MDEA

0.1

Generated by Doxygen 1.8.9.1

Mon Jun 20 2016 09:31:59

Contents

Chapter 1

Génération de modèles diversifiés à l'aide d'algorithme génétiques

Ce projet est un projet de recherche réalisé au Laboratoire d'Informatique, de Robotique et Microélectronique de Montpellier (UMR CNRS et Université de Montpellier). Un résumé est donné ci-dessous.

Résumé/Abstract

Français

La génération de modèles à partir de méta-modèles est une solution apportée par l'ingénierie des modèles pour tester les transformations de modèles ou pour la validation de méta-modèles. Il existe dans la littérature un certain nombre de techniques combinatoires pour l'instanciation de méta-modèle, mais peu se sont intéressées au problème de la diversité. L'utilisation des algorithmes génétiques est une des plus prometteuses en terme de diversité. Durant nos travaux, nous avons expérimentés deux approches, basées sur l'algorithme évolutionnaire NSGA-II, pour la génération de nouveaux modèles, plus diversifiés. Les résultats ainsi obtenus mettre en évidence une réelle augmentation de la distance entre modèles, permettant de fournir des bancs d'essais plus réalistes et plus complets que ceux initiaux.

English

Generation of models that conform to a meta-model is one of the solutions from model driven engineering for models transformations testing or meta-models validation. Some combinatorial techniques have been explored in previous works, but few have focused on problem of diversity. Amongst previous works, genetic algorithms usage seems to be one of the most promising for diversity generation. In this work, we experiment two approach, based on the evolutionnary algorithm NSGA-II, to generate more diverse models. These approaches provide results with an increased distance, enabling creation of more realistic and complete benchmark.

Spécification

L'objectif de cette application est de prendre en entrée un certain nombre de modèles préexistants et d'en générer des nouveaux, plus diversifiés.

Syntaxe : "‘ mdea -in <models directory>=""> [-cx <inter|intra>] [-dist <cosine|hamming|centrality>] [-fitness <min|avg|minavg|minavgs|dist>] [-g <number of="" generations>="">] [-m <percentage of="" mutation="" chance>="">] [-nb <number of="" model="" in="" a="" single="" chromosom>="">] [-out <output directory>="">]
"‘

Format des fichiers

Les fichiers de définition de modèles doivent être définis en suivant la syntaxe suivante : "‘ Vecteur (entiers séparés par des espaces) Emplacement du fichier XCSP Emplacement du fichier .grimm Emplacement du méta-modèle Racine du méta-modèle "‘

Exemple : "‘ 1 2 2 2 3 3 3 1 1 1 1 1 1 1 1 1 1 1 2 34 34 34 34 1 2 34 34 34 34 1 2 34 34 34 34 1 2 34 34 34 1 2 34 34 34 1 1 11 3 1 1 11 3 1 1 11 3 1 1 11 3 1 1 12 3 1 1 11 3 1 1 11 3 1 1 11 3 1 1 12 3 1 1 12 3 1 21 31 1 22 32 1 23 33 1 24 34 1 25 34 1 11 1 12 1 13 javasmall/c0.xml ProjectSmall.grimm MyJava.ecore Project "‘

Chapter 2

Deprecated List

Member Model::setDomains (DomainsPtr d)

Parameters

v	The new value of Model

Member Model::setVal (GenesPtr v)

Parameters

v	The new value of Model

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Class Documentation

6.1 Chromosom Class Reference

Class that represent a genome in our problem.

```
#include <Chromosom.h>
```

Inheritance diagram for Chromosom:

Public Member Functions

- Chromosom ()
- Chromosom (const std::vector< std::string > &files)
- virtual ~Chromosom () noexcept
- virtual const std::vector< Model > & getModels () const
- virtual void setModels (std::vector< Model > models)
- virtual int mutate (float pmut)
- virtual bool isValid () const

Static Public Member Functions

- static size_t setNbModels (size_t)
- static size_t getNbModels ()

Public Attributes

- int **front**

Protected Attributes

- std::vector< Model > **models**

Static Protected Attributes

- static size_t **nbModels** = 2

6.1.1 Detailed Description

Class that represent a genome in our problem.

This class is a genome representation of our problem.

Author

Florian Galinier

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Chromosom::Chromosom ( )

Create a new empty chromosom.

6.1.2.2 Chromosom::Chromosom (const std::vector< std::string > & *files*)

Create a new chromosom with data from input files.

Parameters

file	The file path that contains vector value for chromosom.

6.1.2.3 Chromosom::~Chromosom ( ) [virtual], [noexcept]

Destructor for Chromosom.

6.1.3 Member Function Documentation

6.1.3.1 const std::vector< Model > & Chromosom::getModels ( ) const [virtual]

Return vector of models

Returns

The vector of models

6.1.3.2 size_t Chromosom::getNbModels ( ) [static]

Change actual value of Chromosom

Parameters

v	The new value of Chromosom

6.1.3.3 bool Chromosom::isValid ( ) const [virtual]

Return true if the chromosom own only valid models.

Returns

true iff the chromosom is valid.

6.1.3.4 int Chromosom::mutate (float *pmut*) [virtual]

Mutate models with a chance of pmut percent.

Parameters

pmut	The percentage chance of mutation

Returns

The number of mutation

6.1.3.5 void Chromosom::setModels (std::vector< Model > *models*) [virtual]

Change actual value of Chromosom

Parameters

models	The new value of Chromosom

Reimplemented in GAChromosom.

6.1.3.6 size_t Chromosom::setNbModels (size_t *nb*) [static]

Change actual value of Chromosom

Parameters

v	The new value of Chromosom

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

- model/Chromosom.h
- model/Chromosom.cpp

6.2 CommandLine Class Reference

Class for command line options parsing.

`#include <CommandLine.h>`

Public Member Functions

- CommandLine (std::string app)
- void addOption (std::string option, std::string help, bool required=true)
- void addOption (std::pair< std::string, std::string > option, bool required=true)
- void printUsage () const
- std::map< std::string, std::string > *parse (int argc, char **argv) const

6.2.1 Detailed Description

Class for command line options parsing.

This class allow usage of command line interface by parsing options.

Author

Florian Galinier

6.2.2 Constructor & Destructor Documentation

6.2.2.1 CommandLine::CommandLine (std::string *app*)

Constructor for new CommandLine parser.

Parameters

app	The application name.

6.2.3 Member Function Documentation

6.2.3.1 void CommandLine::addOption (std::string *option*, std::string *help*, bool *required* = `true`)

Add an option to parse.

Parameters

option	The option value.
help	The help to display with this option.
required	Specify if option is optional or required (true for required and by default)

6.2.3.2 void CommandLine::addOption (std::pair< std::string, std::string > *option*, bool *required* = `true`)

Add option method (see addOption(std::string,std::string,bool)) with std::pair.

Parameters

option	std::pair composed of option value and help to display.
required	Specify if option is optional or required (true for required and by default)

6.2.3.3 std::map< std::string, std::string > CommandLine::parse (int *argc*, char ** *argv*) const

Parse command line options.

Parameters

argc	The main argc argument.
argv	The main argv argument.

Returns

Options passed in command.

6.2.3.4 void CommandLine::printUsage ( ) const

Print usage method. This method will exit program.

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

- utils/CommandLine.h
- utils/CommandLine.cpp

6.3 Directory Class Reference

This class provide methods for directory manipulation.

`#include <Directory.h>`

Public Member Functions

- Directory (std::string path)
- virtual ~Directory ()
- virtual size_t getNbFiles (std::string ext="") const
- virtual std::shared_ptr< std::vector< std::string > > getFiles (std::string ext="") const

6.3.1 Detailed Description

This class provide methods for directory manipulation.

This class provide methods for directory manipulation.

Author

Florian Galinier

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Directory::Directory (std::string *path*)

Default constructor with directory path.

Parameters

path	The directory path

6.3.2.2 Directory::~Directory ( ) [virtual]

Directory destructor

6.3.3 Member Function Documentation

6.3.3.1 std::shared_ptr< std::vector< std::string > > Directory::getFiles (std::string *ext* = "") const [virtual]

Return a vector of all files in directory.

Parameters

ext	Optional file extension.

Returns

A vector with files path.

6.3.3.2 size_t Directory::getNbFiles (std::string *ext* = "") const [virtual]

Return the number of files in the directory.

Parameters

ext	Optional file extension.

Returns

The number of files in directory.

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

- utils/Directory.h
- utils/Directory.cpp

6.4 GAChromosom Class Reference

Class that implement a genome for our problem.

`#include <GAChromosom.h>`

Inheritance diagram for GAChromosom:

Collaboration diagram for GAChromosom:

Public Types

- enum **Method** { **COSINE**, **HAMMING**, **CENTRALITY** }
- enum **Cross** { **INTRA**, **INTER** }

Public Member Functions

- **GADefineIdentity** ("GAChromosom", 201)
- GAChromosom ()
- GAChromosom (const std::vector< std::string > &files)
- GAChromosom (const GAChromosom &g)
- virtual ~GAChromosom () noexcept
- virtual MatrixPtr score (MatrixPtr m)
- virtual MatrixPtr score () const
- virtual GAGenome *clone (CloneMethod method) const
- virtual void copy (const GAGenome &g)
- virtual void setModels (std::vector< Model > models)
- std::string to_string () const
- virtual int write (std::ostream &os) const
- virtual MatrixPtr evaluate (GABoolean gaFlag=gaFalse)

Static Public Member Functions

- static float compare (const GAGenome &g1, const GAGenome &g2)
- static void init (GAGenome &g)
- static int **onePointIntraCrossover** (const GAGenome &dad, const GAGenome &mom, GAGenome *sis, GAGenome *bro)
- static int onePointCrossover (const GAGenome &dad, const GAGenome &mom, GAGenome *sis, GA←-Genome *bro)
- static float avgEvaluator (GAGenome &g)
- static MatrixPtr evaluate (GAGenome &g)
- static int mutate (GAGenome &g, float pmut)

Static Public Attributes

- static enum GAChromosom::Method **method** = GAChromosom::Method::COSINE
- static enum GAChromosom::Cross **crossover** = GAChromosom::Cross::INTER

Additional Inherited Members

6.4.1 Detailed Description

Class that implement a genome for our problem.

This class implement the GAGenome interface of GAlib API and is a modelisation of our problem.

Author

Florian Galinier

6.4.2 Constructor & Destructor Documentation

6.4.2.1 GAChromosom::GAChromosom ( )

Default constructor that create a new genome.

6.4.2.2 GAChromosom::GAChromosom (const std::vector< std::string > & *files*)

Create a new genome from files.

Parameters

path	The vector of files path

6.4.2.3 GAChromosom::GAChromosom (const GAChromosom & *g*)

Create genome that is a copy of parameter

Parameters

g	The chromosom to copy

6.4.2.4 GAChromosom::~GAChromosom ( ) [virtual], [noexcept]

GAChromosom destructor

6.4.3 Member Function Documentation

6.4.3.1 float GAChromosom::avgEvaluator (GAGenome & *g*) [static]

Give an average fitness value for the given GAChromosom (greater is better)

Parameters

g	The GAChromosom to evaluate

Returns

The fitness value of the GAChromosom

6.4.3.2 GAGenome∗ GAChromosom::clone (CloneMethod *method*) const [virtual]

Return a new GAGenome that is a copy of actual one.

Parameters

method	Clone method to use for copy (optional)

Returns

A clone of actual GAChromosom

6.4.3.3 float GAChromosom::compare (const GAGenome & *g1*, const GAGenome & *g2*) [static]

Function that compare two GAChromosom and return a distance value.

Parameters

g1	First GAGenome to compare
g2	Second GAGenome to compare

Returns

A float that is equal to 0 if g1 and g2 are the same, -1 if an error occurred and another float greater than 0 that is a degree of difference if g1 and g2 can be compared.

6.4.3.4 void GAChromosom::copy (const GAGenome & *g*) [virtual]

Change actual GAChromosom by copying g values.

Parameters

g	GAGenome to copy

6.4.3.5 MatrixPtr GAChromosom::evaluate (GABoolean *gaFlag* = gaFalse) [virtual]

Give a fitness value of the actual GAChromosom

Parameters

gaFlag	Force or not the evaluation

Returns

The fitness value of the GAChromosom

6.4.3.6 MatrixPtr GAChromosom::evaluate (GAGenome & *g*) [static]

Give a matrix of distance value for the given GAChromosom (greater is better)

Parameters

g	The GAChromosom to evaluate

Returns

The fitness value of the GAChromosom

6.4.3.7 void GAChromosom::init (GAGenome & *g*) [static]

Initialize a new GAChromosom.

Parameters

g	The GAChromosom to initialize

6.4.3.8 int GAChromosom::mutate (GAGenome & *g*, float *pmut*) [static]

Mutate the given GAChromosom.

Parameters

g	The GAChromosom to mutate
pmut	The percentage chance of mutation

Returns

The number of mutation

6.4.3.9 int GAChromosom::onePointCrossover (const GAGenome & *dad*, const GAGenome & *mom*, GAGenome ∗ *sis*, GAGenome ∗ *bro*) [static]

Cross two GAChromosom and return two new one by tranlocation.

Parameters

dad	First GAChromosom to cross
mom	Second GAChoromosom to cross
sis	The first offspring crossed chromosom
bro	The second offspring crossed chromosom

Returns

Number of new chromosom obtain by crossover

6.4.3.10 MatrixPtr GAChromosom::score (MatrixPtr *m*) [virtual]

Set the matrix score.

Parameters

m	The new matrix score.

Returns

The new matrix score.

6.4.3.11 MatrixPtr GAChromosom::score ( ) const [virtual]

Return the matrix score (if matrix is null, score is not yet set).

Returns

The matrix score.

6.4.3.12 void GAChromosom::setModels (std::vector< Model > *models*) [virtual]

Change actual value of GAChromosom

Parameters

models	The new value of GAChromosom

Reimplemented from Chromosom.

6.4.3.13 std::string GAChromosom::to_string ( ) const

Convert chromosom to a string representation.

Returns

A string representation of current chromosom.

6.4.3.14 int GAChromosom::write (std::ostream & *os*) const [virtual]

Print GAChromosom value in param stream.

Parameters

os	The stream where to write

Returns

Undefined

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

- model/GAChromosom.h
- model/GAChromosom.cpp

6.5 Interval< T > Class Template Reference

An interval representation.

```
#include <Interval.h>
```

Public Member Functions

- Interval (const T &min, const T &max)
- Interval (const std::string &bounds)
- virtual ~Interval ()
- virtual const T & lbound () const
- virtual const T & ubound () const
- virtual bool include (const T &a) const
- virtual std::string to_string () const

6.5.1 Detailed Description

template<typename T>class Interval< T >

An interval representation.

This class is an interval representation. Every class that implement comparison operator can be used as template type.

Author

Florian Galinier

6.5.2 Constructor & Destructor Documentation

6.5.2.1 template<typename T> Interval< T >::Interval (const T & *min*, const T & *max*) [inline]

Constructor for a new interval.

Parameters

min	Lower bound of interval
max	Upper bound of interval

6.5.2.2 template<typename T> Interval< T >::Interval (const std::string & *bounds*) [inline]

Parse the string and construct new interval. String must be of type N or N..M

Parameters

bounds	The string to parse

6.5.2.3 template<typename T> virtual Interval< T >::~Interval ( ) [inline], [virtual]

Destructor for an Interval

6.5.3 Member Function Documentation

6.5.3.1 template<typename T> virtual bool Interval< T >::include (const T & *a*) const [inline], [virtual]

Test if a value is included in the interval.

Parameters

a	The value to test.

Returns

true if a is included in the interval; false elsewhere.

6.5.3.2 template<typename T> virtual const T& Interval< T >::lbound ( ) const [inline], [virtual]

Return lower bound of interval;

Returns

Lower bound of the interval.

6.5.3.3 template<typename T> virtual std::string Interval< T >::to_string () const [inline], [virtual]

Convert an interval vector to a string.

Returns

A string representation of the interval

6.5.3.4 template<typename T> virtual const T& Interval< T >::ubound () const [inline], [virtual]

Return upper bound bound of interval;

Returns

Upper bound of the interval.

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

- utils/Interval.h

6.6 IntervalVector< T > Class Template Reference

A set of disjoint intervals.

```
#include <IntervalVector.h>
```

Public Member Functions

- IntervalVector ()
- IntervalVector (const IntervalVector &iv)
- virtual ~IntervalVector ()
- virtual const T & lbound () const
- virtual const T & ubound () const
- virtual bool include (const T &a) const
- virtual int add (const Interval< T > &i)
- virtual int add (std::string bounds)
- virtual int add (const T &min, const T &max)
- virtual std::string to_string () const

6.6.1 Detailed Description

template<typename T> class IntervalVector< T >

A set of disjoint intervals.

This class provide a set of interval.

Author

Florian Galinier

6.6.2 Constructor & Destructor Documentation

6.6.2.1 template<typename T > IntervalVector< T >::IntervalVector ( ) [inline]

Constructor for a new interval vector.

6.6.2.2 template<typename T > IntervalVector< T >::IntervalVector (const IntervalVector< T > & *iv*) [inline]

Constructor for a new interval vector.

6.6.2.3 template<typename T > virtual IntervalVector< T >::~IntervalVector ( ) [inline], [virtual]

Destructor for an IntervalVector

6.6.3 Member Function Documentation

6.6.3.1 template<typename T > virtual int IntervalVector< T >::add (const Interval< T > & *i*) [inline], [virtual]

Add an interval to the vector

Parameters

i	The new interval to add

Returns

1 if the interval intersect another existing one (there will be merged), 2 if the interval is already covered by an existing one, 0 elsewhere.

6.6.3.2 template<typename T > virtual int IntervalVector< T >::add (std::string *bounds*) [inline], [virtual]

Add an interval to the vector

Parameters

bounds	The string to parse

Returns

1 if the interval intersect another existing one (there will be merged), 0 elsewhere.

6.6.3.3 template<typename T > virtual int IntervalVector< T >::add (const T & *min*, const T & *max*) [inline], [virtual]

Add an interval to the vector

Parameters

min	Lower bound of new interval
max	Upper bound of new interval

Returns

1 if the interval intersect another existing one (there will be merged), 0 elsewhere.

6.6.3.4 template<typename T > virtual bool IntervalVector< T >::include (const T & *a*) const [inline], [virtual]

Test if a value is included in one of the intervals.

Parameters

a	The value to test.

Returns

true if a is included in the interval vector; false elsewhere.

6.6.3.5 template<typename T > virtual const T& IntervalVector< T >::lbound () const [inline], [virtual]

Return lower bound of all intervals;

Returns

Lower bound of the interval vector.

6.6.3.6 template<typename T > virtual std::string IntervalVector< T >::to_string () const [inline], [virtual]

Convert an interval to a string.

Returns

A string representation of the interval vector

6.6.3.7 template<typename T > virtual const T& IntervalVector< T >::ubound () const [inline], [virtual]

Return upper bound bound of all intervals;

Returns

Upper bound of the interval vector.

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

- utils/IntervalVector.h

6.7 Logger Class Reference

Class that allow to log information in a file.

`#include <Logger.h>`

Public Member Functions

- Logger (std::string fileName, std::ios_base::openmode mode=std::ios_base::out)
- virtual ~Logger () noexcept
- virtual void write (std::string s)
- template<typename T >
  Logger & operator<< (const T &t)
- virtual Logger & operator<< (const std::string &s)
- virtual Logger & operator<< (const char *c)

6.7.1 Detailed Description

Class that allow to log information in a file.

This class provide an interface to write informations in a file, like population state or rejected mutation rate.

Author

Florian Galinier

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Logger::Logger (std::string *fileName*, std::ios_base::openmode *mode* = `std::ios_base::out`)

Create a new log file

Parameters

fileName	name of the file to create
mode	The mode to use to open

6.7.2.2 Logger::~Logger ( ) `[virtual]`, `[noexcept]`

Destructor of Logger

6.7.3 Member Function Documentation

6.7.3.1 template<typename T > Logger& Logger::operator<< (const T & *t*) `[inline]`

Overloaded operator to use log as a stream

Parameters

t	the object to write

Returns

A reference to current Logger

6.7.3.2 Logger & Logger::operator<< (const std::string & *s*) `[virtual]`

Specialization of operator<<

Parameters

s	the string to write

Returns

A reference to current Logger

6.7.3.3 Logger & Logger::operator<< (const char * *c*) `[virtual]`

Specialization of operator<<

Parameters

c	the string to write

Returns

A reference to current Logger

6.7.3.4 void Logger::write (std::string *s*) [virtual]

Write the given string in the log file

Parameters

s	The string to write

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

- utils/Logger.h
- utils/Logger.cpp

6.8 Matrix Class Reference

Class that implement a square matrix.

`#include <Matrix.h>`

Public Member Functions

- Matrix (size_t t)
- virtual ~Matrix () noexcept
- virtual bool operator! () const
- virtual bool isPositive () const
- Matrix ∗operator- (const Matrix &m2)
- virtual void set (size_t i, size_t j, double v)
- virtual bool operator>= (const Matrix &m2)
- virtual bool operator> (const Matrix &m2)
- const std::vector< double > & operator[ ] (size_t i) const
- virtual double average () const
- virtual double lineAverage (int line) const
- virtual double min () const
- virtual std::string to_csv_string () const
- virtual std::string to_string () const
- size_t size () const

6.8.1 Detailed Description

Class that implement a square matrix.

This class implement is used for store results of multi-dimensionnal score.

Author

Florian Galinier

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Matrix::Matrix(size_t t)

Create a new square matrix of size t.

Parameters

t	The number of rows and columns.

6.8.2.2 Matrix::~Matrix ( ) [virtual], [noexcept]

Destructor for Matrix class.

6.8.3 Member Function Documentation

6.8.3.1 double Matrix::average ( ) const [virtual]

Return an average value of upper triangular matrix.

Returns

Average value of upper triangular matrix.

6.8.3.2 bool Matrix::isPositive ( ) const [virtual]

Return true if the matrix is positive, i.e. if all the matrix elements are greater or equal to 0.

Returns

true if matrix is positive.

6.8.3.3 double Matrix::lineAverage (int *line*) const [virtual]

Return the average of a line.

Parameters

line	The line where to compute average.

Returns

The average of the line.

6.8.3.4 double Matrix::min ( ) const [virtual]

Return the minimum of upper triangular matrix.

Returns

Minimum of upper triangular matrix.

6.8.3.5 bool Matrix::operator! ( ) const [virtual]

Return true if and only if the matrix is null.

Returns

true if matrix is null, false elsewhere.

6.8.3.6 Matrix ∗Matrix::operator(const Matrix & *m2*)

Substract two matrices and return the resulting matrix.

Parameters

m2	The matrix to substract to current one.

Returns

A new pointer Matrix that is the result of substraction.

6.8.3.7 bool Matrix::operator> (const Matrix & *m2*) [virtual]

Compare two matrices and return true only if current matrix is greater than m2.

Parameters

m2	The matrix to compare.

Returns

true only if current matrix is greater than m2.

6.8.3.8 bool Matrix::operator>= (const Matrix & *m2*) [virtual]

Compare two matrices and return true only if current matrix is greater or equal than m2.

Parameters

m2	The matrix to compare.

Returns

true only if current matrix is greater or equal than m2.

6.8.3.9 const std::vector< double > & Matrix::operator[] (size_t *i*) const

Accessor for line of Matrix. Allow to access to matrix element by using [][].

Parameters

i	The line to access.

Returns

The ith line.

6.8.3.10 void Matrix::set (size_t *i*, size_t *j*, double *v*) [inline], [virtual]

Change element at position (i,j).

Parameters

i	The number of the row.

j	The number of the column.
v	The new value of Matrix at (i,j).

6.8.3.11 size_t Matrix::size ( ) const

Return the size of matrix (height and width, both are equal).

Returns

The size of matrix.

6.8.3.12 std::string Matrix::to_csv_string ( ) const `[virtual]`

Return a CSV string representation of matrix.

Returns

CSV string representation of matrix.

6.8.3.13 std::string Matrix::to_string ( ) const `[virtual]`

Return a string representation of matrix.

Returns

string representation of matrix.

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

- model/Matrix.h
- model/Matrix.cpp

6.9 Model Class Reference

Class that represent a model.

```
#include <Model.h>
```

Public Member Functions

- Model ()
- Model (const Model &m)
- Model (std::string genesFile)
- virtual ~Model () noexcept
- virtual void setVal (GenesPtr v)
- virtual GenesPtr getVal () const
- virtual void setDomains (DomainsPtr d)
- virtual DomainsPtr getDomains () const
- virtual bool isValid () const
- virtual bool operator== (const Model &b)
- virtual bool operator!= (const Model &b)
- virtual std::string to_string () const
- virtual std::string generateDotFile (int i)

Static Public Member Functions

- static float evaluateExtern (Model &m1, Model &m2)
- static float evaluate (const Model &m1, const Model &m2)

Protected Attributes

- GenesPtr **genes**
- DomainsPtr **domains**
- std::string **xcsp**
- std::string **mm**
- std::string **root**
- std::string **grimm**
- std::string **dot**
- bool **change**

6.9.1 Detailed Description

Class that represent a model.

This class is a gene vector representation of a model. Due to the fact that a model can be composed by more than a model, we create a class that encapsulate the idea of model.

Author

Florian Galinier

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Model::Model ( )

Create a new empty Model.

6.9.2.2 Model::Model (const Model & *m*)

Create a Model that is a copy of Model m.

Parameters

m	The Model to copy.

6.9.2.3 Model::Model (std::string *genesFile*)

Create a new model with data from input file.

Parameters

genesFile	The file path that contains vector value for model.

6.9.2.4 Model::~Model ( ) `[virtual]`, `[noexcept]`

Model destructor.

6.9.3 Member Function Documentation

6.9.3.1 float Model::evaluate (const Model & *m1*, const Model & *m2*) [static]

Take two models and return an evaluation of the distance between them.

Parameters

m1	First model to test.
m2	Second model to test.

Returns

The distance between the two models.

6.9.3.2 float Model::evaluateExternal (Model & *m1*, Model & *m2*) [static]

Take two models and return an evaluation of the distance between them.

Parameters

m1	First model to test.
m2	Second model to test.

Returns

The distance between the two models.

6.9.3.3 std::string Model::generateDotFile ( ) [virtual]

Generate a dot file from actual model.

Returns

The dot file path.

6.9.3.4 DomainsPtr Model::getDomains ( ) const [virtual]

Return domains of genes in the Model

Returns

The variable domains of Model

6.9.3.5 GenesPtr Model::getVal ( ) const [virtual]

Return actual value of Model

Returns

The current value of Model

6.9.3.6 bool Model::isValid ( ) const [virtual]

Check if the Model is a valid model or not.

Returns

true if Model is valid, elsewhere false.

6.9.3.7 bool Model::operator!= (const Model & *b*) [virtual]

Compare two models and return true if models are differents.

Parameters

b	Other model to compare

Returns

true if this and b are not equal, false elsewhere.

6.9.3.8 bool Model::operator== (const Model & *b*) [virtual]

Compare two models and return true if models are equals.

Parameters

b	Other model to compare

Returns

true if this and b are equal, false elsewhere.

6.9.3.9 void Model::setDomain (DomainsPtr *d*) [virtual]

Change actual value of Model

Deprecated

Parameters

v	The new value of Model

6.9.3.10 void Model::setValue (GenesPtr *v*) [virtual]

Change actual value of Model

Deprecated

Parameters

v	The new value of Model

6.9.3.11 std::string Model::to_string ( ) const [virtual]

Return a string that represent current model.

Returns

A string that represent current model.

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

- model/Model.h
- model/Model.cpp

6.10 NSGAII Class Reference

Implementation of NSGA-II algorithm.

`#include <NSGAII.h>`

Inheritance diagram for NSGAII:

Collaboration diagram for NSGAII:

Public Types

- enum Fitness { **MIN** = 0b0001, **AVG** = 0b0010, **DIST** = 0b0000, **MINAVG** = 0b0100 }

  Mask for fitness choice. Represent each fitness choice. Fitness can be used as mask with logical operator.

Public Member Functions

- **GADefineIdentity** ("NSGAII", 288)
- NSGAII (const GAPopulation &p, unsigned int pm=1)
- virtual ~NSGAII () noexcept
- virtual bool dominate (MatrixPtr m1, MatrixPtr m2)
- virtual std::shared_ptr< std::vector< std::vector< int > > >fastNonDominatedSort ()
- virtual std::map< size_t, double > crowdingDistanceAssignment (std::vector< int > &f)
- virtual void step ()
- virtual Statistics & **extraStatistics** ()
- NSGAII & operator++ ()

Static Public Attributes

- static int fitness = NSGAII::Fitness::AVG
- static unsigned int gen = 0

6.10.1 Detailed Description

Implementation of NSGA-II algorithm.

This class implement the GAGeneticAlgorithm interface of GAlib API in order to provide the NSGA-II algorithm.

Author

Florian Galinier

6.10.2 Member Enumeration Documentation

6.10.2.1 enum NSGAII::Fitness

Mask for fitness choice. Represent each fitness choice. Fitness can be used as mask with logical operator.

Author

Florian Galinier

6.10.3 Constructor & Destructor Documentation

6.10.3.1 NSGAII::NSGAII (const GAPopulation & *p*, unsigned int *pm* = 1)

Constructor for new NSGA-II Algorithm

Parameters

p	The initial population

6.10.3.2 NSGAII::~NSGAII ( ) [virtual], [noexcept]

Destructor

6.10.4 Member Function Documentation

6.10.4.1 std::map< size_t, double > NSGAII::crowdingDistanceAssignment (std::vector< int > & *f*) [virtual]

Assign to each individual of a front a crowding distance value.

Parameters

f	A vector of individual in a front

Returns

A map that associate at each individual a crowding distance.

6.10.4.2 bool NSGAII::dominate (MatrixPtr *m1*, MatrixPtr *m2*) [virtual]

Compare two matrices of score and return true if the first one dominate the second one.

Parameters

m1	The first matrix to test.
m2	The second one.

Returns

true iff m1 dominates m2.

6.10.4.3 std::shared_ptr< std::vector< std::vector< int > > > NSGAII::fastNonDominatedSort ( ) [virtual]

Sort population in Pareto front.

Returns

A vector of front composed by key of front's individual.

6.10.4.4 NSGAII & NSGAII::operator++ ( )

Operator that create a new generation.

Returns

The NSGA-II algorithm with a new population.

6.10.4.5 void NSGAII::step ( ) [virtual]

Create and evaluate a new generation.

6.10.5 Member Data Documentation

6.10.5.1 int NSGAII::fitness = NSGAII::Fitness::AVG [static]

Choosed fitness

6.10.5.2 unsigned int NSGAII::gen = 0 [static]

Current generation.

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

- model/NSGAII.h
- model/NSGAII.cpp

6.11 Population Class Reference

This class manage all chromosoms (especially the initial ones).

`#include <Population.h>`

Inheritance diagram for Population:

Collaboration diagram for Population:

Public Member Functions

- Population ()
- Population (std::string dirPath)
- MatrixPtr evaluate (GABoolean flag=gaFalse) const
- GAChromosom & best (size_t r, size_t c)
- GAChromosom & worst (size_t r, size_t c)

6.11.1 Detailed Description

This class manage all chromosoms (especially the initial ones).

This class provide methods for population management.

Author

Florian Galinier

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Population::Population ( )

Default constructor.

6.11.2.2 Population::Population (std::string *dirPath*)

Create a population from files given in directory.

Parameters

dirPath	The directory where the models files are.

6.11.3 Member Function Documentation

6.11.3.1 GAChromosom & Population::best (size_t *r*, size_t *c*)

Return the best element of population for the objectif in matrix score at m[r][c].

Parameters

r	Matrix row of the objective.
c	Matrix column of the objective

Returns

Best individual for objective score[r][c]

6.11.3.2 MatrixPtr Population::evaluate (GABoolean *flag* = `False`) const

Evaluate all individuals of population

Parameters

Force	or not the evaluation

6.11.3.3 GAChromosom & Population::worst (size_t *r*, size_t *c*)

Return the worst element of population for the objectif in matrix score at m[r][c].

Parameters

r	Matrix row of the objective.
c	Matrix column of the objective

Returns

Worst individual for objective score[r][c]

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

- model/Population.h
- model/Population.cpp

6.12 Statistics Class Reference

A class to store genetic algorithm stats.

`#include <Statistics.h>`

Public Member Functions

- void update (GAPopulation &p)
- const std::vector< float > & getMed () const
- const std::vector< float > & getFirstQuartile () const
- const std::vector< float > & getThirdQuartile () const
- const std::vector< float > & getMin () const
- const std::vector< float > & getMax () const

Static Public Attributes

- static std::string outfile = ""

6.12.1 Detailed Description

A class to store genetic algorithm stats.

This class provides utility to record statistics of the evolutionary algorithm.

Author

Florian Galinier

6.12.2 Member Function Documentation

6.12.2.1 const std::vector< float > & Statistics::getFirstQuartile ( ) const

Getter for first quartiles (each entry is a generation)

Returns

A vector with all recorded first quartiles.

6.12.2.2 const std::vector< float > & Statistics::getMax ( ) const

Getter for maximums (each entry is a generation)

Returns

A vector with all recorded maximums.

6.12.2.3 const std::vector< float > & Statistics::getMed ( ) const

Getter for medians (each entry is a generation)

Returns

A vector with all recorded medians.

6.12.2.4 const std::vector< float > & Statistics::getMin ( ) const

Getter for minimums (each entry is a generation)

Returns

A vector with all recorded minimums.

6.12.2.5 const std::vector< float > & Statistics::getThirdQuartile ( ) const

Getter for third quartiles (each entry is a generation)

Returns

A vector with all recorded third quartiles.

6.12.2.6 void Statistics::updateGA (Population & *p*)

Use population to update stats

Parameters

p	Population used for stats analysis.

6.12.3 Member Data Documentation

6.12.3.1 std::string Statistics::outfile [static]

The name of the output file

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

- model/Statistics.h
- model/Statistics.cpp

Chapter 7

File Documentation

7.1 model/Chromosom.h File Reference

Chromosom class header.

```
#include <vector>
#include <memory>
#include <string>
#include "Model.h"
```

Include dependency graph for Chromosom.h: This graph shows which files directly or indirectly include this file:

Classes

- class Chromosom

  Class that represent a genome in our problem.

7.1.1 Detailed Description

Chromosom class header.

Author

Florian Galinier

Version

0.1

Date

06/04/16

Definition of our problem that will be inherited for the genetic algorithm.

7.2 model/GAChromosom.h File Reference

GAChromosom class header.

```
#include "Chromosom.h"
#include <ga/GAGenome.h>
#include <functional>
#include "utils/Logger.h"
#include "Matrix.h"
```

Include dependency graph for GAChromosom.h: This graph shows which files directly or indirectly include this file:

Classes

- class GAChromosom

  Class that implement a genome for our problem.

Typedefs

- typedef std::shared_ptr< Matrix > MatrixPtr

Functions

- Logger & **operator<<** (Logger &log, const GAChromosom &c)

7.2.1 Detailed Description

GAChromosom class header.

Author

Florian Galinier

Version

0.1

Date

06/04/16

Declaration of our own genome class.

7.2.2 Typedef Documentation

7.2.2.1 typedef std::shared_ptr<Matrix> MatrixPtr

Smart pointer to a Matrix.

7.3 model/Matrix.h File Reference

Matrix class header.

```
#include <vector>
#include <iostream>
#include "utils/Logger.h"
```

Include dependency graph for Matrix.h: This graph shows which files directly or indirectly include this file:

Classes

- class Matrix

  Class that implement a square matrix.

Functions

- std::ostream & operator<< (std::ostream &os, const Matrix &m)
- Logger & operator<< (Logger &log, const Matrix &m)

7.3.1 Detailed Description

Matrix class header.

Author

Florian Galinier

Version

0.1

Date

27/04/16

Declaration of a square matrix class.

7.3.2 Function Documentation

7.3.2.1 std::ostream& operator<< (std::ostream & *os*, const Matrix & *m*)

operator<< overload for iostream.

Parameters

os	The ostream where to write.
m	The matrix to write.

Returns

The resulting ostream.

7.3.2.2 Logger& operator<< (Logger & *log*, const Matrix & *m*)

Specialization of Logger operator<< for Matrix

Parameters

log	The logger where to write
m	The matrix to write

Returns

The used logger

7.4 model/Model.h File Reference

Model class header.

```
#include <vector>
#include <memory>
#include <pugixml.hpp>
#include "utils/IntervalVector.h"
```

Include dependency graph for Model.h: This graph shows which files directly or indirectly include this file:

Classes

- class Model

  Class that represent a model.

Typedefs

- typedef std::vector< int > Genes
- typedef std::shared_ptr< Genes > GenesPtr
- typedef std::vector< IntervalVector< int > > Domains
- typedef std::shared_ptr< Domains > DomainsPtr

Functions

- std::ostream & operator<< (std::ostream &os, const Model &m)

7.4.1 Detailed Description

Model class header.

Author

Florian Galinier

Version

0.1

Date

06/04/16

The representation of a model in our problem is an int vector. Due to the fact that a chromosom can be composed by more than one model vector, we choose to reify it in a Model class.

7.4.2 Typedef Documentation

7.4.2.1 typedef std::vector<IntervalVector<int> >Domains

A vector of all domains. For all i in {0..Genes.size()}, Domains[i] = domains of Genes[i]

7.4.2.2 typedef std::shared_ptr<Domains> DomainsPtr

Smart pointer to a domains vector.

7.4.2.3 typedef std::vector<int> Genes

In our problem, a gene vector is a vector of int. We create an alias for convenience.

7.4.2.4 typedef std::shared_ptr<Genes> GenesPtr

Smart pointer to a genes vector.

7.4.3 Function Documentation

7.4.3.1 std::ostream& operator<<(std::ostream & *os*, const Model & *m*)

operator<< overload for iostream.

Parameters

os	The ostream where to write.
m	The model to write.

Returns

The resulting ostream.

7.5 model/NSGAII.h File Reference

NSGAII class header.

```
#include <ga/GASimpleGA.h>
#include <vector>
#include <map>
#include "Statistics.h"
```

Include dependency graph for NSGAII.h:

Classes

- class NSGAII

  Implementation of NSGA-II algorithm.

7.5.1 Detailed Description

NSGAII class header.

Author

Florian Galinier

Version

0.1

Date

15/04/16

NSGA-II class implementation for the evolutionary algorithm.

7.6 model/Population.h File Reference

Population class header.

```
#include "GAChromosom.h"
#include "utils/Logger.h"
#include <ga/GAPopulation.h>
```

Include dependency graph for Population.h: This graph shows which files directly or indirectly include this file:

Classes

- class Population

  This class manage all chromosoms (especially the initial ones).

Functions

- Logger & operator<< (Logger &log, const Population &p)

7.6.1 Detailed Description

Population class header.

Author

Florian Galinier

Version

0.1

Date

06/04/16

Population that will be used in our genetic algorithm.

7.6.2 Function Documentation

7.6.2.1 Logger& operator<< (Logger & *log*, const Population & *p*)

Specialization of Logger operator<< for Population

Parameters

log	The logger where to write
p	The population to write

Returns

The used logger

7.7 model/Statistics.h File Reference

Statistics class header.

```
#include "Population.h"
#include "utils/Logger.h"
```

Include dependency graph for Statistics.h: This graph shows which files directly or indirectly include this file:

Classes

- class Statistics

  A class to store genetic algorithm stats.

Functions

- Logger & operator<< (Logger &log, const Statistics &s)

7.7.1 Detailed Description

Statistics class header.

Author

Florian Galinier

Version

0.1

Date

03/05/16

This statistics class implementation for the evolutionary algorithm.

7.7.2 Function Documentation

7.7.2.1 Logger& operator<<(Logger & *log*, const Statistics & *s*)

Specialization of Logger operator<< for statistics

Parameters

log	The logger where to write
s	The statistics to write

Returns

The used logger

7.8 utils/CommandLine.h File Reference

CommandLine class header.

```
#include <map>
#include <string>
```

Include dependency graph for CommandLine.h:

Classes

- class CommandLine

  Class for command line options parsing.

7.8.1 Detailed Description

CommandLine class header.

Author

Florian Galinier

Version

0.1

Date

12/04/16

Set of functions for command line options.

7.9 utils/Directory.h File Reference

Directory class header.

```
#include <dirent.h>
#include <vector>
#include <string>
#include <memory>
```

Include dependency graph for Directory.h:

Classes

- class Directory

  This class provide methods for directory manipulation.

7.9.1 Detailed Description

Directory class header.

Author

Florian Galinier

Version

0.1

Date

06/04/16

Utils function for directory manipulation.

7.10 utils/Interval.h File Reference

Interval class header.

```
#include <stdexcept>
```

Include dependency graph for Interval.h: This graph shows which files directly or indirectly include this file:

Classes

- class Interval< T >

  An interval representation.

7.10.1 Detailed Description

Interval class header.

Author

Florian Galinier

Version

0.1

Date

11/04/16

Interval manipulation.

7.11 utils/IntervalVector.h File Reference

IntervalVector class header.

```
#include <vector>
#include "Interval.h"
```

Include dependency graph for IntervalVector.h: This graph shows which files directly or indirectly include this file:

Classes

- class IntervalVector< T >

  A set of disjoint intervals.

7.11.1 Detailed Description

IntervalVector class header.

Author

Florian Galinier

Version

0.1

Date

11/04/16

This class provide a vector of interval.

7.12 utils/Logger.h File Reference

Logger class header.

```
#include <string>
#include <fstream>
#include <ios>
```

Include dependency graph for Logger.h: This graph shows which files directly or indirectly include this file:

Classes

- class Logger

  Class that allow to log information in a file.

7.12.1 Detailed Description

Logger class header.

Author

Florian Galinier

Version

0.1

Date

29/04/16

A logger to write on a file.

Index