My Project

Generated by Doxygen 1.8.15

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 optimazation.pkg1.CreateCSV Class Reference

CreateCSV (p. 5) class.

Public Member Functions

- void **CreateCsv** (double[ ][ ] a, int rows, String b)
- void **CreateCsv0** (double[ ][ ] a, int colums, String C) throws IOException
- void **CreateCsv1** (double[ ][ ]a, String C) throws IOException

3.1.1 Detailed Description

CreateCSV (p. 5) class.

Author

JuneYeob Lee(2462 9603)

-

This class is for creating CSV file about results

Date

4/5/2019 !\Contact:Leej @cwu.edu !\Created on: 3/28/2019

3.1.2 Member Function Documentation

3.1.2.1 CreateCsv()

```
void optimazation.pkg1.CreateCSV.CreateCsv (
             double a[ ][ ],
             int rows,
             String b )
```

CreateCsv method this method will create csv file and calculate datas for csv file

Parameters

a	
rows	
b	

message if cannot find file

3.1.2.2 CreateCsv0()

```
void optimazation.pkg1.CreateCSV.CreateCsv0 (
             double a[][],
             int colums,
             String C ) throws IOException
```

This method for creating the csv file of the best solution

Parameters

a	(a for the best solution);
colums	(the number of colums)
C	

Exceptions

IOException	

3.1.2.3 CreateCsv1()

```
void optimazation.pkg1.CreateCSV.CreateCsv1 (
             doublea [][],
             String C ) throws IOException
```

Parameters

a(the	best solutions 2d array length should be 18 size should be the number of iteration)
C	(File name)

Exceptions

IOException	

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

- CreateCSV.java

3.2 optimazation.pkg1.CreateMatrix Class Reference

CreateMatrix (p. 7) class.

Public Member Functions

- **CreateMatrix** ()

  A constructor(default)
- **CreateMatrix** (int a, int b)

  A constructor.
- void **create** (int length, int size)

  this method will create input 2d arrays with all 0.0
- void **add** (double min, double max, String a) throws FileNotFoundException, UnsupportedEncodingException

3.2.1 Detailed Description

CreateMatrix (p. 7) class.

Author

JuneYeob Lee(24629603)

-

Date

4/5/2019 \Contact:Leej @cwu.edu \Created on: 3/28/2019

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CreateMatrix()

```
optimazation.pkg1.CreateMatrix.CreateMatrix (
             int a,
             int b )
```

A constructor.

Parameters

int	a the first argument(rows)
int	b the second argument(colums)

3.2.3 Member Function Documentation

3.2.3.1 add()

```
void optimazation.pkg1.CreateMatrix.add (
             double min,
             double max,
             String a ) throws FileNotFoundException, UnsupportedEncodingException
```

A add method !this method will fill out input with random numbers(this method will create 7 files which

```
/*!@param double min for first parameter( minimum range for random number)
  !@param double max for second parameter(Maximum range for random number)
```

3.2.3.2 create()

```
void optimazation.pkg1.CreateMatrix.create (
             int length,
             int size )
```

this method will create input 2d arrays with all 0.0

A create method

Parameters

int	length for first parameter(length) !
int	size for second parameter(size)

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

- CreateMatrix.java

3.3 optimazation.pkg1.Functioning Class Reference

Functioning (p. 8) class.

Public Member Functions

- double [ ][ ] **funct** (int length, int size) throws FileNotFoundException, UnsupportedEncodingException
- void **bestVector** (int functionnumber, int solutioncolum, int size) throws FileNotFoundException

3.3.1 Detailed Description

Functioning (p. 8) class.

Author

JuneYeob Lee(24629603) \This **Functioning** (p. 8) class has made to implemente all 18 functions by ranges.

-

Date

4/5/2019 Contact:Leej @cwu.edu Created on: 3/28/2019

3.3.2 Member Function Documentation

3.3.2.1 bestVector()

```
void optimazation.pkg1.Functioning.bestVector (
             int functionnumber,
             int solutioncolum,
             int size ) throws FileNotFoundException
```

/∗ BestVector method is for finding the best vector of the best solution. for the function number search different input file because each function has different range

Parameters

functionnumber	(number of function number 0∼17
solutioncolum	(colum number of result 2d array)
size	(size of input matrix)

Exceptions

FileNotFoundException	

3.3.2.2 funct()

```
double [][] optimazation.pkg1.Functioning.funct (
             int length,
             int size ) throws FileNotFoundException, UnsupportedEncodingException
```

CreateMatrix (p. 7) object has created

Functions (p. 10) object has created result 2d arrays

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

- Functioning.java

3.4 optimazation.pkg1.Functions Class Reference

Public Member Functions

- void **Schwefel** (double[ ][ ] a)
- void **DeJong** (double[ ][ ] a)
- void **Rosenbrock** (double[ ][ ] a)
- void **Rastrigin** (double[ ][ ] a)
- void **Griewangk** (double[ ][ ] a)
- void **SineEnvelope** (double[ ][ ] a)
- void **StretchedV** (double[ ][ ] a)
- void **Ackley1** (double[ ][ ] a)
- void **Ackely2** (double[ ][ ] a)
- void **EggHolder** (double[ ][ ] a)
- void **Rana** (double[ ][ ] a)
- void **Pathological** (double[ ][ ] a)
- void **Michalewicz** (double[ ][ ] a)
- void **Masters** (double[ ][ ] a)
- void **Quartic** (double[ ][ ] a)
- void **Levy** (double[ ][ ] a)
- void **Step** (double[ ][ ] a)
- void **Alpine** (double[ ][ ] a)

3.4.1 Detailed Description

Author

JuneYeob Lee(24629603) **Functions** (p. 10) class is composed with 18 functions

-

Date

4/5/2019 Contact:Leej @cwu.edu Created on: 3/28/2019

3.4.2 Member Function Documentation

3.4.2.1 Ackely2()

```
void optimazation.pkg1.Functions.Ackely2 (
            double a[ ][ ] )
```

Ackley Two function Range[-32,32] expected global minimum = 0

Parameters

a	(input matrix)

3.4.2.2 Ackley1()

```
void optimazation.pkg1.Functions.Ackley1 (
             double a[][] )
```

Ackley One Function

- Range[-32,32] expected global minimum = -7.54276-2.91867(n-3)

Parameters

a	(input matrix)

3.4.2.3 Alpine()

```
void optimazation.pkg1.Functions.Alpine (
             double a[][] )
```

Alpine function Range[-100,100] expected global minimum = 0

Parameters

a	(input matrix)

3.4.2.4 DeJong()

```
void optimazation.pkg1.Functions.DeJong (
             double a[][] )
```

DeJong 1 function Range [-100,100] expected minimum =0

Parameters

a	(input matrix)

3.4.2.5 EggHolder()

```
void optimazation.pkg1.Functions.EggHolder (
            double a[][] )
```

EggHolder Range[-500,500] expected global minimum none

Parameters

a	(input matrix)

3.4.2.6 Griewangk()

```
void optimazation.pkg1.Functions.Griewangk (
            double a[][] )
```

Griewangk function Range[-500,500] expected global minimum =0

Parameters

a	(input matrix)

3.4.2.7 Levy()

```
void optimazation.pkg1.Functions.Levy (
            double a[][] )
```

Levy function Range[-10,10] expected global minimum = 0

Parameters

a	(input matrix)

3.4.2.8 Masters()

```
void optimazation.pkg1.Functions.Masters (
            double a[][] )
```

Masters'Cosine Wave function Range[-30,30] expected global minimum = 1-n when n is size of input matrix

Parameters

a	(input matrix)

3.4.2.9 Michalewicz()

```
void optimazation.pkg1.Functions.Michalewicz (
             double a[][] )
```

Michalewicz function Range[0,Pi] expected global minimum = 0.996n when n is size of input matrix

Parameters

a	(input matrix)

3.4.2.10 Pathological()

```
void optimazation.pkg1.Functions.Pathological (
             double a[][] )
```

Pathological function Range[-100,100] expected global minimum = none

Parameters

a	(input matrix)

3.4.2.11 Quartic()

```
void optimazation.pkg1.Functions.Quartic (
             double a[][] )
```

Quartic function Range[-100,100] expected global minimum = 0

Parameters

a	(input matrix)

3.4.2.12 Rana()

```
void optimazation.pkg1.Functions.Rana (
             double a[][] )
```

Rana function Range[-500,500] expected global minimum =none

Parameters

a	(input matrix)

3.4.2.13 Rastrigin()

```
void optimazation.pkg1.Functions.Rastrigin (
             double a[][] )
```

Rastrigin function Range[-30,30] expectedglobal minimum =0

Parameters

a	(input matrix)

3.4.2.14 Rosenbrock()

```
void optimazation.pkg1.Functions.Rosenbrock (
             double a[][] )
```

Rosenbrock's Saddle function Range[-100,100] expected global minimum =0

Parameters

a	(input matrix)

3.4.2.15 Schwefel()

```
void optimazation.pkg1.Functions.Schwefel (
             double a[][] )
```

Schwefel method input range[-512,512] expect global minimum = 0

Parameters

a	(input matrix)

3.4.2.16 SineEnvelope()

```
void optimazation.pkg1.Functions.SineEnvelope (
             double a[][] )
```

Sine Envelope Sine Wave Range[-30,30] expected global minimum =-1.4915(n-1) when n is size of input matrix

Parameters

a	(input matrix)

3.4.2.17 Step()

```
void optimazation.pkg1.Functions.Step (
             double a[][] )
```

Step function Range[-100,100] expected global minimum = 0

Parameters

a	(input matrix)

3.4.2.18 StretchedV()

```
void optimazation.pkg1.Functions.StretchedV (
             double a[][] )
```

StretchedV Range[-30,30] expected global minimum = 0

Parameters

a	(input matrix)

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

- Functions.java

3.5 optimazation.pkg1.IterativeLocalSearch Class Reference

Public Member Functions

- double [ ][ ] **IterativeLocalSearch** (**LocalSearch** l, int itr, int length, int size, **Functioning** f) throws File↩ NotFoundException, UnsupportedEncodingException, IOException

3.5.1 Detailed Description

Author

JuneYeob Lee(24629603) IterativLocalSearch is that find best solution of local search with iteration. This class should deal with FileNot FoundException, IOException and Unsupported Encoding Exception

3.5.2 Constructor & Destructor Documentation

3.5.2.1 IterativeLocalSearch()

```
double [][] optimazation.pkg1.IterativeLocalSearch.IterativeLocalSearch (
              LocalSearch l,
             int itr,
             int length,
             int size,
              Functioning f ) throws FileNotFoundException, UnsupportedEncodingException, IO↩
Exception
```

IterativeLocal search method to find best solution of the x times iterated local search

Parameters

l	(Local search object)
itr	(number of iteration
length	(number of input rows)
size	(number of input colums)
f	(functioning object)

Returns

Best solution for Local Search

Exceptions

FileNotFoundException	
UnsupportedEncodingException	
IOException	

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

- IterativeLocalSearch.java

3.6 optimazation.pkg1.LocalSearch Class Reference

Public Member Functions

- double [ ][ ] **LocalSearch** (int length, int size, **Functioning** f) throws FileNotFoundException, Unsupported↩ EncodingException, IOException

3.6.1 Detailed Description

Author

JuneYeob Lee Local Search Algorithm class This class should deal with FileNot FoundException, IOException and Unsupported Encoding Exception

3.6.2 Constructor & Destructor Documentation

3.6.2.1 LocalSearch()

```
double [][] optimazation.pkg1.LocalSearch.LocalSearch (
              int length,
              int size,
               Functioning f ) throws FileNotFoundException, UnsupportedEncodingException, IO↩
Exception
```

This method is made for local search algorithm. First set the initial solutions for 18 functions but it's not best because it's picked randomly this method will find the local minimum solution or global minimum.(in the neighbor area.) it will iterate until the standing point is smaller that left or right.

Parameters

length	(number of input rows)
size	(number of input colums)
f	(functioning object)

Returns

Best solution for Local Search

Exceptions

FileNotFoundException	

Exceptions

UnsupportedEncodingException	
IOException	

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

- LocalSearch.java

3.7 optimazation.pkg1.Main Class Reference

Optimization1 class.

Static Public Member Functions

- static void **main** (String[ ] args) throws FileNotFoundException, IOException
  main method

3.7.1 Detailed Description

Optimization1 class.

3.7.2 Member Function Documentation

3.7.2.1 main()

```
static void optimazation.pkg1.Main.main (
             String [] args ) throws FileNotFoundException, IOException  [static]
```

main method

Parameters

args	the command line arguments

Exceptions

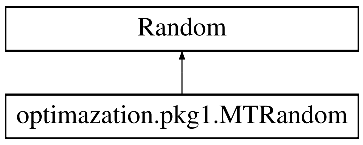
java.io.FileNotFoundException	
java.io.UnsupportedEncodingException	

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

- Main.java

3.8 optimazation.pkg1.MTRandom Class Reference

Inheritance diagram for optimazation.pkg1.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)

3.8.1 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()** (p. 21) for generating bits in the pseudo random number sequence. Anyone using this class should invoke the public inherited methods (next↩ Int(), 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`.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 MTRandom() [1/5]

```
optimazation.pkg1.MTRandom.MTRandom ( )
```

The default constructor for an instance of **MTRandom** (p. 19). 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.current↩ TimeMillis().

3.8.2.2 MTRandom() [2/5]

```
optimazation.pkg1.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

compatible	Compatibility flag for replicating original behaviour.

3.8.2.3 MTRandom() [3/5]

```
optimazation.pkg1.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

seed	The seed value with which to initialise this class.

3.8.2.4 MTRandom() [4/5]

```
optimazation.pkg1.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

buf	The non-empty byte array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

3.8.2.5 MTRandom() [5/5]

```
optimazation.pkg1.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

buf	The non-empty integer array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

3.8.3 Member Function Documentation

3.8.3.1 next()

```
final synchronized int optimazation.pkg1.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

bits	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.

3.8.3.2 pack()

```
static int [] optimazation.pkg1.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 }
```

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

buf	The non-null byte array to be packed.

Returns

A non-null integer array of the packed bytes.

Exceptions

NullPointerException	if the given byte array is null.

3.8.3.3 setSeed() [1/3]

```
final synchronized void optimazation.pkg1.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** (p. 19) (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

seed	The 64 bit value used to initialise the random number generator state.

3.8.3.4 setSeed() [2/3]

```
final void optimazation.pkg1.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

buf	The non-empty byte array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

3.8.3.5 setSeed() [3/3]

```
final synchronized void optimazation.pkg1.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

buf	The non-empty integer array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

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

- MTRandom.java

3.9 optimazation.pkg1.RandomWalk Class Reference

Public Member Functions

- double [ ][ ] **randomWalk** (int itr, int length, int size, **Functioning** f) throws FileNotFoundException, UnsupportedEncodingException, IOException

3.9.1 Detailed Description

Author

JuneYeob Lee **RandomWalk** (p. 24) calss which generated the best fitness for 18 solutions. randomly pick the solution, iterate 30 times for 18 functions.

3.9.2 Member Function Documentation

3.9.2.1 randomWalk()

```
double [ ][ ] optimazation.pkg1.RandomWalk.randomWalk (
             int itr,
             int length,
             int size,
              Functioning f ) throws FileNotFoundException, UnsupportedEncodingException, IO↩
Exception
```

This method is used for blind search. minumum iterate 30 times each time it generate random integer from Mersenne Twister.

Parameters

itr	iteration time.
length	numbers of input vectors(at least 30).
size	size of input it should be 10 or 20 or 30.
f	functioning object which has funct method to get the result of 18 functions.

Returns

This method return the bestfitness for 18 functions at least 30 times iterations.

Exceptions

FileNotFoundException	
UnsupportedEncodingException	
IOException	

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

- RandomWalk.java

Index