiseAnExceptionIfTheSolutionListIsEmpty() (Java method), 621
 shouldNormalizeRaiseAnExceptionTheDimensionOfTheMaxAndMinValuesAreTheSame() (Java method), 621
 shouldNormalizeRaiseAnExceptionTheFrontIsNull() (Java method), 621
 shouldNormalizeRaiseAnExceptionTheSolutionListIsNull() (Java method), 621
 shouldNormalizeReturnTheCorrectFrontIfThisContainsOnePoint() (Java method), 621
 shouldOccupiedHypercubesReturnTheNumberOfOccupiedHypercubes() (Java method), 544
 shouldOccupiedHypercubesReturnZeroIfThereAreNotOccupiedHypercubes() (Java method), 544
 shouldPassingPointsWithDifferentDimensionsRaiseAnException() (Java method), 656, 657
 shouldPointsInTheSameDirectionHaveADistanceOfOne() (Java method), 591
 shouldProneDoNothingIfTheArchiveIsEmpty() (Java method), 559
 shouldRankingOfAPopulationWithFiveSolutionsWorkProperly() (Java method), 677
 shouldRankingOfAPopulationWithThreeDominatedSolutionsReturnTheCorrectFront() (Java method), 677
 shouldRankingOfAPopulationWithTwoDominatedSolutionsReturnTwoSubfronts() (Java method), 677
 shouldRankingOfAPopulationWithTwoNonDominatedSolutionsReturnOneSubfront() (Java method), 677
 shouldReadFrontAnEmptyFileCreateAnEmptyFront() (Java method), 614
 shouldReadFrontFourPointsCreateTheCorrectFront() (Java method), 614
 shouldReadFrontFromAFile() (Java method), 614
 shouldReadFrontFromAFileWithALineContainingWrongDataRaiseAnException() (Java method), 614
 shouldReadFrontFromAFileWithALineWithALineMissingDataRaiseAnException() (Java method), 614
 shouldReduceReferenceSetSize() (Java method), 36
 shouldReferenceSetUpdateCreateTheTwoRefSetsAfterBeingInvokedTheFirstTime() (Java method), 36
 shouldReproduction() (Java method), 212
 shouldRestartRemoveTheRequestedPercentageOfSolutions() (Java method), 552
 shouldRepairDoubleSolutionAtBoundsAssignTheLowerBoundIfValueIsLower() (Java method), 538
 shouldRepairDoubleSolutionAtBoundsAssignTheUpperBoundIfValueIsGreater() (Java method), 538
 shouldRepairDoubleSolutionAtBoundsRaiseAnExceptionIfTheBoundsAreInvalid() (Java method), 538
 shouldRepairDoubleSolutionAtRandomAssignARandomValueIfValueIsGreater() (Java method), 539
 shouldRepairDoubleSolutionAtRandomAssignARandomValueIfValueIsLower() (Java method), 539
 shouldRepairDoubleSolutionAtRandomRaiseAnExceptionIfTheBoundsAreInvalid() (Java method), 539
 shouldSecondPointToCompareEqualsToNullRaiseAnException() (Java method), 650, 652, 653, 656, 657
 shouldSelectNRandomDifferentSolutions() (Java method), 212
 shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheListSizeIsTooSmall() (Java method), 552
 shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheSolutionListIsEmpty() (Java method), 552
 shouldSelectNRandomDifferentSolutionsRaiseAnExceptionIfTheSolutionListIsTooSmall() (Java method), 552
 shouldSelectNRandomDifferentSolutionsReturnASingleSolution() (Java method), 552
 shouldSelectNRandomDifferentSolutionsReturnTheCorrectListOfSolutions() (Java method), 303, 552
 shouldSelectNRandomDifferentSolutionsReturnTheCorrectNumberOfSolutions() (Java method), 553
 shouldSelectNRandomDifferentSolutionsReturnTheSolutionSInTheListIfTheListSizeIsTooSmall() (Java method), 553
 shouldSetAttributeAssignTheAttributeValueToTheSolution() (Java method), 679
 shouldSetDimensionValueAssignTheCorrectValue() (Java method), 648
 shouldSetDimensionValueWithInvalidIndexesRaiseAnException() (Java method), 648

- (Java method), 648
- shouldSetEvaluations() (Java method), 212
- shouldSetObjectiveAssignTheTheCorrectValue() (Java method), 655
- shouldSetPointAssignTheCorrectObject() (Java method), 614
- shouldSetPointRaiseAnExceptionWhenTheIndexIsGreaterThanTheFrontSize() (Java method), 614
- shouldSetPointRaiseAnExceptionWhenTheIndexIsNegative() (Java method), 615
- shouldSetPointRaiseAnExceptionWhenThePointIsNull() (Java method), 615
- shouldSolutionCombinationProduceTheRightNumberOfSolutions() (Java method), 651
- shouldSolutionListsAreEqualsReturnIfTwoIdenticalSolutionListsAreCompared() (Java method), 553
- shouldSolutionListsAreEqualsReturnIfTwoSolutionListsWithIdenticalSolutionsAreCompared() (Java method), 553
- shouldSortReturnAnOrderedFront() (Java method), 615
- shouldSubsetGenerationProduceAnEmptyListIfAllTheSolutionsAreMarked() (Java method), 37
- shouldTheAlgorithmReturnAGoodQualityFrontWhenSolvingAComplexProblem() (Java method), 141, 178
- shouldTheAlgorithmReturnAnExceptionIfIndicatingANonExistingDesignEntity() (Java method), 197
- shouldTheAlgorithmReturnANumberOfSolutionsWhenSolvingASimpleProblem() (Java method), 35, 37, 59, 80, 92, 115, 133, 141, 157, 179, 185, 190, 197
- shouldTheAlgorithmReturnTheCorrectSolutionWhenSolvingASimpleProblem() (Java method), 221, 224
- shouldTheCrowdingDistanceOfAnEmptySetDoNothing() (Java method), 675
- shouldTheCrowdingDistanceOfASingleSolutionBeInfinity() (Java method), 675
- shouldTheCrowdingDistanceOfThreeSolutionsCorrectlyAssign() (Java method), 676
- shouldTheCrowdingDistanceOfTwoSolutionsBeInfinity() (Java method), 676
- shouldTheHashCodeOfTwoIdenticalSolutionsBeTheSame() (Java method), 527
- shouldTheHypervolumeHaveAMinimumValue() (Java method), 190
- shouldTheHypervolumeHaveAMinimumValue() (Java method), 35, 59, 80, 93, 115, 133, 157, 179, 185, 197
- shouldTheRankingOfAnEmptyPopulationReturnOneSubfront() (Java method), 677
- shouldTheRankingOfAnEmptyPopulationReturnZeroSubfront() (Java method), 677
- shouldTheSumOfGetNumberOfBitsBeEqualToTheSumOfBitsOfParents() (Java method), 528
- shouldTwoPerpendicularPointsHaveADistanceOfZero() (Java method), 591
- shouldUpdateAListOfSolutionsLeadToTheCorrectNadirPoint() (Java method), 651
- shouldUpdateProgressWhenAnyIteration() (Java method), 212
- shouldUpdateProgressWhenFirstIteration() (Java method), 212
- shouldUpdateWithOneSolutionMakeTheIdealPointHaveTheSolutionValues() (Java method), 651
- shouldUpdateWithOneSolutionMakeTheNadirPointHaveTheSolutionValues() (Java method), 651
- shouldUpdateWithThreeSolutionsLeadToTheCorrectIdealPoint() (Java method), 649
- shouldUpdateWithThreeSolutionsLeadToTheCorrectNadirPoint() (Java method), 649
- shouldUpdateWithTwoSolutionsLeadToTheCorrectIdealPoint() (Java method), 649
- shouldUpdateWithTwoSolutionsLeadToTheCorrectNadirPoint() (Java method), 649
- shutdown() (Java method), 592, 593
- sigma (Java field), 456
- signsAreMarked() (Java method), 40, 47
- SimpleDescribedEntity (Java class), 626
- SimpleDescribedEntity() (Java constructor), 627
- SimpleDescribedEntity(String) (Java constructor), 627
- SimpleDescribedEntity(String, String) (Java constructor), 627
- SimpleDuplicateEntityTest (Java class), 628
- SimpleMeasure (Java class), 254
- SimpleMeasure() (Java constructor), 254
- SimpleMeasure(String) (Java constructor), 254
- SimpleMeasure(String, String) (Java constructor), 254
- SimpleMeasureManager (Java class), 254
- SimpleMeasureManagerTest (Java class), 256
- SimplePullMeasure (Java class), 257
- SimplePullMeasure() (Java constructor), 258
- SimplePullMeasure(String) (Java constructor), 258
- SimplePullMeasure(String, String) (Java constructor), 258
- SimplePushMeasure (Java class), 258
- SimplePushMeasure() (Java constructor), 259
- SimplePushMeasure(String) (Java constructor), 259
- SimplePushMeasure(String, String) (Java constructor), 258
- SimplePushMeasureTest (Java class), 259
- SimpleRandomMutation (Java class), 295
- SimpleRandomMutation(double) (Java constructor), 295
- SimpleRandomMutation(double, RandomGenerator) (Java constructor), 295
- SimpleRandomMutationTest (Java class), 296
- simpleTest() (Java method), 482
- SimplePointCrossover (Java class), 277
- SimplePointCrossover(double) (Java constructor), 277
- SimplePointCrossover(double, RandomGenerator) (Java constructor), 277
- SimplePointCrossover(double, RandomGenerator, RandomGenerator) (Java constructor), 277

- BoundedRandomGenerator) (Java constructor), 277
- SinglePointCrossoverTest (Java class), 278
- size() (Java method), 556, 558, 562, 625
- slimDetail_ (Java field), 40
- sLinear(float, float) (Java method), 406
- SMPSO (Java class), 171
- SMPSO (Java field), 178
- SMPSO(DoubleProblem, int, BoundedArchive, MutationOperator, int, double, double, double, double, double, double, double, double, double, double, SolutionListEvaluator) (Java constructor), 171
- SMPSOBigDataRunner (Java class), 503
- SMPSOBuilder (Java class), 174
- SMPSOBuilder(DoubleProblem, BoundedArchive) (Java constructor), 174
- SMPSOOhv2IT (Java class), 184
- SMPSOOhv2Runner (Java class), 504
- SMPSOOhvIT (Java class), 185
- SMPSOOhvRunner (Java class), 504
- SMPSOIT (Java class), 178
- SMPSOMeasures (Java class), 179
- SMPSOMeasures(DoubleProblem, int, BoundedArchive, MutationOperator, int, double, double, double, double, double, double, double, double, double, double, SolutionListEvaluator) (Java constructor), 179
- SMPSOMeasuresRunner (Java class), 505
- SMPSOMeasuresWithChartsRunner (Java class), 505
- SMPSORP (Java class), 181
- SMPSORP(DoubleProblem, int, List, List, MutationOperator, int, double, double, double, double, double, double, double, double, double, double, SolutionListEvaluator) (Java constructor), 182
- SMPSORPChangingTheReferencePointsAndChartsRunner (Java class), 505
- SMPSORPWithMultipleReferencePointsAndChartsRunner (Java class), 506
- SMPSORPWithMultipleReferencePointsRunner (Java class), 506
- SMPSORPWithOneReferencePointRunner (Java class), 507
- SMPSORRunner (Java class), 507, 514
- SMPSOVariant (Java enum), 178
- SMSEMOA (Java class), 185
- SMSEMOA(Problem, int, int, double, CrossoverOperator, MutationOperator, SelectionOperator, Comparator, Hypervolume) (Java constructor), 186
- SMSEMOABuilder (Java class), 187
- SMSEMOABuilder(Problem, CrossoverOperator, MutationOperator) (Java constructor), 188
- SMSEMOAIT (Java class), 190
- SMSEMOARunner (Java class), 507
- sMulti(float, int, int, float) (Java method), 406
- solution (Java field), 245
- Solution (Java interface), 518
- SolutionAttribute (Java interface), 674
- SolutionBuilder (Java interface), 519
- solutionCombination(List) (Java method), 27, 32
- SolutionEvaluator (Java interface), 520
- solutionFitness (Java field), 83
- SolutionListEvaluator (Java interface), 592
- SolutionListExtremeValues (Java class), 605
- solutionListMeasure (Java field), 60, 142, 179, 182, 198
- SolutionListOutput (Java class), 606
- SolutionListOutput(List) (Java constructor), 606
- solutionListsAreEquals(List, List) (Java method), 550
- SolutionListUtils (Java class), 546
- SolutionListUtilsTest (Java class), 550
- SolutionUtils (Java class), 553
- SolutionUtilsTest (Java class), 554
- sort(Comparator) (Java method), 609, 611
- sortByDensityEstimator() (Java method), 69, 556, 559–561, 565
- SPEA2 (Java class), 190
- SPEA2(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator) (Java constructor), 191
- SPEA2BinaryRunner (Java class), 508
- SPEA2Builder (Java class), 192
- SPEA2Builder(Problem, CrossoverOperator, MutationOperator) (Java constructor), 193
- SPEA2Runner (Java class), 508
- specialPopulation (Java field), 40
- SPECIFIC_WEIGHT (Java field), 355
- specificMOEAComputations() (Java method), 121, 123, 197
- Sphere (Java class), 427
- Sphere() (Java constructor), 428
- Sphere(double[]) (Java method), 399
- sphere(double[]) (Java method), 433
- Sphere(Integer) (Java constructor), 428
- sphere_noise(double[]) (Java method), 433
- spPopulationOrder (Java field), 40
- Spread (Java class), 477
- Spread() (Java constructor), 477
- Spread(Front) (Java constructor), 477
- spread(Front, Front) (Java method), 478
- Spread(String) (Java constructor), 477
- Srinivas (Java class), 324
- Srinivas() (Java constructor), 324
- stableMatching(int[][], int[][], int, int) (Java method), 117
- StandardPSO2007 (Java class), 225
- StandardPSO2007(DoubleProblem, int, int, int, int, SolutionListEvaluator) (Java constructor), 225

- StandardPSO2007(DoubleProblem, int, int, int, SolutionListEvaluator) (Java constructor), 225
- StandardPSO2007Runner (Java class), 514
- StandardPSO2011 (Java class), 227
- StandardPSO2011(DoubleProblem, int, int, int, int, SolutionListEvaluator) (Java constructor), 227
- StandardPSO2011(DoubleProblem, int, int, int, SolutionListEvaluator) (Java constructor), 228
- StandardPSO2011Runner (Java class), 514
- start() (Java method), 243
- startup() (Java method), 141, 211, 213, 650
- std(List) (Java method), 129
- STEADY_STATE (Java field), 223
- SteadyStateGeneticAlgorithm (Java class), 223
- SteadyStateGeneticAlgorithm(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator) (Java constructor), 223
- SteadyStateGeneticAlgorithmBinaryEncodingRunner (Java class), 515
- SteadyStateGeneticAlgorithmRunner (Java class), 515
- SteadyStateGeneticAlgorithmTestIT (Java class), 224
- SteadyStateMOCeII (Java field), 92
- SteadyStateNSGAI (Java class), 144
- SteadyStateNSGAI (Java field), 139, 170
- SteadyStateNSGAI(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, Comparator, SolutionListEvaluator) (Java constructor), 145
- SteadyStateNSGAIIIRunner (Java class), 509
- stmSelection() (Java method), 118
- stop() (Java method), 243
- STRAIN_COMPRESS (Java field), 355
- STRAIN_CUT (Java field), 355
- STRAIN_TRACTION (Java field), 355
- strainAdmissibleCut_ (Java field), 363
- StrainCutMax_ (Java field), 356
- Straini(int, int, int) (Java method), 370
- Straini_ (Java field), 357
- Strainj_ (Java field), 357
- StrainMax_ (Java field), 356
- StrainMin_ (Java field), 356
- StrainMxyMax_ (Java field), 356
- StrainMxyMin_ (Java field), 356
- StrainMxzMax_ (Java field), 356
- StrainMxzMin_ (Java field), 356
- StrainNxxMax_ (Java field), 357
- StrainNxxMin_ (Java field), 357
- StrainResidualCut_ (Java field), 357
- StrainResidualMax_ (Java field), 357
- StrainResidualMin_ (Java field), 357
- strenghtRawFitness (Java field), 191
- StrengthFitnessComparator (Java class), 588
- StrengthRawFitness (Java class), 680
- strengthRawFitness (Java field), 29
- STRESS (Java field), 356
- STRESS_CUT (Java field), 356
- subfrontFillsIntoThePopulation(Ranking, int, List) (Java method), 78, 137, 306, 307
- sublen14 (Java field), 397
- sublen15 (Java field), 398
- subP (Java field), 40
- subPNum (Java field), 40
- subproblem (Java field), 40
- subproblemNum_ (Java field), 40, 47
- subproblemSortI() (Java method), 45
- subregionDist (Java field), 110
- subregionIdx (Java field), 110
- subsetGeneration() (Java method), 27, 32
- subVector(float[], int, int) (Java method), 408
- SUM_OF_OBJECTIVES (Java field), 74
- sumFitness(int) (Java method), 113
- sumOfFrequencyValues (Java field), 29
- sumOfObjectives(List) (Java method), 71
- sumOfReverseFrequencyValues (Java field), 30
- supplyBadSolution() (Java method), 45
- swarmSize (Java field), 52, 182
- ## T
- t1(float[], int) (Java method), 409–411, 413–418
- t2(float[], int) (Java method), 409–411, 416–418
- t2(float[], int, int) (Java method), 413–415
- t3(float[]) (Java method), 409
- t3(float[], int, int) (Java method), 410, 412, 416, 417, 419
- t4(float[], int, int) (Java method), 409
- t_ (Java field), 40, 47
- T_DOUBLE (Java field), 357
- T_SINGLE (Java field), 357
- Tanaka (Java class), 325
- Tanaka() (Java constructor), 325
- TCHE (Java field), 54, 103
- TchebycheffUtilityFunctionsSet (Java class), 132
- TchebycheffUtilityFunctionsSet(String) (Java constructor), 132
- TchebycheffUtilityFunctionsSet(String, List) (Java constructor), 132
- team (Java field), 41
- tearDown() (Java method), 155, 299, 528
- teardown() (Java method), 586
- tempBorder (Java field), 41
- test(S, S, AdaptiveGridArchive) (Java method), 160
- test_func_arg_types (Java field), 430
- test_func_class_names (Java field), 430
- testAddingDifferentEntityWithDifferentNameProperlyAdds() (Java method), 626
- testAddingDifferentEntityWithSameNameThrowsException() (Java method), 626
- testAddingSameEntityModifiesNothing() (Java method), 626

- testNoRunningRemainsAtZero() (Java method), 244
 - testNotifiedWhenRegistered() (Java method), 259
 - testNotNotifiedWhenUnregistered() (Java method), 259
 - testNullDescriptionWhenNoDescription() (Java method), 628
 - testR2HasProperNameAndDescriptionWithEmptyConstructor() (Java method), 474
 - testR2HasProperNameAndDescriptionWithFileConstructor() (Java method), 474
 - testR2HasProperNameAndDescriptionWithFileFrontConstructor() (Java method), 474
 - testR2HasProperNameAndDescriptionWithFrontConstructor() (Java method), 474
 - testR2HasProperNameAndDescriptionWithVectorConstructor() (Java method), 475
 - testR2HasProperNameAndDescriptionWithVectorFrontConstructor() (Java method), 475
 - testRemoveBothAtOnce() (Java method), 257
 - testRemoveBothMeasuresRemoveKey() (Java method), 257
 - testRemoveMultipleMeasures() (Java method), 257
 - testReset() (Java method), 242
 - testResetGoBackToZeroWhenStopped() (Java method), 244
 - testResetNotificationsOccur() (Java method), 242
 - testResetRestartFromZeroWhenRunning() (Java method), 244
 - testResetToAGivenValue() (Java method), 242
 - testResetToCurrentTimeWhenListenerIsRunning() (Java method), 248
 - testResetToZeroWhenNoListenerIsRunning() (Java method), 248
 - testRetrieveObjectiveNames() (Java method), 534
 - testRetrieveVariableNames() (Java method), 536
 - testReturnSameWrapperForSameListener() (Java method), 248
 - testReturnSameWrapperForSameMeasure() (Java method), 248
 - testSameNameAndDescriptionThanOriginalMeasure() (Java method), 248
 - testSetGetDescription() (Java method), 628
 - testSetGetName() (Java method), 628
 - testSetGetPullMeasure() (Java method), 257
 - testSetGetPushMeasure() (Java method), 257
 - testSetMeasureGetBoth() (Java method), 257
 - testSpecializedPushActLikeParentPush() (Java method), 245
 - testStartEmpty() (Java method), 257
 - testToStringInCaseInsensitiveOrder() (Java method), 626
 - testUnboundedDoubleToIntegerFactoryMethodReturnsGenerator() (Java method), 662
 - testUnboundedDoubleToIntegerFactoryMethodReturnsGeneratorWithCorrectValues() (Java method), 662
 - testUnlinkCorrectlyIgnored() (Java method), 242
 - toArray() (Java method), 626
 - toArray(T[]) (Java method), 626
 - tolerance (Java field), 569
 - toString() (Java method), 523, 542, 611, 626, 628, 644, 646
 - TOURNAMENTS_ROUNDS (Java field), 82
 - TournamentSelection (Java class), 307
 - TournamentSelection(Comparator, int) (Java constructor), 307
 - TournamentSelection(int) (Java constructor), 307
 - TournamentSelectionTest (Java class), 308
 - OurSelection(int) (Java method), 114, 118
 - ttl2(int, double[], double[], double[][]) (Java method), 219
 - TRADEOFF_UTILITY (Java field), 74
 - tradeoffUtility(List) (Java method), 71
 - Transformations (Java class), 405
 - translateObjectives(List) (Java method), 150
 - tred2(int, double[][], double[], double[]) (Java method), 219
 - TSP (Java class), 428
 - TSP(String) (Java constructor), 428
 - TwoDimensionalMesh (Java class), 639
 - TwoDimensionalMesh(int, int, int[][]) (Java constructor), 640
 - TwoDimensionalMeshTest (Java class), 640
 - TwoPointCrossover (Java class), 279
 - TwoPointCrossover(double) (Java constructor), 279
 - TypeMaterial_ (Java field), 357
- ## U
- UF1 (Java class), 328
 - UF1() (Java constructor), 328
 - UF1(int) (Java constructor), 328
 - UF10 (Java class), 328
 - UF10() (Java constructor), 329
 - UF10(int) (Java constructor), 329
 - UF2 (Java class), 329
 - UF2() (Java constructor), 329
 - UF2(int) (Java constructor), 329
 - UF3 (Java class), 330
 - UF3() (Java constructor), 330
 - UF3(int) (Java constructor), 330
 - UF4 (Java class), 330
 - UF4() (Java constructor), 330
 - UF4(Integer) (Java constructor), 330
 - UF5 (Java class), 331
 - UF5() (Java constructor), 331
 - UF5(int, int, double) (Java constructor), 331
 - UF6 (Java class), 332
 - UF6() (Java constructor), 332
 - UF6(Integer, int, double) (Java constructor), 332
 - UF7 (Java class), 332
 - UF7() (Java constructor), 333

UF7(int) (Java constructor), 333
 UF8 (Java class), 333
 UF8() (Java constructor), 333
 UF8(int) (Java constructor), 333
 UF9 (Java class), 334
 UF9() (Java constructor), 334
 UF9(int, double) (Java constructor), 334
 UNIFORM (Java field), 74
 uniform(List) (Java method), 71
 UniformMutation (Java class), 296
 UniformMutation(double, double) (Java constructor), 296
 UniformMutation(double, double, RandomGenerator) (Java constructor), 296
 UniformMutationTest (Java class), 297
 unlink(PushMeasure) (Java method), 241
 unregister(MeasureListener) (Java method), 238, 253, 259
 update(double[]) (Java method), 642, 646, 649, 651
 update(List) (Java method), 649, 651
 updateArchive(S) (Java method), 113
 updateBorder() (Java method), 45
 updateFrontCharts(List) (Java method), 575, 577
 updateGrid(List) (Java method), 543
 updateGrid(S, List) (Java method), 543
 updateIdealPoint(S) (Java method), 45
 updateIndicatorChart(String, Double) (Java method), 575
 updateLeaders(List) (Java method), 25, 155, 173, 184, 227, 229
 updateLeadersDensityEstimator() (Java method), 173, 184
 updateMax(List) (Java method), 125
 updateNadirPoint() (Java method), 45
 updateNadirPoint(List) (Java method), 121
 updateNadirPoint(S) (Java method), 121
 updateNeighborhood() (Java method), 45
 updateNeighborhood(DoubleSolution, int, NeighborType) (Java method), 104
 updateNeighborhood(S, int, NeighborType) (Java method), 102
 updateParetoOptimal(List, List) (Java method), 570
 updateParetoOptimal(R, List) (Java method), 567
 updateParticlesMemory(List) (Java method), 25, 155, 173, 184, 227, 229
 updatePointOfInterest(List) (Java method), 14, 127, 167, 170, 197, 680
 updatePosition(List) (Java method), 25, 155, 173, 184, 227, 229
 updateProgress() (Java method), 18, 22, 25, 51, 54, 61, 63, 78, 89, 94, 121, 135, 143–145, 147, 155, 160, 163, 167, 173, 181, 184, 187, 192, 199, 203, 208, 214, 216, 219, 221, 224, 227, 229, 567, 571
 updateReferencePoint(List) (Java method), 122, 125, 577
 updateReferencePoint(S) (Java method), 122

updateThreshold(List) (Java method), 589
 updateVelocity(int) (Java method), 54
 updateVelocity(List) (Java method), 25, 155, 173, 184, 227, 230
 upper (Java field), 668
 UPPERLIMIT (Java field), 327
 utility (Java field), 114, 116, 132
 utilityFunction() (Java method), 114, 118
 utilityFunctions (Java field), 122
 uY_ (Java field), 363

V

VAL1 (Java field), 666
 VAL2 (Java field), 666
 VAL3 (Java field), 666
 validate(double[][], int) (Java method), 201
 value (Java field), 245
 VAR_eY_LOWER_LIMIT (Java field), 358
 VAR_eY_UPPER_LIMIT (Java field), 358
 VAR_eZ_LOWER_LIMIT (Java field), 358
 VAR_eZ_UPPER_LIMIT (Java field), 358
 VAR_POSITION (Java field), 358
 VAR_Y_LOWER_LIMIT (Java field), 358
 VAR_Y_UPPER_LIMIT (Java field), 358
 VAR_Z_LOWER_LIMIT (Java field), 358
 VAR_Z_UPPER_LIMIT (Java field), 358
 Variable (Java interface), 520
 Variable_Position() (Java method), 370
 VariableComparator (Java class), 681–684
 VariableFactory (Java class), 534
 VariableFactoryTest (Java class), 536
 VARIABLES (Java field), 358
 variance(List) (Java method), 130
 Variant (Java enum), 109
 variant (Java field), 174
 varyingProbability (Java field), 569
 vectorCorrelation(List, List) (Java method), 340
 Viennet2 (Java class), 325
 Viennet2() (Java constructor), 325
 Viennet3 (Java class), 326
 Viennet3() (Java constructor), 326
 Viennet4 (Java class), 326
 Viennet4() (Java constructor), 326
 Vij_ (Java field), 358
 ViolationThresholdComparator (Java class), 589
 ViolationThresholdComparator() (Java constructor), 589

W

w (Java field), 456
 WASFGA (Java class), 195
 WASFGA(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, double, List) (Java constructor), 196

- WASFGA(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, double, List, String) (Java constructor), 195
- WASFGABinaryRunner (Java class), 509
- WASFGAIT (Java class), 197
- WASFGAMEasures (Java class), 198
- WASFGAMEasures(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, double, List) (Java constructor), 198
- WASFGAMEasures(Problem, int, int, CrossoverOperator, MutationOperator, SelectionOperator, SolutionListEvaluator, double, List, String) (Java constructor), 198
- WASFGAMEasuresRunner (Java class), 510
- WASFGAMEasuresRunner3D (Java class), 510
- WASFGARanking (Java class), 199
- WASFGARanking(AbstractUtilityFunctionsSet) (Java constructor), 200
- WASFGARunner (Java class), 510
- Water (Java class), 327
- Water() (Java constructor), 327
- weierstrass(double[]) (Java method), 433
- weierstrass(double[], double, double, int) (Java method), 433
- WEIGHTED_CHEBYSHEV (Java field), 75
- WEIGHTED_PRODUCT (Java field), 75
- WEIGHTED_SUM (Java field), 75
- weightedChebyshev(List, double[]) (Java method), 72
- weightedChebyshev(List, double[], double[]) (Java method), 72
- weightedProduct(List, double[]) (Java method), 72
- weightedSum(List, double[]) (Java method), 72
- WeightElement_ (Java field), 359
- WeightNode_ (Java field), 359
- weights (Java field), 195
- WeightVectorNeighborhood (Java class), 638
- WeightVectorNeighborhood(int, int) (Java constructor), 638
- WeightVectorNeighborhood(int, int, int, String) (Java constructor), 638
- WeightVectorNeighborhoodTest (Java class), 639
- WeightVectors (Java class), 200
- Well44497bGenerator (Java class), 672
- Well44497bGenerator() (Java constructor), 673
- Well44497bGenerator(long) (Java constructor), 673
- WFG (Java class), 406
- WFG(Integer, Integer, Integer) (Java constructor), 407
- WFG1 (Java class), 408
- WFG1() (Java constructor), 408
- WFG1(Integer, Integer, Integer) (Java constructor), 408
- WFG2 (Java class), 409
- WFG2() (Java constructor), 409
- WFG2(Integer, Integer, Integer) (Java constructor), 410
- WFG3 (Java class), 410
- WFG3() (Java constructor), 411
- WFG3(Integer, Integer, Integer) (Java constructor), 411
- WFG4 (Java class), 412
- WFG4() (Java constructor), 412
- WFG4(Integer, Integer, Integer) (Java constructor), 412
- WFG5 (Java class), 413
- WFG5() (Java constructor), 413
- WFG5(Integer, Integer, Integer) (Java constructor), 413
- WFG6 (Java class), 414
- WFG6() (Java constructor), 414
- WFG6(Integer, Integer, Integer) (Java constructor), 414
- WFG7 (Java class), 415
- WFG7() (Java constructor), 415
- WFG7(Integer, Integer, Integer) (Java constructor), 415
- WFG8 (Java class), 416
- WFG8() (Java constructor), 416
- WFG8(Integer, Integer, Integer) (Java constructor), 416
- WFG9 (Java class), 417
- WFG9() (Java constructor), 418
- WFG9(Integer, Integer, Integer) (Java constructor), 418
- WFGHypervolume (Java class), 480
- WFGHypervolume() (Java constructor), 480
- WFGHypervolume(Front) (Java constructor), 481
- WFGHypervolume(String) (Java constructor), 480
- WfgHypervolumeFront (Java class), 482
- WfgHypervolumeFront() (Java constructor), 483
- WfgHypervolumeFront(int, int) (Java constructor), 483
- WfgHypervolumeFront(List) (Java constructor), 483
- WFGHypervolumeTest (Java class), 482
- WfgHypervolumeVersion (Java class), 483
- WfgHypervolumeVersion(int, int) (Java constructor), 484
- WfgHypervolumeVersion(int, int, Point) (Java constructor), 484
- WORST_IN_ARCHIVE (Java field), 69
- wrapListener(MeasureListener) (Java method), 246
- wrapManager(MeasureManager, Object) (Java method), 246
- wrapMeasure(PushMeasure) (Java method), 247
- writeObjectivesToMatrix(List) (Java method), 550
- ## X
- xA(double[], double[], double[][]) (Java method), 433
- xy(double[], double[]) (Java method), 433
- ## Y
- Y_ (Java field), 359
- ## Z
- z (Java field), 52, 456
- Z_ (Java field), 359
- ZDT1 (Java class), 419
- ZDT1() (Java constructor), 419

ZDT1(Integer) (Java constructor), [419](#)
ZDT2 (Java class), [420](#)
ZDT2() (Java constructor), [420](#)
ZDT2(Integer) (Java constructor), [420](#)
ZDT3 (Java class), [420](#)
ZDT3() (Java constructor), [420](#)
ZDT3(Integer) (Java constructor), [421](#)
ZDT4 (Java class), [421](#)
ZDT4() (Java constructor), [421](#)
ZDT4(Integer) (Java constructor), [421](#)
ZDT5 (Java class), [422](#)
ZDT5() (Java constructor), [422](#)
ZDT5(Integer) (Java constructor), [422](#)
ZDT6 (Java class), [423](#)
ZDT6() (Java constructor), [423](#)
ZDT6(Integer) (Java constructor), [424](#)
ZDTScalabilityIStudy (Java class), [233](#)
ZDTScalabilityIStudy2 (Java class), [233](#)
ZDTStudy (Java class), [234](#)
ZDTStudy2 (Java class), [235](#)
zM (Java field), [456](#)