

## Optimization Algorithm Analysis

Generated by Doxygen 1.9.3



<b>1 Optimization</b>	<b>1</b>
1.1 Running the project	1
1.2 Experiment File Formats	1
1.2.1 Scheduling Algorithm File Format	1
1.2.2 Minimization Algorithm File Format	2
1.3 Output File Format	2
1.3.1 Scheduling Algorithms	2
1.3.2 Minimization Algorithms	3
1.4 Documentation	3
<b>2 Namespace Index</b>	<b>5</b>
2.1 Packages	5
<b>3 Hierarchical Index</b>	<b>7</b>
3.1 Class Hierarchy	7
<b>4 Class Index</b>	<b>9</b>
4.1 Class List	9
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Namespace Documentation</b>	<b>13</b>
6.1 Package mt	13
6.2 Package project	13
<b>7 Class Documentation</b>	<b>15</b>
7.1 project.Algorithms Class Reference	15
7.1.1 Detailed Description	15
7.1.2 Constructor & Destructor Documentation	15
7.1.2.1 Algorithms()	15
7.1.3 Member Function Documentation	16
7.1.3.1 blindSearch()	16
7.1.3.2 DE()	16
7.1.3.3 getFitness()	17
7.1.3.4 getSolution()	17
7.1.3.5 localSearch()	17
7.1.3.6 PSO()	18
7.1.3.7 repeatedLocalSearch()	18
7.2 Main Class Reference	19
7.2.1 Detailed Description	19
7.2.2 Member Function Documentation	19
7.2.2.1 experiment()	19
7.2.2.2 fileLoop()	20

7.2.2.3	main()	20
7.2.2.4	makeOutFile()	21
7.2.2.5	minimizeDriver()	21
7.2.2.6	readFromFile()	21
7.2.2.7	readLines()	22
7.2.2.8	scheduleDriver()	22
7.2.2.9	scheduleLoop()	22
7.2.2.10	writeFile()	23
7.2.2.11	writeResult()	23
7.3	mt.MTRandom Class Reference	24
7.3.1	Detailed Description	25
7.3.2	Constructor & Destructor Documentation	25
7.3.2.1	MTRandom() [1/5]	25
7.3.2.2	MTRandom() [2/5]	25
7.3.2.3	MTRandom() [3/5]	26
7.3.2.4	MTRandom() [4/5]	26
7.3.2.5	MTRandom() [5/5]	26
7.3.3	Member Function Documentation	27
7.3.3.1	next()	27
7.3.3.2	pack()	28
7.3.3.3	setSeed() [1/3]	28
7.3.3.4	setSeed() [2/3]	29
7.3.3.5	setSeed() [3/3]	29
7.4	project.NEH Class Reference	30
7.4.1	Detailed Description	30
7.4.2	Constructor & Destructor Documentation	30
7.4.2.1	NEH()	30
7.4.3	Member Function Documentation	31
7.4.3.1	FSS()	31
7.4.3.2	FSSB()	31
7.4.3.3	runNEH()	32
7.4.3.4	transposeMatrix()	32
7.4.4	Member Data Documentation	32
7.4.4.1	m	33
7.4.4.2	makespan	33
7.4.4.3	matrix	33
7.4.4.4	n	33
7.4.4.5	schedule	33
7.5	project.Particle Class Reference	33
7.5.1	Detailed Description	34
7.5.2	Constructor & Destructor Documentation	34
7.5.2.1	Particle()	34

7.5.3 Member Function Documentation	34
7.5.3.1 setPBest()	34
7.5.4 Member Data Documentation	35
7.5.4.1 fitness	35
7.5.4.2 pBest	35
7.5.4.3 solution	35
7.5.4.4 velocity	35
7.6 project.Population Class Reference	36
7.6.1 Detailed Description	36
7.6.2 Constructor & Destructor Documentation	36
7.6.2.1 Population()	36
7.6.3 Member Function Documentation	36
7.6.3.1 genNeighborhood()	37
7.6.3.2 genRandomArray()	37
7.6.3.3 genRandomMatrix()	37
7.6.3.4 getFitness()	38
7.6.3.5 getPopulation()	38
7.6.3.6 getRange()	38
7.6.3.7 getSolution()	39
7.6.3.8 setFitness()	39
7.6.3.9 setPopulation()	39
7.6.3.10 setSolution()	39
7.7 project.Problem Class Reference	40
7.7.1 Detailed Description	40
7.7.2 Constructor & Destructor Documentation	40
7.7.2.1 Problem()	40
7.7.3 Member Function Documentation	41
7.7.3.1 ackley_one()	41
7.7.3.2 ackley_two()	41
7.7.3.3 de_jong_1()	41
7.7.3.4 egg_holder()	42
7.7.3.5 getFitness()	42
7.7.3.6 griewank()	42
7.7.3.7 nthRoot()	42
7.7.3.8 rastrigin()	43
7.7.3.9 rosenbrock()	43
7.7.3.10 schwefel()	43
7.7.3.11 sine_envelope()	44
7.7.3.12 sine_V()	44
7.7.3.13 square()	44
<b>8 File Documentation</b>	<b>45</b>

---

8.1 Main.java File Reference . . . . .	45
8.2 Main.java . . . . .	45
8.3 mt/MTRandom.java File Reference . . . . .	48
8.4 MTRandom.java . . . . .	48
8.5 project/Algorithms.java File Reference . . . . .	51
8.6 Algorithms.java . . . . .	51
8.7 project/NEH.java File Reference . . . . .	55
8.8 NEH.java . . . . .	55
8.9 project/Particle.java File Reference . . . . .	57
8.10 Particle.java . . . . .	58
8.11 project/Population.java File Reference . . . . .	58
8.12 Population.java . . . . .	58
8.13 project/Problem.java File Reference . . . . .	60
8.14 Problem.java . . . . .	60
8.15 README.md File Reference . . . . .	62
<b>Index</b>	<b>63</b>

# Chapter 1

## Optimization

This is project that is used for testing optimization algorithms.

The currently implemented algorithms are:

- `Blind Search`
- `Repeated Local Search`
- `Differential Evolution`
- `Particle Swarm Optimization`
- `Nawaz-Enscore-Ham (NEH)`

### 1.1 Running the project

Using the Java Virtual Machine (JVM), run the project from the [Main](#) file in this project to generate the desired output files.

In the terminal a prompt will appear. Either type "minimization" or "scheduling" for the desired algorithm types. The minimization algorithms are: Blind Search, Repeated Local Search, Differential Evolution, Particle Swarm Optimization. The scheduling algorithm is: NEH.

### 1.2 Experiment File Formats

#### 1.2.1 Scheduling Algorithm File Format

Line 1: `[# of machines] [# of jobs]`

Following lines: processing times for each job for each machine. Each line represents a machine and the times for each job are separated by spaces.

Example:

```
5 4
5 9 9 4
```

```

9 3 4 8
8 10 5 8
10 1 8 7
1 8 6 2

```

The first line in the example says there are five machines (five lines after the first one) and four jobs (4 numbers per following line). The values in the following lines are the processing times for each job for each machine (e.g., on machine 1, job 1 has a processing time of 5).

Sample files can be found in the project folder in the `Taillard_TestData` folder.

## 1.2.2 Minimization Algorithm File Format

```
[algorithm] [DE method] [crossover type] [dimension] [population size] [problem
type] [range] [num experiments]
```

- The values for `[algorithm]` are 1 for DE, 2 for PSO, 3 for Blind Search, and 4 for Repeated Local Search.
- The values for `[DE method]` are 1 for DE/best/1, 2 for DE/rand/1, 3 for DE/rand-to-best/1, 4 for DE/best/2, and 5 for DE/rand/2.
- The values for `[crossover type]` are 1 for exponential crossover and 2 for binomial crossover.
- The value for `[dimension]` is the number of elements in each solution vector.
- The value for `[population size]` is the number of solution vectors in the population.
- The value for `[problem type]` is the objective function label: 1 - Schwefel, 2 - De Jong 1, 3 - Rosenbrock, 4 - Rastrigin, 5 - Griewank, 6 - Sine Envelope Sine Wave, 7 - Stretch V Sine Wave, 8 - Ackley One, 9 - Ackley Two, 10 - Egg Holder.
- The value for `[range]` is the range of initial values for each element in the solution vector.
- The value for `[num experiments]` is the number of experiments to run.

The `experiments.txt` file contains a list of all the experiments to run. The experiments are run in the order they appear in the file. The `experiments.txt` file should be in the same directory as the [Main](#) file to properly run the project.

## 1.3 Output File Format

The output files are named according to the current system time. The output files are placed in the directory of the [Main](#) file.

### 1.3.1 Scheduling Algorithms

Generates a CSV where each row is a different experiment. The first column is the number of machines, the second column is the number of jobs, the third column is the resulting makespan value, the fourth column is the run time for the experiment, the following columns in a row hold the resulting schedule for the experiment.



### 1.3.2 Minimization Algorithms

Generates a CSV where each row is the resulting fitness values from each experiment. The first entry in each row has information on the algorithm run, how many experiments were run, and the time it took to run the experiments.

## 1.4 Documentation

The documentation.pdf file contains the documentation for this project. The documentation was generated using Doxygen. A Doxygen config file is included if you want to configure the documentation to your liking.



## Chapter 2

# Namespace Index

### 2.1 Packages

Here are the packages with brief descriptions (if available):

<a href="#">mt</a> . . . . .	<a href="#">13</a>
<a href="#">project</a> . . . . .	<a href="#">13</a>



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

project.Algorithms . . . . .	15
Main . . . . .	19
project.NEH . . . . .	30
project.Particle . . . . .	33
project.Population . . . . .	36
project.Problem . . . . .	40
Random	
mt.MTRandom . . . . .	24



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">project.Algorithms</a>	15
<a href="#">Main</a>	19
<a href="#">mt.MTRandom</a>	24
<a href="#">project.NEH</a>	30
<a href="#">project.Particle</a>	33
<a href="#">project.Population</a>	36
<a href="#">project.Problem</a>	40





## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Main.java</a>	45
<a href="#">mt/MTRandom.java</a>	48
<a href="#">project/Algorithms.java</a>	51
<a href="#">project/NEH.java</a>	55
<a href="#">project/Particle.java</a>	57
<a href="#">project/Population.java</a>	58
<a href="#">project/Problem.java</a>	60



## Chapter 6

# Namespace Documentation

### 6.1 Package mt

#### Classes

- class [MTRandom](#)

### 6.2 Package project

#### Classes

- class [Algorithms](#)
- class [NEH](#)
- class [Particle](#)
- class [Population](#)
- class [Problem](#)



## Chapter 7

# Class Documentation

### 7.1 project.Algorithms Class Reference

#### Public Member Functions

- [Algorithms](#) (int algorithm, [Population](#) pop, int problem, int crosstype, int method, int index)
- double[] [getSolution](#) ()
- double [getFitness](#) ()
- double[] [blindSearch](#) (int iterations, double[] bestSol, double fitness)
- double[] [localSearch](#) (double[] initialSol, double[] bestSol, boolean tau)
- double[] [repeatedLocalSearch](#) (double[] initialSol, double[] bestGlobalSol, double[] bestIterSol, boolean tau, int iterations)
- double[] [DE](#) (int method, int D, int NP, double CR, double F, double lambda, int generations)
- [Particle](#) [PSO](#) (int iterations, int numParticles, int dimensions, double c1, double c2)

#### 7.1.1 Detailed Description

Definition at line 15 of file [Algorithms.java](#).

#### 7.1.2 Constructor & Destructor Documentation

##### 7.1.2.1 Algorithms()

```
project.Algorithms.Algorithms (
    int algorithm,
    Population pop,
    int problem,
    int crosstype,
    int method,
    int index )
```

Constructor for IAlgs

**Parameters**

<i>algorithm</i>	- algorithm to run
<i>pop</i>	- population
<i>problem</i>	- problem type
<i>crosstype</i>	- crossover type

Definition at line 51 of file [Algorithms.java](#).

### 7.1.3 Member Function Documentation

#### 7.1.3.1 blindSearch()

```
double[] project.Algorithms.blindSearch (
    int iterations,
    double[] bestSol,
    double fitness )
```

Returns the best solution found by the blind search algorithm

**Parameters**

<i>iterations</i>	- number of iterations
<i>bestSol</i>	- best solution found
<i>fitness</i>	- fitness of the best solution

**Returns**

- best solution found

Definition at line 99 of file [Algorithms.java](#).

#### 7.1.3.2 DE()

```
double[] project.Algorithms.DE (
    int method,
    int D,
    int NP,
    double CR,
    double F,
    double lambda,
    int generations )
```

Runs an experiment for the Differential Evolution algorithm

**Parameters**

<i>method</i>	- mutation method
<i>D</i>	- Dimensions
<i>NP</i>	- <a href="#">Population</a> size
<i>CR</i>	- Crossover rate
<i>F</i>	- Scaling factor
<i>lambda</i>	- Scaling factor
<i>generations</i>	- Number of generations

**Returns**

- The best fitness vector

Definition at line [220](#) of file [Algorithms.java](#).

**7.1.3.3 getFitness()**

```
double project.Algorithms.getFitness ( )
```

**Returns**

- the fitness of the solution vector

Definition at line [86](#) of file [Algorithms.java](#).

**7.1.3.4 getSolution()**

```
double[] project.Algorithms.getSolution ( )
```

**Returns**

- solution vector of the algorithm

Definition at line [79](#) of file [Algorithms.java](#).

**7.1.3.5 localSearch()**

```
double[] project.Algorithms.localSearch (
    double[] initialSol,
    double[] bestSol,
    boolean tau )
```

Returns the best solution found by the local search algorithm

**Parameters**

<i>initialSol</i>	- initial solution
<i>bestSol</i>	- best solution found
<i>tau</i>	- boolean variable to determine if the algorithm should terminate

**Returns**

- best solution found

Definition at line 130 of file [Algorithms.java](#).

**7.1.3.6 PSO()**

```
Particle project.Algorithms.PSO (
    int iterations,
    int numParticles,
    int dimensions,
    double c1,
    double c2 )
```

Runs an experiment for the [Particle](#) Swarm Optimization algorithm

**Parameters**

<i>iterations</i>	- Number of iterations
<i>numParticles</i>	- Number of particles
<i>dimensions</i>	- Number of dimensions
<i>c1</i>	- cognitive factor
<i>c2</i>	- social factor

**Returns**

- The best fitness particle

Definition at line 345 of file [Algorithms.java](#).

**7.1.3.7 repeatedLocalSearch()**

```
double[] project.Algorithms.repeatedLocalSearch (
    double[] initialSol,
    double[] bestGlobalSol,
    double[] bestIterSol,
    boolean tau,
    int iterations )
```

Returns the best solution found by repeated local search algorithms



## Parameters

<i>initialSol</i>	- initial solution
<i>bestGlobalSol</i>	- best overall solution found
<i>bestIterSol</i>	- best solution found in each iteration
<i>tau</i>	- boolean variable to determine if the algorithm should terminate
<i>iterations</i>	- maximum number of iterations

## Returns

- best solution found

Definition at line 177 of file [Algorithms.java](#).

The documentation for this class was generated from the following file:

- project/[Algorithms.java](#)

## 7.2 Main Class Reference

### Static Public Member Functions

- static void [main](#) (String[] args)
- static void [minimizeDriver](#) ()
- static void [scheduleDriver](#) ()
- static void [fileLoop](#) (String line, BufferedReader br, BufferedWriter bw)
- static void [experiment](#) (int n, [Population](#) pop, long[] times, int algorithm, int problem, int method, int crosstype)
- static void [scheduleLoop](#) (int[][] times, int m, int n, int alg, BufferedWriter bw)
- static ArrayList< String > [readLines](#) (int i)
- static void [writeFile](#) (BufferedWriter bw, int problem, int n, int m, int alg, double range, long sum, [Population](#) pop, int method, int crosstype)
- static void [writeResult](#) (NEH algos, BufferedWriter bw, int m, int n, long time)
- static BufferedReader [readFromFile](#) (String filename)
- static BufferedWriter [makeOutFile](#) ()

### 7.2.1 Detailed Description

Definition at line 9 of file [Main.java](#).

### 7.2.2 Member Function Documentation

#### 7.2.2.1 [experiment\(\)](#)

```
static void Main.experiment (
    int n,
    Population pop,
    long[] times,
    int algorithm,
    int problem,
    int method,
    int crosstype ) [static]
```

Runs [n] minimization experiments of problem type [problem], and stores fitness values in the population

**Parameters**

<i>n</i>	- number of experiments
<i>pop</i>	- population to store fitness values
<i>times</i>	- array to store time values
<i>algorithm</i>	- algorithm to use
<i>problem</i>	- problem type
<i>method</i>	- method to use for mutation in DE
<i>crosstype</i>	- crossover type for DE

Definition at line 189 of file [Main.java](#).

**7.2.2.2 fileLoop()**

```
static void Main.fileLoop (
    String line,
    BufferedReader br,
    BufferedWriter bw ) [static]
```

Runs the minimization experiments for the given line

**Parameters**

<i>line</i>	- line from input file
<i>br</i>	- BufferedReader for input file
<i>bw</i>	- BufferedWriter for output file

**Exceptions**

<i>IOException</i>	- if there is an error with the input or output files
--------------------	-------------------------------------------------------

algorithm - algorithm to use (1 - DE, 2 - PSO) method - method to use (1 - DE/Best/1, 2 - DE/Rand/1, 3 - DE/Rand-To-Best/1, 4 - DE/Best/2, 5 - DE/Rand/2) crosstype - crossover type (1 - exponential, 2 - binomial) m - dimensions n - population size problem - problem type range - range of values numExperiments - number of experiments to run

Definition at line 126 of file [Main.java](#).

**7.2.2.3 main()**

```
static void Main.main (
    String[] args ) [static]
```

Valid inputs: "minimization", "scheduling".

Choose the algorithm type to use. 0: minimization algorithms 1: scheduling algorithms

Definition at line 11 of file [Main.java](#).

#### 7.2.2.4 makeOutFile()

```
static BufferedWriter Main.makeOutFile ( ) [static]
```

Creates a CSV file to write the experiments to

##### Returns

BufferedWriter of that CSV file

##### Exceptions

<i>IOException</i>	- if there is an error with the output file
--------------------	---------------------------------------------

Definition at line 352 of file [Main.java](#).

#### 7.2.2.5 minimizeDriver()

```
static void Main.minimizeDriver ( ) [static]
```

Driver for the minimization functions. Reading from the input file and creating the output file

Input file format: [algorithm] [DE method] [crossover type] [dimension] [population size] [problem type] [range] [num experiments]

Definition at line 44 of file [Main.java](#).

#### 7.2.2.6 readFromFile()

```
static BufferedReader Main.readFromFile (
    String filename ) [static]
```

Reads input file with given filename and returns a buffered reader for the input file

##### Parameters

<i>filename</i>	- name of the input file
-----------------	--------------------------

##### Returns

buffered reader for the input file

##### Exceptions

<i>FileNotFoundException</i>	- if there is an error with the input file
------------------------------	--------------------------------------------

Definition at line 335 of file [Main.java](#).

### 7.2.2.7 readLines()

```
static ArrayList< String > Main.readLines (
    int i ) [static]
```

This method reads the lines from the input file and returns them as an ArrayList (for scheduling)

#### Parameters

<i>i</i>	- the number of the input file
----------	--------------------------------

#### Returns

- the ArrayList of lines from the input file

Definition at line 228 of file [Main.java](#).

### 7.2.2.8 scheduleDriver()

```
static void Main.scheduleDriver ( ) [static]
```

Driver for the scheduling functions.

Definition at line 76 of file [Main.java](#).

### 7.2.2.9 scheduleLoop()

```
static void Main.scheduleLoop (
    int times[ ][ ],
    int m,
    int n,
    int alg,
    BufferedWriter bw ) [static]
```

Runs the scheduling experiment for the given input file

#### Parameters

<i>times</i>	- the processing times matrix.
<i>m</i>	- the number of machines.
<i>n</i>	- the number of jobs.
<i>alg</i>	- the scheduling algorithm to use.
<i>bw</i>	- the BufferedWriter for the output file.

Definition at line 210 of file [Main.java](#).

#### 7.2.2.10 writeFile()

```
static void Main.writeFile (
    BufferedWriter bw,
    int problem,
    int n,
    int m,
    int alg,
    double range,
    long sum,
    Population pop,
    int method,
    int crosstype ) [static]
```

Save minimization experiments to file

##### Parameters

<i>bw</i>	- BufferedWriter to write to
<i>problem</i>	- Problem type
<i>n</i>	- Number of experiments
<i>m</i>	- Dimensions
<i>range</i>	- Range of values
<i>sum</i>	- Total time for the experiment
<i>pop</i>	- Population
<i>method</i>	- Method to use for mutation in DE
<i>crosstype</i>	- Crossover type for DE

##### Exceptions

<i>IOException</i>	- if there is an error with the output file
--------------------	---------------------------------------------

Definition at line 258 of file [Main.java](#).

#### 7.2.2.11 writeResult()

```
static void Main.writeResult (
    NEH algos,
    BufferedWriter bw,
    int m,
    int n,
    long time ) [static]
```

Writes the results of the scheduling experiment to the CSV file

**Parameters**

<i>algos</i>	- the results of the experiment
<i>bw</i>	- the CSV file to write to
<i>m</i>	- the number of machines
<i>n</i>	- the number of jobs
<i>time</i>	- the time it took to run the experiment

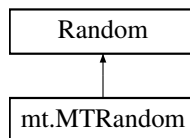
Definition at line 313 of file [Main.java](#).

The documentation for this class was generated from the following file:

- [Main.java](#)

## 7.3 mt.MTRandom Class Reference

Inheritance diagram for mt.MTRandom:



### Public Member Functions

- [MTRandom](#) ()
- [MTRandom](#) (boolean compatible)
- [MTRandom](#) (long seed)
- [MTRandom](#) (byte[] buf)
- [MTRandom](#) (int[] buf)
- final synchronized void [setSeed](#) (long seed)
- final void [setSeed](#) (byte[] buf)
- final synchronized void [setSeed](#) (int[] buf)

### Static Public Member Functions

- static int[] [pack](#) (byte[] buf)

### Protected Member Functions

- final synchronized int [next](#) (int bits)

### 7.3.1 Detailed Description

#### Version

1.0

#### Author

David Beaumont, Copyright 2005

A Java implementation of the MT19937 (Mersenne Twister) pseudo random number generator algorithm based upon the original C code by Makoto Matsumoto and Takuji Nishimura (see <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html> for more information).

As a subclass of `java.util.Random` this class provides a single canonical method `next()` for generating bits in the pseudo random number sequence. Anyone using this class should invoke the public inherited methods (`nextInt()`, `nextFloat()` etc.) to obtain values as normal. This class should provide a drop-in replacement for the standard implementation of `java.util.Random` with the additional advantage of having a far longer period and the ability to use a far larger seed value.

This is **not** a cryptographically strong source of randomness and should **not** be used for cryptographic systems or in any other situation where true random numbers are required.

This software is licensed under the [CC-GNU LGPL](#).

Definition at line 88 of file [MTRandom.java](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 MTRandom() [1/5]

```
mt.MTRandom.MTRandom ( )
```

The default constructor for an instance of [MTRandom](#). This invokes the no-argument constructor for `java.util.Random` which will result in the class being initialised with a seed value obtained by calling `System.currentTimeMillis()`.

Definition at line 127 of file [MTRandom.java](#).

#### 7.3.2.2 MTRandom() [2/5]

```
mt.MTRandom.MTRandom (
    boolean compatible )
```

This version of the constructor can be used to implement identical behaviour to the original C code version of this algorithm including exactly replicating the case where the seed value had not been set prior to calling `genrand_int32`.

If the compatibility flag is set to true, then the algorithm will be seeded with the same default value as was used in the original C code. Furthermore the `setSeed()` method, which must take a 64 bit long value, will be limited to using only the lower 32 bits of the seed to facilitate seamless migration of existing C code into Java where identical behaviour is required.

Whilst useful for ensuring backwards compatibility, it is advised that this feature not be used unless specifically required, due to the reduction in strength of the seed value.

## Parameters

<i>compatible</i>	Compatibility flag for replicating original behaviour.
-------------------	--------------------------------------------------------

Definition at line 150 of file [MTRandom.java](#).

**7.3.2.3 MTRandom()** [3/5]

```
mt.MTRandom.MTRandom (
    long seed )
```

This version of the constructor simply initialises the class with the given 64 bit seed value. For a better random number sequence this seed value should contain as much entropy as possible.

## Parameters

<i>seed</i>	The seed value with which to initialise this class.
-------------	-----------------------------------------------------

Definition at line 163 of file [MTRandom.java](#).

**7.3.2.4 MTRandom()** [4/5]

```
mt.MTRandom.MTRandom (
    byte[] buf )
```

This version of the constructor initialises the class with the given byte array. All the data will be used to initialise this instance.

## Parameters

<i>buf</i>	The non-empty byte array of seed information.
------------	-----------------------------------------------

## Exceptions

<i>NullPointerException</i>	if the buffer is null.
<i>IllegalArgumentException</i>	if the buffer has zero length.

Definition at line 176 of file [MTRandom.java](#).

**7.3.2.5 MTRandom()** [5/5]

```
mt.MTRandom.MTRandom (
    int[] buf )
```



This version of the constructor initialises the class with the given integer array. All the data will be used to initialise this instance.

#### Parameters

<i>buf</i>	The non-empty integer array of seed information.
------------	--------------------------------------------------

#### Exceptions

<i>NullPointerException</i>	if the buffer is null.
<i>IllegalArgumentException</i>	if the buffer has zero length.

Definition at line 190 of file [MTRandom.java](#).

### 7.3.3 Member Function Documentation

#### 7.3.3.1 next()

```
final synchronized int mt.MTRandom.next (
    int bits ) [protected]
```

This method forms the basis for generating a pseudo random number sequence from this class. If given a value of 32, this method behaves identically to the `genrand_int32` function in the original C code and ensures that using the standard `nextInt()` function (inherited from `Random`) we are able to replicate behaviour exactly.

Note that where the number of bits requested is not equal to 32 then bits will simply be masked out from the top of the returned integer value. That is to say that:

```
mt.setSeed(12345);
int foo = mt.nextInt(16) + (mt.nextInt(16) << 16);
```

will not give the same result as

```
mt.setSeed(12345);
int foo = mt.nextInt(32);
```

#### Parameters

<i>bits</i>	The number of significant bits desired in the output.
-------------	-------------------------------------------------------

#### Returns

The next value in the pseudo random sequence with the specified number of bits in the lower part of the integer.

Definition at line 336 of file [MTRandom.java](#).

### 7.3.3.2 pack()

```
static int[] mt.MTRandom.pack (
    byte[] buf ) [static]
```

This simply utility method can be used in cases where a byte array of seed data is to be used to repeatedly re-seed the random number sequence. By packing the byte array into an integer array first, using this method, and then invoking `setSeed()` with that; it removes the need to re-pack the byte array each time `setSeed()` is called.

If the length of the byte array is not a multiple of 4 then it is implicitly padded with zeros as necessary. For example:

```
byte[] { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 }
```

becomes

```
int[] { 0x04030201, 0x00000605 }
```

<p<blockquote>

Note that this method will not complain if the given byte array is empty and will produce an empty integer array, but the `setSeed()` method will throw an exception if the empty integer array is passed to it.

#### Parameters

<i>buf</i>	The non-null byte array to be packed.
------------	---------------------------------------

#### Returns

A non-null integer array of the packed bytes.

#### Exceptions

<i>NullPointerException</i>	if the given byte array is null.
-----------------------------	----------------------------------

Definition at line 407 of file [MTRandom.java](#).

### 7.3.3.3 setSeed() [1/3]

```
final void mt.MTRandom.setSeed (
    byte[] buf )
```

This method resets the state of this instance using the byte array of seed data provided. Note that calling this method is equivalent to calling "setSeed(pack(buf))" and in particular will result in a new integer array being generated during the call. If you wish to retain this seed data to allow the pseudo random sequence to be restarted then it would be more efficient to use the "pack()" method to convert it into an integer array first and then use that to re-seed the instance. The behaviour of the class will be the same in both cases but it will be more efficient.

#### Parameters

<i>buf</i>	The non-empty byte array of seed information.
------------	-----------------------------------------------

#### Exceptions

<i>NullPointerException</i>	if the buffer is null.
<i>IllegalArgumentException</i>	if the buffer has zero length.

Definition at line 266 of file [MTRandom.java](#).

#### 7.3.3.4 setSeed() [2/3]

```
final synchronized void mt.MTRandom.setSeed (
    int[] buf )
```

This method resets the state of this instance using the integer array of seed data provided. This is the canonical way of resetting the pseudo random number sequence.

#### Parameters

<i>buf</i>	The non-empty integer array of seed information.
------------	--------------------------------------------------

#### Exceptions

<i>NullPointerException</i>	if the buffer is null.
<i>IllegalArgumentException</i>	if the buffer has zero length.

Definition at line 279 of file [MTRandom.java](#).

#### 7.3.3.5 setSeed() [3/3]

```
final synchronized void mt.MTRandom.setSeed (
    long seed )
```

This method resets the state of this instance using the 64 bits of seed data provided. Note that if the same seed data is passed to two different instances of [MTRandom](#) (both of which share the same compatibility state) then the sequence of numbers generated by both instances will be identical.

If this instance was initialised in 'compatibility' mode then this method will only use the lower 32 bits of any seed value passed in and will match the behaviour of the original C code exactly with respect to state initialisation.

## Parameters

<i>seed</i>	The 64 bit value used to initialise the random number generator state.
-------------	------------------------------------------------------------------------

Definition at line 231 of file [MTRandom.java](#).

The documentation for this class was generated from the following file:

- [mt/MTRandom.java](#)

## 7.4 project.NEH Class Reference

### Public Member Functions

- [NEH](#) (int[ ][ ] times, int [m](#), int [n](#), int alg)
- int [runNEH](#) (int alg)
- int [FSS](#) (int[ ][ ] times, int [m](#), int [n](#))
- int [FSSB](#) (int[ ][ ] times, int [m](#), int [n](#))

### Static Public Member Functions

- static int[ ][ ] [transposeMatrix](#) (int[ ][ ] [matrix](#))

### Public Attributes

- int[ ][ ] [matrix](#)
- int[ ] [schedule](#)
- int [makespan](#)
- int [m](#)
- int [n](#)

#### 7.4.1 Detailed Description

Definition at line 15 of file [NEH.java](#).

#### 7.4.2 Constructor & Destructor Documentation

##### 7.4.2.1 NEH()

```
project.NEH.NEH (
    int times[ ][ ],
    int m,
    int n,
    int alg )
```

Constructor

**Parameters**

<i>times</i>	- the processing times of each job on each machine
<i>m</i>	- the number of machines
<i>n</i>	- the number of jobs
<i>alg</i>	- the algorithm to use

Definition at line 39 of file [NEH.java](#).

**7.4.3 Member Function Documentation****7.4.3.1 FSS()**

```
int project.NEH.FSS (
    int times[ ][ ],
    int m,
    int n )
```

The FSS algorithm for the makespan problem.

**Parameters**

<i>times</i>	- the matrix of processing times
<i>m</i>	- the number of machines
<i>n</i>	- the number of jobs

**Returns**

the makespan value of the schedule

Definition at line 170 of file [NEH.java](#).

**7.4.3.2 FSSB()**

```
int project.NEH.FSSB (
    int times[ ][ ],
    int m,
    int n )
```

The FSSB algorithm for the makespan problem.

**Parameters**

<i>times</i>	- the matrix of processing times
<i>m</i>	- the number of machines
<i>n</i>	- the number of jobs

**Returns**

the makespan value of the schedule

Definition at line 202 of file [NEH.java](#).

**7.4.3.3 runNEH()**

```
int project.NEH.runNEH (
    int alg )
```

The [NEH](#) algorithm to determine the schedule

**Parameters**

<i>alg</i>	- the algorithm to use
------------	------------------------

**Returns**

the makespan of the determined schedule

Definition at line 53 of file [NEH.java](#).

**7.4.3.4 transposeMatrix()**

```
static int[][] project.NEH.transposeMatrix (
    int matrix[][] ) [static]
```

This method transposes the given matrix Function from: <https://stackoverflow.com/questions/26197466/transpose-matrix>

**Parameters**

<i>matrix</i>	- the matrix to be transposed
---------------	-------------------------------

**Returns**

- the transposed matrix

Definition at line 147 of file [NEH.java](#).

**7.4.4 Member Data Documentation**

#### 7.4.4.1 m

```
int project.NEH.m
```

Definition at line 26 of file [NEH.java](#).

#### 7.4.4.2 makespan

```
int project.NEH.makespan
```

Definition at line 23 of file [NEH.java](#).

#### 7.4.4.3 matrix

```
int [][] project.NEH.matrix
```

Definition at line 17 of file [NEH.java](#).

#### 7.4.4.4 n

```
int project.NEH.n
```

Definition at line 29 of file [NEH.java](#).

#### 7.4.4.5 schedule

```
int [] project.NEH.schedule
```

Definition at line 20 of file [NEH.java](#).

The documentation for this class was generated from the following file:

- [project/NEH.java](#)

## 7.5 project.Particle Class Reference

### Public Member Functions

- [Particle](#) (double[] [solution](#), double range, int problem)
- void [setPBest](#) ([Particle pBest](#))

## Public Attributes

- double[] [solution](#)
- double [velocity](#)
- double [fitness](#)
- [Particle](#) [pBest](#)

### 7.5.1 Detailed Description

Definition at line 14 of file [Particle.java](#).

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 Particle()

```
project.Particle.Particle (
    double[] solution,
    double range,
    int problem )
```

Constructor for [Particle](#)

##### Parameters

<i>solution</i>	- solution vector
<i>range</i>	- range of values for the particle
<i>problem</i>	- problem type

Definition at line 34 of file [Particle.java](#).

### 7.5.3 Member Function Documentation

#### 7.5.3.1 setPBest()

```
void project.Particle.setPBest (
    Particle pBest )
```

Sets the pBest particle

##### Parameters

<i>pBest</i>	- the new pBest particle
--------------	--------------------------



Definition at line 49 of file [Particle.java](#).

## 7.5.4 Member Data Documentation

### 7.5.4.1 fitness

```
double project.Particle.fitness
```

Definition at line 22 of file [Particle.java](#).

### 7.5.4.2 pBest

```
Particle project.Particle.pBest
```

Definition at line 25 of file [Particle.java](#).

### 7.5.4.3 solution

```
double [] project.Particle.solution
```

Definition at line 16 of file [Particle.java](#).

### 7.5.4.4 velocity

```
double project.Particle.velocity
```

Definition at line 19 of file [Particle.java](#).

The documentation for this class was generated from the following file:

- project/[Particle.java](#)

## 7.6 project.Population Class Reference

### Public Member Functions

- [Population](#) (int *n*, int *m*, double *range*)
- double[] [getFitness](#) ()
- void [setFitness](#) (int *i*, double *fitness*)
- void [setSolution](#) (int *i*, double[] *sol*)
- double[][] [getPopulation](#) ()
- double[][] [getSolution](#) ()
- void [setPopulation](#) (double[][] *population*)
- double [getRange](#) ()
- double[][] [genRandomMatrix](#) (int *n*, int *m*)
- double[] [genRandomArray](#) (int *m*)
- double[][] [genNeighborhood](#) (int *n*, int *m*, double[] *solution*)

### 7.6.1 Detailed Description

Definition at line 14 of file [Population.java](#).

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 Population()

```
project.Population.Population (
    int n,
    int m,
    double range )
```

Constructor for the [Population](#) class.

#### Parameters

<i>n</i>	- number of experiments
<i>m</i>	- number of dimensions
<i>range</i>	- range of the values selected [-range, range]

Definition at line 35 of file [Population.java](#).

### 7.6.3 Member Function Documentation

### 7.6.3.1 genNeighborhood()

```
double[][] project.Population.genNeighborhood (
    int n,
    int m,
    double[] solution )
```

Returns a neighbor of the given solution vector

#### Parameters

<i>n</i>	- number of experiments
<i>m</i>	- number of dimensions
<i>solution</i>	- solution vector

#### Returns

- the neighborhood of the given solution vector

Definition at line 148 of file [Population.java](#).

### 7.6.3.2 genRandomArray()

```
double[] project.Population.genRandomArray (
    int m )
```

Returns an array of random values within the range

#### Parameters

<i>m</i>	- dimension of the array
----------	--------------------------

#### Returns

- array of random values

Definition at line 131 of file [Population.java](#).

### 7.6.3.3 genRandomMatrix()

```
double[][] project.Population.genRandomMatrix (
    int n,
    int m )
```

Creates an n x m matrix initialized to pseudo-random values

**Parameters**

$n$	- number of experiments
$m$	- number of dimensions

**Returns**

-  $n \times m$  matrix

Definition at line 114 of file [Population.java](#).

**7.6.3.4 getFitness()**

```
double[] project.Population.getFitness ( )
```

Returns the fitness vector

**Returns**

- fitnesses of the population

Definition at line 47 of file [Population.java](#).

**7.6.3.5 getPopulation()**

```
double[][] project.Population.getPopulation ( )
```

Returns the population matrix

**Returns**

- population matrix

Definition at line 76 of file [Population.java](#).

**7.6.3.6 getRange()**

```
double project.Population.getRange ( )
```

Returns the range of the initial population values

**Returns**

- range

Definition at line 103 of file [Population.java](#).

### 7.6.3.7 `getSolution()`

```
double[][] project.Population.getSolution ( )
```

Returns the solutions matrix

Returns

- solutions matrix

Definition at line 85 of file [Population.java](#).

### 7.6.3.8 `setFitness()`

```
void project.Population.setFitness (
    int i,
    double fitness )
```

Sets the value of the fitness at the index

Parameters

<i>i</i>	- index
<i>fitness</i>	- fitness value

Definition at line 57 of file [Population.java](#).

### 7.6.3.9 `setPopulation()`

```
void project.Population.setPopulation (
    double population[][] )
```

Sets the population matrix to the given matrix

Parameters

<i>population</i>	- given population matrix
-------------------	---------------------------

Definition at line 94 of file [Population.java](#).

### 7.6.3.10 `setSolution()`

```
void project.Population.setSolution (
    int i,
    double[] sol )
```

Sets the values of the solution vector at the index

#### Parameters

<i>i</i>	- index
<i>sol</i>	- solution vector

Definition at line 67 of file [Population.java](#).

The documentation for this class was generated from the following file:

- project/[Population.java](#)

## 7.7 project.Problem Class Reference

### Public Member Functions

- [Problem](#) (double[] vector, int probNum)
- double [getFitness](#) ()
- double [nthRoot](#) (int root, double value)
- double [square](#) (double value)
- double [schwefel](#) ()
- double [de\\_jong\\_1](#) ()
- double [rosenbrock](#) ()
- double [rastrigin](#) ()
- double [griewank](#) ()
- double [sine\\_envelope](#) ()
- double [sine\\_V](#) ()
- double [ackley\\_one](#) ()
- double [ackley\\_two](#) ()
- double [egg\\_holder](#) ()

### 7.7.1 Detailed Description

Definition at line 12 of file [Problem.java](#).

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 Problem()

```
project.Problem.Problem (
    double[] vector,
    int probNum )
```

Constructor for [Problem](#)

**Parameters**

<i>vector</i>	- vector of values
<i>probNum</i>	- problem type

Definition at line 25 of file [Problem.java](#).

### 7.7.3 Member Function Documentation

#### 7.7.3.1 `ackley_one()`

```
double project.Problem.ackley_one ( )
```

Returns the value of the Ackley One function using the values in the vector

**Returns**

- returns the fitness value of the Ackley One function

Definition at line 194 of file [Problem.java](#).

#### 7.7.3.2 `ackley_two()`

```
double project.Problem.ackley_two ( )
```

Returns the value of the Ackley Two function using the values in the vector

**Returns**

- returns the fitness value of the Ackley Two function

Definition at line 209 of file [Problem.java](#).

#### 7.7.3.3 `de_jong_1()`

```
double project.Problem.de_jong_1 ( )
```

Returns the value of the De Jong 1 function using the values in the vector

**Returns**

- returns the fitness value of the De Jong 1 function

Definition at line 109 of file [Problem.java](#).

#### 7.7.3.4 egg\_holder()

```
double project.Problem.egg_holder ( )
```

Returns the value of the Egg Holder function using the values in the vector

##### Returns

- returns the fitness value of the Egg Holder function

Definition at line 226 of file [Problem.java](#).

#### 7.7.3.5 getFitness()

```
double project.Problem.getFitness ( )
```

Returns the fitness value of the problem

##### Returns

- returns the fitness value of the problem

Definition at line 66 of file [Problem.java](#).

#### 7.7.3.6 griewank()

```
double project.Problem.griewank ( )
```

Returns the value of the Griewank function using the values in the vector

##### Returns

- returns the fitness value of the Griewank function

Definition at line 148 of file [Problem.java](#).

#### 7.7.3.7 nthRoot()

```
double project.Problem.nthRoot (
    int root,
    double value )
```

Returns the nth root of the input value



**Parameters**

<i>root</i>	- the root to use
<i>value</i>	- the value to use

**Returns**

- returns decimal form of the nth root of the input value

Definition at line 77 of file [Problem.java](#).

**7.7.3.8 rastrigin()**

```
double project.Problem.rastrigin ( )
```

Returns the value of the Rastrigin function using the values in the vector

**Returns**

- returns the fitness value of the Rastrigin function

Definition at line 135 of file [Problem.java](#).

**7.7.3.9 rosenbrock()**

```
double project.Problem.rosenbrock ( )
```

Returns the value of the Rosenbrock function using the values in the vector

**Returns**

- returns the fitness value of the Rosenbrock function

Definition at line 122 of file [Problem.java](#).

**7.7.3.10 schwefel()**

```
double project.Problem.schwefel ( )
```

Returns the value of the Schwefel function using the values in the vector

**Returns**

- returns the fitness value of the Schwefel function

Definition at line 96 of file [Problem.java](#).

#### 7.7.3.11 `sine_envelope()`

```
double project.Problem.sine_envelope ( )
```

Returns the value of the Sine Envelope function using the values in the vector

##### Returns

- returns the fitness value of the Sine Envelope function

Definition at line 164 of file [Problem.java](#).

#### 7.7.3.12 `sine_V()`

```
double project.Problem.sine_V ( )
```

Returns the value of the Sine V function using the values in the vector

##### Returns

- returns the fitness value of the Sine V function

Definition at line 179 of file [Problem.java](#).

#### 7.7.3.13 `square()`

```
double project.Problem.square (
    double value )
```

Returns the square of the input value

##### Parameters

<i>value</i>	- the value to use
--------------	--------------------

##### Returns

- returns the decimal form of the square of the input value

Definition at line 87 of file [Problem.java](#).

The documentation for this class was generated from the following file:

- [project/Problem.java](#)

## Chapter 8

# File Documentation

### 8.1 Main.java File Reference

#### Classes

- class [Main](#)

### 8.2 Main.java

[Go to the documentation of this file.](#)

```
00001 import project.Population;
00002 import project.Algorithms;
00003 import project.NEH;
00004
00005 import java.util.*;
00006 import java.io.*;
00007 import java.util.stream.LongStream;
00008
00009 public class Main {
00010
00011     public static void main(String[] args) {
00012         Scanner sc = new Scanner(System.in);
00013         System.out.println("Types of functions: minimization or scheduling.");
00014         String type = "";
00015
00019         while (!type.equals("minimization") && !type.equals("scheduling")) {
00020             System.out.print("Enter the wanted optimization functions type: ");
00021             type = sc.nextLine();
00022         }
00023
00024         int function = type == "minimization" ? 0 : 1;
00025
00031         switch (function) {
00032             case 0:
00033                 minimizeDriver();
00034             case 1:
00035                 scheduleDriver();
00036         }
00037
00038         sc.close();
00039     }
00040
00044     public static void minimizeDriver() {
00045         try {
00053             BufferedReader br = readFromFile("experiments.txt");
00054             BufferedWriter bw = makeOutFile();
00055
00056             // Reads line from input file
00057             String line = br.readLine();
00058
00059             // Main loop for conducting experiments
```

```

00060         fileLoop(line, br, bw);
00061
00062         // Close input and output files when done
00063         bw.close();
00064         br.close();
00065
00066     } catch (IOException e) {
00067
00068         // If there is an error, print it
00069         e.printStackTrace();
00070     }
00071 }
00072
00073 public static void scheduleDriver() {
00074     try {
00075         // Reading from the input file and creating the output file
00076         BufferedWriter bw = makeOutFile();
00077         // Loop for each algorithm (FSS and FSSB)
00078         for (int a = 0; a < 2; a++) {
00079             // Writing the header of the output file
00080             bw.write("Machines" + "," + "Jobs" + "," + "Makespan" + "," + "Time" + "," +
00081 "Results("
00082                 + (a == 0 ? "FSS" : "FSSB") + ")\n");
00083             // Loop to process all the input files
00084             for (int i = 1; i <= 120; i++) {
00085                 ArrayList<String> lines = readLines(i);
00086
00087                 // Getting the number of machines and jobs from the first line
00088                 String[] parts = lines.get(0).split(" ");
00089
00090                 // Number of machines
00091                 int m = Integer.parseInt(parts[0]);
00092
00093                 // Number of jobs
00094                 int n = Integer.parseInt(parts[1]);
00095
00096                 // Creating the processing times matrix
00097                 int[][] times = new int[m][n];
00098                 for (int j = 1; j < lines.size(); j++) {
00099                     String[] lineParts = lines.get(j).split(" ");
00100                     for (int k = 0; k < n; k++) {
00101                         times[j - 1][k] = Integer.parseInt(lineParts[k]);
00102                     }
00103                 }
00104
00105                 // Run experiment for given inputs
00106                 scheduleLoop(times, m, n, a, bw);
00107             }
00108         }
00109         bw.close();
00110     } catch (IOException e) {
00111         // If there is an error, print it
00112         e.printStackTrace();
00113     }
00114 }
00115
00116 public static void fileLoop(String line, BufferedReader br, BufferedWriter bw) {
00117     try {
00118         // Loop until end of input file
00119         while (line != null) {
00120
00121             // Get experiment parameters
00122             String[] parts = line.split(" ");
00123
00124             int algorithm = Integer.parseInt(parts[0]);
00125             int method = Integer.parseInt(parts[1]);
00126             int crosstype = Integer.parseInt(parts[2]);
00127             int m = Integer.parseInt(parts[3]);
00128             int n = Integer.parseInt(parts[4]);
00129             int problem = Integer.parseInt(parts[5]);
00130             double range = Double.parseDouble(parts[6]);
00131             int numExperiments = Integer.parseInt(parts[7]);
00132
00133             // Initialize population
00134             Population pop = new Population(n, m, range);
00135             long[] times = new long[n];
00136
00137             // Run the experiments
00138             experiment(numExperiments, pop, times, algorithm, problem, method, crosstype);
00139
00140             // Sum time array for total time for the experiment
00141             long sum = LongStream.of(times).sum();
00142
00143             // Save experiment in a CSV file
00144             writeFile(bw, problem, numExperiments, m, algorithm, range, sum, pop, method,
00145 crosstype);
00146         }
00147     }

```

```

00167         // Read next line from input file
00168         line = br.readLine();
00169     }
00170 } catch (IOException e) {
00171
00172     // If there is an error, print it
00173     e.printStackTrace();
00174 }
00175 }
00176
00189 public static void experiment(int n, Population pop, long[] times, int algorithm, int problem, int
method,
00190     int crosstype) {
00191     for (int i = 0; i < n; i++) {
00192         long start = System.nanoTime();
00193         Algorithms alg = new Algorithms(algorithm, pop, problem, crosstype, method, i);
00194         System.out.println(i);
00195         times[i] = System.nanoTime() - start;
00196         pop.setFitness(i, alg.getFitness());
00197         pop.setSolution(i, alg.getSolution());
00198     }
00199 }
00200
00210 public static void scheduleLoop(int[][] times, int m, int n, int alg, BufferedWriter bw) {
00211     // Start timer
00212     long time = System.nanoTime();
00213     // Run the experiment
00214     NEH algos = new NEH(times, m, n, alg);
00215     // Stop timer
00216     time = System.nanoTime() - time;
00217     // Write the results to the output file
00218     writeResult(algos, bw, m, n, time);
00219 }
00220
00228 public static ArrayList<String> readLines(int i) {
00229     ArrayList<String> lines = new ArrayList<String>();
00230     try {
00231         BufferedReader br = readFromFile("./project/Taillard_TestData/" + i + ".txt");
00232         String line = br.readLine();
00233         while (line != null) {
00234             lines.add(line);
00235             line = br.readLine();
00236         }
00237         br.close();
00238     } catch (IOException e) {
00239         e.printStackTrace();
00240     }
00241     return lines;
00242 }
00243
00258 public static void writeFile(BufferedWriter bw, int problem, int n, int m, int alg, double range,
long sum,
00259     Population pop, int method, int crosstype) {
00260     try {
00261         // Gets the algorithm type
00262         String algType = (alg == 1) ? "" : "Partical Swarm Optimization";
00263         String meth = "";
00264         // Gets the method type for DE
00265         switch (method) {
00266             case 1:
00267                 meth = "DE/Best/1";
00268                 break;
00269             case 2:
00270                 meth = "DE/Rand/1";
00271                 break;
00272             case 3:
00273                 meth = "DE/Rand-To-Best/1";
00274                 break;
00275             case 4:
00276                 meth = "DE/Best/2";
00277                 break;
00278             case 5:
00279                 meth = "DE/Rand/2";
00280                 break;
00281             default:
00282                 break;
00283         }
00284         // Gets the crossover type for DE
00285         String cross = (crosstype == 1) ? "exp" : (crosstype == 2) ? "bin" : "";
00286
00287         // Writes the summary of the experiments
00288         bw.write("Problem " + problem + " with " + n + " experiments of dimension " + m + " in
range [-" + range
00289             + " : " + range + "]"
00290             + " using the " + algType + meth + cross + " algorithm that took " + (double) sum
/ 1000000
00291             + "milliseconds to run" + ",");

```

```

00292
00293         // Writes the fitness values of the population
00294         for (int i = 0; i < n; i++) {
00295             bw.write(pop.getFitness()[i] + ",");
00296         }
00297         bw.write("\n");
00298     } catch (IOException e) {
00299         e.printStackTrace();
00300     }
00301 }
00302
00303
00313 public static void writeResult(NEH algos, BufferedWriter bw, int m, int n, long time) {
00314     // Prints the results to the output file
00315     try {
00316         // Writing # machines, # jobs, makespan, time, results
00317         bw.write(m + "," + n + "," + algos.makespan + "," + (double) time / 1000000 + ",");
00318         for (int i = 0; i < n; i++) {
00319             bw.write(algos.schedule[i] + 1 + ",");
00320         }
00321         bw.write("\n");
00322     } catch (IOException e) {
00323         e.printStackTrace();
00324     }
00325 }
00326
00335 public static BufferedReader readFromFile(String filename) {
00336     BufferedReader br = null;
00337     try {
00338         File experiments = new File(filename);
00339         br = new BufferedReader(new FileReader(experiments));
00340     } catch (FileNotFoundException e) {
00341         e.printStackTrace();
00342     }
00343     return br;
00344 }
00345
00352 public static BufferedWriter makeOutFile() {
00353     BufferedWriter bw = null;
00354     try {
00355         bw = new BufferedWriter(new FileWriter("experiments" + System.currentTimeMillis() +
00356             ".csv"));
00357     } catch (IOException e) {
00358         e.printStackTrace();
00359     }
00360     return bw;
00361 }
00362 }

```

## 8.3 mt/MTRandom.java File Reference

### Classes

- class [mt.MTRandom](#)

### Packages

- package [mt](#)

## 8.4 MTRandom.java

[Go to the documentation of this file.](#)

```

00001 /*
00002  * MTRandom : A Java implementation of the MT19937 (Mersenne Twister)
00003  *             pseudo random number generator algorithm based upon the
00004  *             original C code by Makoto Matsumoto and Takuji Nishimura.
00005  * Author    : David Beaumont
00006  * Email     : mersenne-at-www.goui.net
00007  *

```

```

00008  * For the original C code, see:
00009  *   http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
00010  *
00011  * This version, Copyright (C) 2005, David Beaumont.
00012  *
00013  * This library is free software; you can redistribute it and/or
00014  * modify it under the terms of the GNU Lesser General Public
00015  * License as published by the Free Software Foundation; either
00016  * version 2.1 of the License, or (at your option) any later version.
00017  *
00018  * This library is distributed in the hope that it will be useful,
00019  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00020  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
00021  * Lesser General Public License for more details.
00022  *
00023  * You should have received a copy of the GNU Lesser General Public
00024  * License along with this library; if not, write to the Free Software
00025  * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00026  */
00027 package mt;
00028
00029 import java.util.Random;
00030
00088 public class MTRandom extends Random {
00089
00096     private static final long serialVersionUID = -515082678588212038L;
00097
00098     // Constants used in the original C implementation
00099     private final static int UPPER_MASK = 0x80000000;
00100     private final static int LOWER_MASK = 0x7fffffff;
00101
00102     private final static int N = 624;
00103     private final static int M = 397;
00104     private final static int MAGIC[] = { 0x0, 0x9908b0df };
00105     private final static int MAGIC_FACTOR1 = 1812433253;
00106     private final static int MAGIC_FACTOR2 = 1664525;
00107     private final static int MAGIC_FACTOR3 = 1566083941;
00108     private final static int MAGIC_MASK1 = 0x9d2c5680;
00109     private final static int MAGIC_MASK2 = 0xefc60000;
00110     private final static int MAGIC_SEED = 19650218;
00111     private final static long DEFAULT_SEED = 5489L;
00112
00113     // Internal state
00114     private transient int[] mt;
00115     private transient int mti;
00116     private transient boolean compat = false;
00117
00118     // Temporary buffer used during setSeed(long)
00119     private transient int[] ibuf;
00120
00127     public MTRandom() {
00128     }
00129
00150     public MTRandom(boolean compatible) {
00151         super(0L);
00152         compat = compatible;
00153         setSeed(compat ? DEFAULT_SEED : System.currentTimeMillis());
00154     }
00155
00163     public MTRandom(long seed) {
00164         super(seed);
00165     }
00166
00176     public MTRandom(byte[] buf) {
00177         super(0L);
00178         setSeed(buf);
00179     }
00180
00190     public MTRandom(int[] buf) {
00191         super(0L);
00192         setSeed(buf);
00193     }
00194
00195     // Initializes mt[N] with a simple integer seed. This method is
00196     // required as part of the Mersenne Twister algorithm but need
00197     // not be made public.
00198     private final void setSeed(int seed) {
00199
00200         // Annoying runtime check for initialisation of internal data
00201         // caused by java.util.Random invoking setSeed() during init.
00202         // This is unavoidable because no fields in our instance will
00203         // have been initialised at this point, not even if the code
00204         // were placed at the declaration of the member variable.
00205         if (mt == null)
00206             mt = new int[N];
00207
00208         // ---- Begin Mersenne Twister Algorithm ----

```

```

00209         mt[0] = seed;
00210         for (mti = 1; mti < N; mti++) {
00211             mt[mti] = (MAGIC_FACTOR1 * (mt[mti - 1] ^ (mt[mti - 1] >> 30)) + mti);
00212         }
00213         // ---- End Mersenne Twister Algorithm ----
00214     }
00215
00231     public final synchronized void setSeed(long seed) {
00232         if (compat) {
00233             setSeed((int) seed);
00234         } else {
00235
00236             // Annoying runtime check for initialisation of internal data
00237             // caused by java.util.Random invoking setSeed() during init.
00238             // This is unavoidable because no fields in our instance will
00239             // have been initialised at this point, not even if the code
00240             // were placed at the declaration of the member variable.
00241             if (ibuf == null)
00242                 ibuf = new int[2];
00243
00244             ibuf[0] = (int) seed;
00245             ibuf[1] = (int) (seed >> 32);
00246             setSeed(ibuf);
00247         }
00248     }
00249
00266     public final void setSeed(byte[] buf) {
00267         setSeed(pack(buf));
00268     }
00269
00279     public final synchronized void setSeed(int[] buf) {
00280         int length = buf.length;
00281         if (length == 0)
00282             throw new IllegalArgumentException("Seed buffer may not be empty");
00283         // ---- Begin Mersenne Twister Algorithm ----
00284         int i = 1, j = 0, k = (N > length ? N : length);
00285         setSeed(MAGIC_SEED);
00286         for (; k > 0; k--) {
00287             mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 30)) * MAGIC_FACTOR2)) + buf[j] + j;
00288             i++;
00289             j++;
00290             if (i >= N) {
00291                 mt[0] = mt[N - 1];
00292                 i = 1;
00293             }
00294             if (j >= length)
00295                 j = 0;
00296         }
00297         for (k = N - 1; k > 0; k--) {
00298             mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 30)) * MAGIC_FACTOR3)) - i;
00299             i++;
00300             if (i >= N) {
00301                 mt[0] = mt[N - 1];
00302                 i = 1;
00303             }
00304         }
00305         mt[0] = UPPER_MASK; // MSB is 1; assuring non-zero initial array
00306         // ---- End Mersenne Twister Algorithm ----
00307     }
00308
00336     protected final synchronized int next(int bits) {
00337         // ---- Begin Mersenne Twister Algorithm ----
00338         int y, kk;
00339         if (mti >= N) { // generate N words at one time
00340
00341             // In the original C implementation, mti is checked here
00342             // to determine if initialisation has occurred; if not
00343             // it initialises this instance with DEFAULT_SEED (5489).
00344             // This is no longer necessary as initialisation of the
00345             // Java instance must result in initialisation occurring
00346             // Use the constructor MTRandom(true) to enable backwards
00347             // compatible behaviour.
00348
00349             for (kk = 0; kk < N - M; kk++) {
00350                 y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
00351                 mt[kk] = mt[kk + M] ^ (y >> 1) ^ MAGIC[y & 0x1];
00352             }
00353             for (; kk < N - 1; kk++) {
00354                 y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
00355                 mt[kk] = mt[kk + (M - N)] ^ (y >> 1) ^ MAGIC[y & 0x1];
00356             }
00357             y = (mt[N - 1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
00358             mt[N - 1] = mt[M - 1] ^ (y >> 1) ^ MAGIC[y & 0x1];
00359             mti = 0;
00360         }
00361     }
00362

```



```

00363         y = mt[mti++];
00364
00365         // Tempering
00366         y ^= (y >> 11);
00367         y ^= (y << 7) & MAGIC_MASK1;
00368         y ^= (y << 15) & MAGIC_MASK2;
00369         y ^= (y >> 18);
00370         // ---- End Mersenne Twister Algorithm ----
00371         return (y >> (32 - bits));
00372     }
00373
00374     // This is a fairly obscure little code section to pack a
00375     // byte[] into an int[] in little endian ordering.
00376
00407     public static int[] pack(byte[] buf) {
00408         int k, blen = buf.length, ilen = ((buf.length + 3) >> 2);
00409         int[] ibuf = new int[ilen];
00410         for (int n = 0; n < ilen; n++) {
00411             int m = (n + 1) << 2;
00412             if (m > blen)
00413                 m = blen;
00414             for (k = buf[--m] & 0xff; (m & 0x3) != 0; k = (k << 8) | buf[--m] & 0xff)
00415                 ;
00416             ibuf[n] = k;
00417         }
00418         return ibuf;
00419     }
00420 }

```

## 8.5 project/Algorithms.java File Reference

### Classes

- class [project.Algorithms](#)

### Packages

- package [project](#)

## 8.6 Algorithms.java

[Go to the documentation of this file.](#)

```

00001 /*
00002  * Algorithms : A Java implementation of optimization algorithms: Blind Search,
00003  *             Iterated Local Search, Differential Evolution, Particle Swarm
00004  *             Optimization.
00005  * Author      : Ethan Krug
00006  * Email       : ethan.c.krug@gmail.com
00007  * Date        : May, 2022
00008  *
00009  * Copyright (C) 2022 Ethan Krug
00010  */
00011 package project;
00012
00013 import mt.MTRandom;
00014
00015 public class Algorithms {
00016     // The population of the algorithm
00017     private Population population;
00018
00019     // The number of dimensions
00020     private int m;
00021
00022     // The number of experiments/iterations
00023     private int n;
00024
00025     // The solution vectors in the population
00026     private double[][] popMatrix;
00027
00028     // The problem type of the algorithm

```

```

00029     private int problem;
00030
00031     // The crossover method for the algorithm
00032     private int crosstype;
00033
00034     // The solution vector of the algorithm
00035     private double[] solution;
00036
00037     // Initialization of the fitness for the current algorithm
00038     private double fitness = Double.MAX_VALUE;
00039
00040     // Random number generator
00041     private MTRandom r;
00042
00051     public Algorithms(int algorithm, Population pop, int problem, int crosstype, int method, int
index) {
00052         this.population = pop;
00053         this.m = pop.getPopulation()[0].length;
00054         this.n = pop.getPopulation().length;
00055         popMatrix = pop.getPopulation();
00056         this.problem = problem;
00057         this.crosstype = crosstype;
00058         this.r = new MTRandom(false);
00059         switch (algorithm) {
00060             case 1:
00061                 this.solution = DE(method, popMatrix[0].length, popMatrix.length, 0.6, 0.9, 0.8, 100);
00062                 break;
00063             case 2:
00064                 Particle p = PSO(100, popMatrix.length, popMatrix[0].length, 0.8, 1.2);
00065                 this.solution = p.solution;
00066                 break;
00067             case 3:
00068                 this.solution = blindSearch(n, pop.getPopulation()[index], fitness);
00069                 break;
00070             case 4:
00071                 this.solution = repeatedLocalSearch(pop.getPopulation()[index], null, null, false, n);
00072                 break;
00073         }
00074     }
00075
00079     public double[] getSolution() {
00080         return solution;
00081     }
00082
00086     public double getFitness() {
00087         Problem p = new Problem(solution, problem);
00088         return p.getFitness();
00089     }
00090
00099     public double[] blindSearch(int iterations, double[] bestSol, double fitness) {
00100         for (int i = 0; i < iterations; i++) {
00101             // Gets a random solution
00102             double[] arg = population.genRandomArray(m);
00103
00104             // Evaluate the fitness of the random solution
00105             Problem prob = new Problem(arg, problem);
00106
00107             // Get the fitness of the solution
00108             double fitnessNew = prob.getFitness();
00109
00110             // If the fitness is better than the current best solution,
00111             // update the best solution
00112             if (fitnessNew < fitness) {
00113                 fitness = fitnessNew;
00114                 bestSol = arg;
00115             }
00116         }
00117         this.fitness = fitness;
00118         return bestSol;
00119     }
00120
00130     public double[] localSearch(double[] initialSol, double[] bestSol, boolean tau) {
00131         // Run algorithm until the solution doesn't improve
00132         while (tau) {
00133             tau = false;
00134
00135             // Generate the neighborhood of the current solution
00136             double[][] neighborhood = population.genNeighborhood(n, m, initialSol);
00137
00138             // Initialize best neighborhood fitness and solution
00139             double bestFitnessInNeighborhood = Double.MAX_VALUE;
00140             double[] bestSolutionInNeighborhood = new double[m];
00141
00142             // Evaluate the fitness of the neighborhood
00143             for (int i = 0; i < neighborhood.length; i++) {
00144                 Problem prob = new Problem(neighborhood[i], problem);
00145                 double fitnessNew = prob.getFitness();

```

```

00146
00147         // If the fitness is better than the current best solution in neighborhood,
00148         // update the best solution in neighborhood
00149         if (fitnessNew <= this.fitness) {
00150             bestFitnessInNeighborhood = fitnessNew;
00151             bestSolutionInNeighborhood = neighborhood[i];
00152         }
00153     }
00154
00155     // If the best solution in the neighborhood is better than the current best
00156     // solution, update the best solution
00157     if (bestFitnessInNeighborhood < this.fitness) {
00158         this.fitness = bestFitnessInNeighborhood;
00159         bestSol = bestSolutionInNeighborhood;
00160         tau = true;
00161     }
00162 }
00163 return bestSol;
00164 }
00165
00177 public double[] repeatedLocalSearch(double[] initialSol, double[] bestGlobalSol, double[]
bestIterSol, boolean tau,
00178     int iterations) {
00179
00180     // Initialize best solutions
00181     bestGlobalSol = initialSol;
00182     bestIterSol = initialSol;
00183     tau = true;
00184
00185     // Iteration counter
00186     int t = 1;
00187
00188     // Run the algorithm until all iterations are complete
00189     while (t <= iterations) {
00190         // Get the local search solution for current iteration
00191         bestIterSol = localSearch(bestGlobalSol, bestIterSol, tau);
00192
00193         // Get fitness of current iteration and global iteration
00194         Problem prob = new Problem(bestIterSol, problem);
00195         Problem probGlobal = new Problem(bestGlobalSol, problem);
00196
00197         // If the fitness of the current iteration is better than the global
00198         // solution, update the global solution
00199         if (prob.getFitness() < probGlobal.getFitness()) {
00200             bestGlobalSol = bestIterSol;
00201         }
00202         t++;
00203         bestIterSol = population.genRandomArray(m);
00204     }
00205     return bestGlobalSol;
00206 }
00207
00220 public double[] DE(int method, int D, int NP, double CR, double F, double lambda, int generations)
{
00221     int generation = 0;
00222     while (generation < generations) {
00223         // Iterate over every solution in the population
00224         for (int i = 0; i < NP; i++) {
00225
00226             // Getting the randomly selected vector indexes
00227             int r1 = i, r2 = i, r3 = i, r4 = i, r5 = i, jrand = r.nextInt(D);
00228             while (same(i, r1, r2, r3, r4, r5)) {
00229                 r1 = r.nextInt(NP);
00230                 r2 = r.nextInt(NP);
00231                 r3 = r.nextInt(NP);
00232                 r4 = r.nextInt(NP);
00233                 r5 = r.nextInt(NP);
00234             }
00235
00236             // noisy vector
00237             double[] u = new double[D];
00238
00239             // Mutation of noisy vector
00240             boolean crossed = false;
00241             while (!crossed) {
00242                 for (int k = 0; k < D; k++) {
00243                     if (r.nextDouble() < CR || k == jrand) {
00244                         u[k] = method(method, lambda, F, r1, r2, r3, r4, r5, k, i);
00245                         crossed = true;
00246                     } else {
00247                         u[k] = popMatrix[i][k];
00248                     }
00249                 }
00250                 if (crosstype == 2) {
00251                     crossed = true;
00252                 }
00253             }

```

```

00254
00255         // Selection
00256         Problem p = new Problem(u, problem);
00257         Problem x = new Problem(popMatrix[i], problem);
00258         if (Math.abs(p.getFitness()) <= Math.abs(x.getFitness())) {
00259             popMatrix[i] = u;
00260         }
00261     }
00262     generation++;
00263 }
00264     return popMatrix[bestSol()];
00265 }
00266
00278     private boolean same(int i, int r1, int r2, int r3, int r4, int r5) {
00279         return i == r1 || i == r2 || i == r3 || i == r4 || i == r5 || r1 == r2 || r1 == r3 || r1 == r4
00280             || r1 == r5 || r2 == r3 || r2 == r4 || r2 == r5 || r3 == r4 || r3 == r5 || r4 == r5;
00281     }
00282
00298     private double method(int method, double lambda, double F, int r1, int r2, int r3, int r4, int r5,
00299 int k, int i) {
00300         switch (method) {
00301             case 1: // DE/best/1
00302                 return popMatrix[bestSol()][k] + F * (popMatrix[r1][k] - popMatrix[r2][k]);
00303             case 2: // DE/rand/1
00304                 return popMatrix[r1][k] + F * (popMatrix[r2][k] - popMatrix[r3][k]);
00305             case 3: // DE/rand-to-best/1
00306                 return popMatrix[i][k] + lambda * (popMatrix[bestSol()][k] - popMatrix[i][k])
00307                     + F * (popMatrix[r1][k] - popMatrix[r2][k]);
00308             case 4: // DE/best/2
00309                 return popMatrix[bestSol()][k]
00310                     + F * (popMatrix[r1][k] + popMatrix[r2][k] - popMatrix[r3][k] -
00311 popMatrix[r4][k]);
00312             case 5: // DE/rand/2
00313                 return popMatrix[r5][k]
00314                     + F * (popMatrix[r1][k] + popMatrix[r2][k] - popMatrix[r3][k] -
00315 popMatrix[r4][k]);
00316             default: // Invalid method
00317                 return Double.NaN;
00318         }
00319     }
00320
00323     private int bestSol() {
00324         int best = 0;
00325         for (int i = 0; i < popMatrix.length; i++) {
00326             Problem p = new Problem(popMatrix[i], problem);
00327             Problem b = new Problem(popMatrix[best], problem);
00328             if (Math.abs(p.getFitness()) <= Math.abs(b.getFitness())) {
00329                 best = i;
00330             }
00331         }
00332         return best;
00333     }
00334
00345     public Particle PSO(int iterations, int numParticles, int dimensions, double c1, double c2) {
00346         // Get range of values
00347         double range = population.getRange();
00348
00349         // Initialize the particles
00350         Particle[] particles = new Particle[numParticles];
00351         for (int i = 0; i < numParticles; i++) {
00352             particles[i] = new Particle(popMatrix[i], range, problem);
00353             particles[i].setPBest(particles[i]);
00354         }
00355
00356         // Initialize the best particle
00357         Particle gBest = particles[0];
00358         for (int i = 1; i < numParticles; i++) {
00359             if (Math.abs(particles[i].fitness) < gBest.fitness) {
00360                 gBest = particles[i];
00361             }
00362         }
00363
00364         // Runs the main part of the PSO algorithm
00365         for (int t = 0; t < iterations; t++) {
00366             // Update the states of every particle in the swarm
00367             for (int j = 0; j < numParticles; j++) {
00368                 // Update the velocity and position of the particles
00369                 for (int k = 0; k < dimensions; k++) {
00370                     // Calculate the new velocity
00371                     double addVel = particles[j].velocity
00372                         + c1 * r.nextDouble() * (particles[j].pBest.solution[k] -
00373 particles[j].solution[k])
00374                         + c2 * r.nextDouble() * (gBest.solution[k] - particles[j].solution[k]);
00375                     // Add the new velocity to the current position
00376                     particles[j].solution[k] += addVel;
00377                 }
00378             }
00379         }
00380     }

```

```

00378             // Update the pBest and fitness of the particle
00379             Problem p = new Problem(particles[j].solution, problem);
00380             if (Math.abs(p.getFitness()) < Math.abs(particles[j].fitness)) {
00381                 particles[j].pBest = particles[j];
00382             }
00383             particles[j].fitness = p.getFitness();
00384
00385             // Update the gBest
00386             if (Math.abs(particles[j].fitness) < gBest.fitness) {
00387                 gBest = particles[j];
00388             }
00389         }
00390     }
00391     return gBest;
00392 }
00393 }

```

## 8.7 project/NEH.java File Reference

### Classes

- class [project.NEH](#)

### Packages

- package [project](#)

## 8.8 NEH.java

[Go to the documentation of this file.](#)

```

00001 /*
00002  * Algorithms : A Java implementation of optimization algorithm:
00003  *              Nawaz-Enscore-Ham (NEH) permutation flow shop
00004  *              scheduling algorithm. Implements two makespan
00005  *              algorithms: Flow Shop Scheduling (FSS),
00006  *              Flow Shop Scheduling with Blocking (FSSB).
00007  * Author      : Ethan Krug
00008  * Email       : ethan.c.krug@gmail.com
00009  * Date        : June, 2022
00010  *
00011  * Copyright (C) 2022 Ethan Krug
00012  */
00013 package project;
00014
00015 public class NEH {
00016     // Hold the processing times of each job on each machine
00017     public int[][] matrix;
00018
00019     // The schedule determined by the algorithm
00020     public int[] schedule;
00021
00022     // The makespan of the schedule
00023     public int makespan;
00024
00025     // The number of machines
00026     public int m;
00027
00028     // The number of jobs
00029     public int n;
00030
00039     public NEH(int[][] times, int m, int n, int alg) {
00040         matrix = times;
00041         schedule = new int[n];
00042         this.m = m;
00043         this.n = n;
00044         makespan = runNEH(alg);
00045     }
00046
00053     public int runNEH(int alg) {
00054         // Total processing time for each job

```

```

00055         int[] totals = new int[n];
00056
00057         // Get the total processing time for each job
00058         for (int i = 0; i < n; i++) {
00059             totals[i] = 0;
00060             for (int j = 0; j < m; j++) {
00061                 totals[i] += matrix[j][i];
00062             }
00063         }
00064
00065         // Sort the jobs by total processing time in descending order
00066         int[] sorted = new int[n];
00067         for (int i = 0; i < n; i++) {
00068             sorted[i] = i;
00069         }
00070         for (int i = 0; i < n; i++) {
00071             for (int j = i + 1; j < n; j++) {
00072                 if (totals[sorted[i]] < totals[sorted[j]]) {
00073                     int temp = sorted[i];
00074                     sorted[i] = sorted[j];
00075                     sorted[j] = temp;
00076                 }
00077             }
00078         }
00079
00080         // Transpose matrix for ease of use
00081         int[][] transMatrix = transposeMatrix(matrix);
00082
00083         // Current number of jobs in the schedule
00084         int L = 2;
00085
00086         // Initialize the schedule and the makespan
00087         schedule[0] = sorted[0];
00088         int currMake = -1;
00089
00090         // Find the best schedule with the shortest makespan
00091         while (L < n + 1) {
00092             // Best schedule and makespan for the current L
00093             int[] best = new int[L];
00094             int bestScore = Integer.MAX_VALUE;
00095
00096             // For every possible schedule of length L
00097             for (int i = 0; i < L; i++) {
00098                 // Creating job order
00099                 int[] order = new int[L];
00100                 int newJob = sorted[L - 1];
00101                 order[i] = newJob;
00102                 for (int j = 0, k = 0; j < L; j++) {
00103                     if (i != j) {
00104                         order[j] = schedule[k];
00105                         k++;
00106                     }
00107                 }
00108
00109                 // Calculating makespan
00110                 int[][] tempMatrix = new int[L][m];
00111                 for (int j = 0; j < L; j++) {
00112                     tempMatrix[j] = transMatrix[order[j]];
00113                 }
00114                 tempMatrix = transposeMatrix(tempMatrix);
00115                 int makespan = alg == 0 ? FSS(tempMatrix, m, L) : FSSB(tempMatrix, m, L);
00116
00117                 // If the makespan is better than the best one, update the best
00118                 if (makespan < bestScore) {
00119                     bestScore = makespan;
00120                     best = order;
00121                 }
00122             }
00123
00124             // Update the schedule
00125             for (int i = 0; i < L; i++) {
00126                 schedule[i] = best[i];
00127             }
00128
00129             // Update the best makespan for the current L
00130             currMake = bestScore;
00131
00132             // Increase number of jobs in the schedule
00133             L += 1;
00134         }
00135
00136         return currMake;
00137     }
00138
00147     public static int[][] transposeMatrix(int[][] matrix) {
00148         int m = matrix.length;
00149         int n = matrix[0].length;

```

```

00150
00151     int[][] transposedMatrix = new int[n][m];
00152
00153     for (int x = 0; x < n; x++) {
00154         for (int y = 0; y < m; y++) {
00155             transposedMatrix[x][y] = matrix[y][x];
00156         }
00157     }
00158
00159     return transposedMatrix;
00160 }
00161
00170 public int FSS(int[][] times, int m, int n) {
00171     int[][] makespan = new int[m][n];
00172     makespan[0][0] = times[0][0];
00173
00174     // Processing job 1 on all machines
00175     for (int i = 1; i < m; i++) {
00176         makespan[i][0] = makespan[i - 1][0] + times[i][0];
00177     }
00178
00179     // Processing all jobs on machine 1
00180     for (int j = 1; j < n; j++) {
00181         makespan[0][j] = makespan[0][j - 1] + times[0][j];
00182     }
00183
00184     // Processing all jobs on all machines
00185     for (int i = 1; i < m; i++) {
00186         for (int j = 1; j < n; j++) {
00187             makespan[i][j] = Math.max(makespan[i - 1][j], makespan[i][j - 1]) + times[i][j];
00188         }
00189     }
00190
00191     return makespan[m - 1][n - 1];
00192 }
00193
00202 public int FSSB(int[][] times, int m, int n) {
00203     int[][] makespan = new int[m][n];
00204     makespan[0][0] = times[0][0];
00205
00206     // Processing job 1 on all machines
00207     for (int i = 1; i < m; i++) {
00208         makespan[i][0] = makespan[i - 1][0] + times[i][0];
00209     }
00210
00211     // Finding departure times for all jobs on all machines
00212     for (int j = 1; j < n; j++) {
00213         for (int i = 1; i < m - 1; i++) {
00214             makespan[i][j] = Math.max(makespan[i - 1][j] + times[i][j], makespan[i + 1][j - 1]);
00215         }
00216         makespan[m - 1][j] = makespan[m - 2][j] + times[m - 1][j];
00217     }
00218
00219     return makespan[m - 1][n - 1];
00220 }
00221 }

```

## 8.9 project/Particle.java File Reference

### Classes

- class [project.Particle](#)

### Packages

- package [project](#)

## 8.10 Particle.java

[Go to the documentation of this file.](#)

```

00001  /*
00002   * Particle : A particle object implemented for the Particle Swarm Optimization
00003   *             algorithm.
00004   * Author    : Ethan Krug
00005   * Email     : ethan.c.krug@gmail.com
00006   * Date      : May, 2022
00007   *
00008   * Copyright (C) 2022 Ethan Krug
00009   */
00010  package project;
00011
00012  import mt.MTRandom;
00013
00014  public class Particle {
00015      // The current position of the particle
00016      public double[] solution;
00017
00018      // The current velocity of the particle
00019      public double velocity;
00020
00021      // The fitness of the particle
00022      public double fitness;
00023
00024      // The best position the particle has ever achieved
00025      public Particle pBest;
00026
00034      public Particle(double[] solution, double range, int problem) {
00035          this.solution = solution;
00036          this.fitness = new Problem(solution, problem).getFitness();
00037          this.pBest = null;
00038          MTRandom r = new MTRandom();
00039          // Upper bound is [range] and lower bound is [-range], so 50% of
00040          // [U - L] = [range]
00041          this.velocity = r.nextDouble() * range;
00042      }
00043
00049      public void setPBest(Particle pBest) {
00050          this.pBest = pBest;
00051      }
00052  }

```

## 8.11 project/Population.java File Reference

### Classes

- class [project.Population](#)

### Packages

- package [project](#)

## 8.12 Population.java

[Go to the documentation of this file.](#)

```

00001  /*
00002   * Population : Population object implemented as a control structure to assist
00003   *             in running the optimization algorithms.
00004   * Author    : Ethan Krug
00005   * Email     : ethan.c.krug@gmail.com
00006   * Date      : May, 2022
00007   *
00008   * Copyright (C) 2022 Ethan Krug
00009   */
00010  package project;
00011

```



```

00012 import mt.MTRandom;
00013
00014 public class Population {
00015
00016     // Holds the generated values for each experiment
00017     private double[][] population;
00018
00019     // Holds the fitness values of each experiment
00020     private double[] fitness;
00021
00022     // Holds the solution vectors
00023     private double[][] solutions;
00024
00025     // Range of possible values
00026     private double range;
00027
00035     public Population(int n, int m, double range) {
00036         this.range = range;
00037         fitness = new double[n];
00038         solutions = new double[n][m];
00039         population = genRandomMatrix(n, m);
00040     }
00041
00047     public double[] getFitness() {
00048         return fitness;
00049     }
00050
00057     public void setFitness(int i, double fitness) {
00058         this.fitness[i] = fitness;
00059     }
00060
00067     public void setSolution(int i, double[] sol) {
00068         solutions[i] = sol;
00069     }
00070
00076     public double[][] getPopulation() {
00077         return population;
00078     }
00079
00085     public double[][] getSolution() {
00086         return solutions;
00087     }
00088
00094     public void setPopulation(double[][] population) {
00095         this.population = population;
00096     }
00097
00103     public double getRange() {
00104         return range;
00105     }
00106
00114     public double[][] genRandomMatrix(int n, int m) {
00115         double[][] matrix = new double[n][m];
00116         MTRandom r = new MTRandom(false);
00117         for (int i = 0; i < n; i++) {
00118             for (int j = 0; j < m; j++) {
00119                 matrix[i][j] = r.nextDouble() * (range - (-range)) + (-range);
00120             }
00121         }
00122         return matrix;
00123     }
00124
00131     public double[] genRandomArray(int m) {
00132         double[] array = new double[m];
00133         MTRandom r = new MTRandom(false);
00134         for (int i = 0; i < m; i++) {
00135             array[i] = r.nextDouble() * (range - (-range)) + (-range);
00136         }
00137         return array;
00138     }
00139
00148     public double[][] genNeighborhood(int n, int m, double[] solution) {
00149         double[][] neighborhood = new double[n][m];
00150         MTRandom r = new MTRandom(false);
00151
00152         // Fill a neighborhood with pseudo-random values based off of the solution
00153         // vector within a given range
00154         for (int i = 0; i < n; i++) {
00155             for (int j = 0; j < m; j++) {
00156                 double value = solution[j] + r.nextDouble() * (range - (-range)) + (-range);
00157                 if (value > range) {
00158                     value = range;
00159                 } else if (value < -range) {
00160                     value = -range;
00161                 }
00162
00163                 neighborhood[i][j] = value;

```

```

00164         }
00165     }
00166     return neighborhood;
00167 }
00168 }

```

## 8.13 project/Problem.java File Reference

### Classes

- class [project.Problem](#)

### Packages

- package [project](#)

## 8.14 Problem.java

[Go to the documentation of this file.](#)

```

00001 /*
00002  * Problem : A Java implementation of ten mathematical functions typically
00003  *           used in testing optimization algorithms.
00004  * Author   : Ethan Krug
00005  * Email    : ethan.c.krug@gmail.com
00006  * Date     : May, 2022
00007  *
00008  * Copyright (C) 2022 Ethan Krug
00009  */
00010 package project;
00011
00012 public class Problem {
00013     // Holds the generated values passed to the constructor
00014     private double[] values;
00015
00016     // Holds the fitness value of values based on the problem type
00017     private double fitness;
00018
00025     public Problem(double[] vector, int probNum) {
00026         values = vector;
00027         switch (probNum) {
00028             case 1:
00029                 fitness = schwefel();
00030                 break;
00031             case 2:
00032                 fitness = de_jong_1();
00033                 break;
00034             case 3:
00035                 fitness = rosenbrock();
00036                 break;
00037             case 4:
00038                 fitness = rastrigin();
00039                 break;
00040             case 5:
00041                 fitness = griewank();
00042                 break;
00043             case 6:
00044                 fitness = sine_envelope();
00045                 break;
00046             case 7:
00047                 fitness = sine_V();
00048                 break;
00049             case 8:
00050                 fitness = ackley_one();
00051                 break;
00052             case 9:
00053                 fitness = ackley_two();
00054                 break;
00055             case 10:
00056                 fitness = egg_holder();
00057                 break;

```

```

00058     }
00059 }
00060
00066 public double getFitness() {
00067     return fitness;
00068 }
00069
00077 public double nthRoot(int root, double value) {
00078     return Math.pow(value, 1.0 / root);
00079 }
00080
00087 public double square(double value) {
00088     return value * value;
00089 }
00090
00096 public double schwefel() {
00097     double sum = 0;
00098     for (int i = 0; i < values.length; i++) {
00099         sum += -values[i] * Math.sin(Math.sqrt(Math.abs(values[i])));
00100     }
00101     return 418.9829 * values.length - sum;
00102 }
00103
00109 public double de_jong_1() {
00110     double sum = 0;
00111     for (int i = 0; i < values.length; i++) {
00112         sum += square(values[i]);
00113     }
00114     return sum;
00115 }
00116
00122 public double rosenbrock() {
00123     double sum = 0;
00124     for (int i = 0; i < values.length - 1; i++) {
00125         sum += 100 * square((square(values[i]) - values[i + 1])) + square((1 - values[i]));
00126     }
00127     return sum;
00128 }
00129
00135 public double rastrigin() {
00136     double sum = 0;
00137     for (int i = 0; i < values.length; i++) {
00138         sum += square(values[i]) - 10 * Math.cos(2 * Math.PI * values[i]);
00139     }
00140     return 10 * values.length + sum;
00141 }
00142
00148 public double griewank() {
00149     double sum = 0;
00150     double prod = 1;
00151     for (int i = 0; i < values.length; i++) {
00152         sum += square(values[i]);
00153         prod *= Math.cos(values[i] / Math.sqrt(i + 1));
00154     }
00155     return sum / 4000 - prod + 1;
00156 }
00157
00164 public double sine_envelope() {
00165     double sum = 0;
00166     for (int i = 0; i < values.length - 1; i++) {
00167         double top = square(Math.sin(square(values[i]) + square(values[i + 1]) - 0.5));
00168         double bottom = square(1 + 0.001 * (square(values[i]) + square(values[i + 1])));
00169         sum += 0.5 + top / bottom;
00170     }
00171     return -sum;
00172 }
00173
00179 public double sine_V() {
00180     double sum = 0;
00181     for (int i = 0; i < values.length - 1; i++) {
00182         double first = nthRoot(4, square(values[i]) + square(values[i + 1]));
00183         double second = square(Math.sin(50 * nthRoot(10, square(values[i]) + square(values[i +
00184 1]))));
00185         sum += first * second + 1;
00186     }
00187     return sum;
00188 }
00194 public double ackley_one() {
00195     double sum = 0;
00196     for (int i = 0; i < values.length - 1; i++) {
00197         double first = Math.pow(Math.E, -0.2) * Math.sqrt(square(values[i]) + square(values[i +
00198 1]));
00199         double second = 3 * (Math.cos(2 * values[i]) + Math.sin(2 * values[i + 1]));
00200         sum += first + second;
00201     }
00202     return sum;

```

```
00202     }
00203
00209     public double ackley_two() {
00210         double sum = 0;
00211         for (int i = 0; i < values.length - 1; i++) {
00212             double first = Math.pow(Math.E,
00213                 0.2 * Math.sqrt((square(values[i]) + square(values[i + 1])) / 2));
00214             double second = Math.pow(Math.E,
00215                 0.5 * (Math.cos(2 * Math.PI * values[i]) + Math.cos(2 * Math.PI * values[i +
00216 1]))));
00216             sum += 20 + Math.E - (20 / first) - second;
00217         }
00218         return sum;
00219     }
00220
00226     public double egg_holder() {
00227         double sum = 0;
00228         for (int i = 0; i < values.length - 1; i++) {
00229             double first = -values[i] * Math.sin(Math.sqrt(Math.abs(values[i] - values[i + 1] - 47)));
00230             double second = (values[i + 1] + 47) * Math.sin(Math.sqrt(Math.abs(values[i + 1] + 47 +
00231 values[i] / 2)));
00231             sum += first - second;
00232         }
00233         return sum;
00234     }
00235
00236 }
```

## 8.15 README.md File Reference

# Index

- ackley\_one
  - project.Problem, [41](#)
- ackley\_two
  - project.Problem, [41](#)
- Algorithms
  - project.Algorithms, [15](#)
- blindSearch
  - project.Algorithms, [16](#)
- DE
  - project.Algorithms, [16](#)
- de\_jong\_1
  - project.Problem, [41](#)
- egg\_holder
  - project.Problem, [41](#)
- experiment
  - Main, [19](#)
- fileLoop
  - Main, [20](#)
- fitness
  - project.Particle, [35](#)
- FSS
  - project.NEH, [31](#)
- FSSB
  - project.NEH, [31](#)
- genNeighborhood
  - project.Population, [36](#)
- genRandomArray
  - project.Population, [37](#)
- genRandomMatrix
  - project.Population, [37](#)
- getFitness
  - project.Algorithms, [17](#)
  - project.Population, [38](#)
  - project.Problem, [42](#)
- getPopulation
  - project.Population, [38](#)
- getRange
  - project.Population, [38](#)
- getSolution
  - project.Algorithms, [17](#)
  - project.Population, [38](#)
- griewank
  - project.Problem, [42](#)
- localSearch
  - project.Algorithms, [17](#)
- m
  - project.NEH, [32](#)
- Main, [19](#)
  - experiment, [19](#)
  - fileLoop, [20](#)
  - main, [20](#)
  - makeOutFile, [20](#)
  - minimizeDriver, [21](#)
  - readFromFile, [21](#)
  - readLines, [22](#)
  - scheduleDriver, [22](#)
  - scheduleLoop, [22](#)
  - writeFile, [23](#)
  - writeResult, [23](#)
- main
  - Main, [20](#)
- Main.java, [45](#)
- makeOutFile
  - Main, [20](#)
- makespan
  - project.NEH, [33](#)
- matrix
  - project.NEH, [33](#)
- minimizeDriver
  - Main, [21](#)
- mt, [13](#)
- mt.MTRandom, [24](#)
  - MTRandom, [25, 26](#)
  - next, [27](#)
  - pack, [28](#)
  - setSeed, [28, 29](#)
- mt/MTRandom.java, [48](#)
- MTRandom
  - mt.MTRandom, [25, 26](#)
- n
  - project.NEH, [33](#)
- NEH
  - project.NEH, [30](#)
- next
  - mt.MTRandom, [27](#)
- nthRoot
  - project.Problem, [42](#)
- pack
  - mt.MTRandom, [28](#)
- Particle
  - project.Particle, [34](#)
- pBest
  - project.Particle, [35](#)

- Population
  - project.Population, 36
- Problem
  - project.Problem, 40
- project, 13
- project.Algorithms, 15
  - Algorithms, 15
  - blindSearch, 16
  - DE, 16
  - getFitness, 17
  - getSolution, 17
  - localSearch, 17
  - PSO, 18
  - repeatedLocalSearch, 18
- project.NEH, 30
  - FSS, 31
  - FSSB, 31
  - m, 32
  - makespan, 33
  - matrix, 33
  - n, 33
  - NEH, 30
  - runNEH, 32
  - schedule, 33
  - transposeMatrix, 32
- project.Particle, 33
  - fitness, 35
  - Particle, 34
  - pBest, 35
  - setPBest, 34
  - solution, 35
  - velocity, 35
- project.Population, 36
  - genNeighborhood, 36
  - genRandomArray, 37
  - genRandomMatrix, 37
  - getFitness, 38
  - getPopulation, 38
  - getRange, 38
  - getSolution, 38
  - Population, 36
  - setFitness, 39
  - setPopulation, 39
  - setSolution, 39
- project.Problem, 40
  - ackley\_one, 41
  - ackley\_two, 41
  - de\_jong\_1, 41
  - egg\_holder, 41
  - getFitness, 42
  - griewank, 42
  - nthRoot, 42
  - Problem, 40
  - rastrigin, 43
  - rosenbrock, 43
  - schwefel, 43
  - sine\_envelope, 43
  - sine\_V, 44
  - square, 44
  - project/Algorithms.java, 51
  - project/NEH.java, 55
  - project/Particle.java, 57, 58
  - project/Population.java, 58
  - project/Problem.java, 60
  - PSO
    - project.Algorithms, 18
  - rastrigin
    - project.Problem, 43
  - readFromFile
    - Main, 21
  - readLines
    - Main, 22
  - README.md, 62
  - repeatedLocalSearch
    - project.Algorithms, 18
  - rosenbrock
    - project.Problem, 43
  - runNEH
    - project.NEH, 32
  - schedule
    - project.NEH, 33
  - scheduleDriver
    - Main, 22
  - scheduleLoop
    - Main, 22
  - schwefel
    - project.Problem, 43
  - setFitness
    - project.Population, 39
  - setPBest
    - project.Particle, 34
  - setPopulation
    - project.Population, 39
  - setSeed
    - mt.MTRandom, 28, 29
  - setSolution
    - project.Population, 39
  - sine\_envelope
    - project.Problem, 43
  - sine\_V
    - project.Problem, 44
  - solution
    - project.Particle, 35
  - square
    - project.Problem, 44
  - transposeMatrix
    - project.NEH, 32
  - velocity
    - project.Particle, 35
  - writeFile
    - Main, 23
  - writeResult
    - Main, 23