PBN Analyzer

Generated by Doxygen 1.8.6

Wed Apr 23 2014 16:47:02

Contents

1 PBN Analyzer 2.1

An application for Simulation/Inverse Decomposition of Probabilistic Boolean Networks.

1.1 Introduction

This is an open-source application that can be used in the research/analysis of PBN systems.

For users, you can inverse decompose your own transition matrix data and get the Entropy, Decomposition and Time data, or you can simulate random boolean matrices and weights for transition matrices.

For developer users, you can test your own algorithms using this API. Only the inverse algorithm part has to be implemented, all other functionalities are provided by the application itself.

1.2 Compilation

This application now comes with the source file with makefile and is currently a console application working on UNIX-like systems.

If you are on MacOS or Linux System, just unzip the "Analyzer.zip" file, then use the terminal to go into "Release" folder, type "make" to compile the binary. Then type "./Analyzer" to run the application.

If you are on Windows system, you should first download and install "cygwin" with "gcc, make" packages following the website After that, put the unzipped file in your home directory where your "cygwin" executable is. Then repeat the steps for MacOS system.

1.3 Mannual

When running the Analyzer application, first a menu will pop up and user are aked to choose a funcional module.

1. Simulate data : This module will simulate test data and save the data in directory "Input/Simulated". It is advised to input descriptive suffix when asked for.
2. Test provided data : This module will let the user test their own provided data. First make sure that your matrix data is in ".txt" format with columns separated by "Tabs" and rows separated by "newLines". The application has a "default mode" where it assumes you are testing "Provided data" rather than "Simulated data" and the algorithm used will be "OPTIMAL". User can configure the mode according to their own need.
3. and 4. are batch testing modules, on "Simulated data" and "Provided data" respectively. The test data are prepared and upon chosen, the module will directly start running.

1.4 Developer

If you want to use the application for testing your own inverse decomposition algorithm, you should follow the steps: Step 1 : Go to "Release/config.txt" and add the name of your algorithm in a new line before "_DEFAULT_"; Step 2 : Go to "Interactor.h" and add your algorithm name to the Enum "RandomTypes"; Step 3 : Here since the class responsible for doing inverse decomposition is the "Iterator" class, there are basically two ways to add a new algorithm. One is write a subclass of "Iterator" class and override the "Iterate" method to include your new algorithm. The second will be just modifying the original "Iterate" method. But for the second way you might need to check the implementation of "iterateOnce" and "chooseEntry" methods. Both method requires your implementation of the algorithm in as separate function.

1.5 Design

The major design used in this application is the "Model-View-Controller" design, where the "Interactor" serves as the "Abstract of Controller". The "View" part is abstracted in an interface called "Displayer". When moving to other platforms or building a GUI, the most important part to rewrite is the "Menu" which is a subclass of "Interactor" that implements the "Displayer" interface. Hence completing the task of user interaction.

The algorithms are encapsulated in the "Model" part consists of "Simulator" and "Iterator", where "Simulator" class is responsible for simulating transition matrix and "Iterator" class is responsible for doing inverse decomposition.

For further details, or usage of classes during development, please refer to the documentation of this application that follows.

2 Module Documentation

2.1 Basic Matrices

Classes

- class AbstractMatrix

  This class defines some of the basic operations we need for our matrix.
- class rMatrix

  This class holds the data needed for representing a boolean matrix.
- class tMatrix

  Internal data structure representing a matrix that has constant column sums.

2.1.1 Detailed Description

This group is the fundamental data structures used to save/load matrices and perform necessary computations.

2.2 User Interaction

Classes

- class Displayer::DisplayerClass

  Abstract interface for direct interation with end users.
- class Interactor

  An abstract controller class that contains method prototypes or basic implementations of methods for further overriding.
- class Iterator

  This class encapsulates the functionality of an inverse iterator.
- class Menu

  This is a concrete controller class for the application.
- class Simulator

  This is a class that handles simulation process.

2.2.1 Detailed Description

This group contains an interface for User Interaction and also a concrete class that implements the functionality. Upgrading or migrating to other platforms/languages need reimplementation of this class.

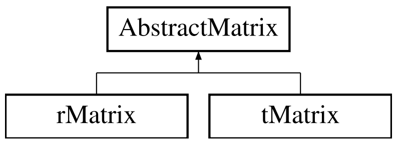
3 Class Documentation

3.1 AbstractMatrix Class Reference

This class defines some of the basic operations we need for our matrix.

`#include <AbstractMatrix.h>`

Inheritance diagram for AbstractMatrix:



Public Member Functions

- virtual int getSize () const
- virtual void print (ostream &output)=0

Protected Attributes

- int size

3.1.1 Detailed Description

This class defines some of the basic operations we need for our matrix.

This is the basic abstract square matrix class, and all Boolean Network matrix and transition matrix should inherit from this class.

3.1.2 Member Function Documentation

3.1.2.1 virtual int AbstractMatrix::getSize ( ) const `[virtual]`

Simple getter for the matrix, suppose the matrix is constructed with n genes, then the size is 2^n.

Returns

The number of rows for the matrix

3.1.2.2 virtual void AbstractMatrix::print (ostream & *output*) `[pure virtual]`

Print the matrix in a n by n matrix form to the output stream

Parameters

output	The stream for output

Implemented in tMatrix, and rMatrix.

3.1.3 Member Data Documentation

3.1.3.1 int AbstractMatrix::size `[protected]`

The size of the square matrix, inherited by its children.

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

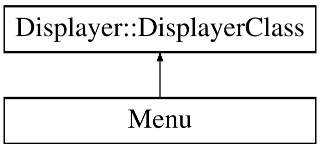
- AbstractMatrix.h

3.2 Displayer::DisplayerClass Class Reference

Abstract interface for direct interation with end users.

`#include <Displayer.h>`

Inheritance diagram for Displayer::DisplayerClass:



Public Member Functions

- virtual void showMenu ()=0
- virtual void getChoice ()=0

Protected Attributes

- char choice

  A character representing the choice of users.

Private Member Functions

- virtual void parseChoice ()=0
- virtual void displayMessage (string message)=0
- virtual void checkMatrix (tMatrix ∗transition)=0
- virtual void exitProgram ()=0

3.2.1 Detailed Description

Abstract interface for direct interation with end users.

This class declares function prototypes for user interactions. Any controller class should implement this interface.

3.2.2 Member Function Documentation

3.2.2.1 virtual void Displayer::DisplayerClass::checkMatrix (**tMatrix** ∗ *transition*) `[private], [pure virtual]`

Display the resulting matrix to end user for double-check.

Parameters

transition	A tMatrix instance to display

Implemented in Menu.

3.2.2.2 virtual void Displayer::DisplayerClass::displayMessage (string *message*) `[private], [pure virtual]`

Display a message to end user.

Parameters

message	The message to display.

Implemented in Menu.

3.2.2.3 virtual void Displayer::DisplayerClass::exitProgram ( ) `[private], [pure virtual]`

Action taken to exit the program. Specific behaviors when exiting could be included in this method.

Implemented in Menu.

3.2.2.4 virtual void Displayer::DisplayerClass::getChoice ( ) `[pure virtual]`

Get the choice from end users.

Implemented in Menu.

3.2.2.5 virtual void Displayer::DisplayerClass::parseChoice ( ) `[private], [pure virtual]`

A method to parse the choice from user. Deciding which mehod to call upon each choice.

Implemented in Menu.

3.2.2.6 virtual void Displayer::DisplayerClass::showMenu ( ) `[pure virtual]`

Displaying the menu containing choices to end users. This method should be implemented differently according to different purposes. i.e GUI or Console.

Implemented in Menu.

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

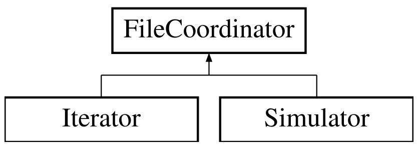
- Displayer.h

3.3 FileCoordinator Class Reference

Abstract class for file manipulations.

`#include <FileCoordinator.h>`

Inheritance diagram for FileCoordinator:



Public Types

- enum fileStates { READ_SUCCESS, READ_FAILURE }

  Specifies the states of the file being processed.
- enum saveTypes { MATRIX, ENTROPY, WEIGHTS, ITERATION }

  Specifies the types of data to be saved.

Public Member Functions

- string getFullPath ()
- string getFileName ()

- string getFailureMessage ()
- virtual fileStates readFile (string fileName, int numOfGenes)

Protected Attributes

- string FAILURE_MESSAGE

  Failure message expressing the cause of read file failure;.
- string FOLDER

  Folder of the file to process;.
- string FILEPREFIX

  Prefix of the file. i.e "sparseTransition_".
- string suffix

  Suffix of the file specified by user; i.e "newTest".
- string filename

  Name of the file; i.e "CEM", "overall".
- tMatrix $*$ transition

  The transition matrix currently being processed.

Private Member Functions

- virtual string saveMatrix ()
- virtual void saveVector (vector$<$ double $>$ &stats, saveTypes type)
- virtual void saveIteration ()

3.3.1 Detailed Description

Abstract class for file manipulations.

This is an abstract class that declared function prototypes used for file manipulations. Classes encapsulating algorithms should subclass from this class for file interaction and should implement the virtual methods as needed.

3.3.2 Member Enumeration Documentation

3.3.2.1 enum FileCoordinator::fileStates

Specifies the states of the file being processed.

The file being read could be read successfully or there might be an exception. When the state is "READ_FAILURE", a failure message will be generated for the user to correct.

Enumerator

READ_SUCCESS File successfully read in.

READ_FAILURE Error occurred when reading file.

3.3.2.2 enum FileCoordinator::saveTypes

Specifies the types of data to be saved.

The types of data to be saved will be used to determine which method to be used for saving process.

Enumerator

MATRIX Type MATRIX is used when a transition matrix is saved;.

ENTROPY Type ENTROPY is used when saving entropies of decomposition;.

WEIGHTS Type WEIGHTS is used when saving wegits of decomposition;.

ITERATION Type ITERATION is used when saving the iteration information containing minimal entropy result.

3.3.3 Member Function Documentation

3.3.3.1 string FileCoordinator::getFailureMessage ( )

Get the failure message if exceptions happen during file manipulation process.

Returns

The failure message.

3.3.3.2 string FileCoordinator::getFileName ( )

Get the filename for the file that is currently being processed or created. i.e If full path is "Input/Provided/CEM.txt", will return "CEM".

Returns

The name of the file(without extension) being processed.

3.3.3.3 string FileCoordinator::getFullPath ( )

Get the full path of the file that is currently being processed or created.

Returns

The full path of the file.

3.3.3.4 virtual **fileStates** FileCoordinator::readFile (string *fileName*, int *numOfGenes*) `[virtual]`

A function prototype declared for reading in a transition matrix from a certain file. Overriden by "Iterator" class.

Parameters

fileName	The name of the file, without extension and path;
numOfGenes	The number of genes corresponding to the matrix in the file;

Returns

A fileStates indicating whether the reading process is successful.

Reimplemented in Iterator.

3.3.3.5 virtual void FileCoordinator::saveIteration ( ) `[private]`, `[virtual]`

Saves iteration information into current file.

Reimplemented in Iterator.

3.3.3.6 virtual string FileCoordinator::saveMatrix ( ) `[private]`, `[virtual]`

Saves the transition matrix to current file.

Returns

Returns the full path of the file.

Reimplemented in Simulator.

3.3.3.7 virtual void FileCoordinator::saveVector (vector$<$ double $>$ & *stats*, **saveTypes** *type*) `[private]`, `[virtual]`

Saves a vector into the current file, depending on the parameters.

Parameters

stats	The vector data to save.
type	The saveType specifying the type of data. i.e ENTROPY, WEIGHTS

Reimplemented in Iterator.

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

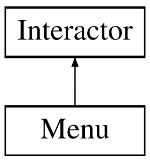
- FileCoordinator.h

3.4 Interactor Class Reference

An abstract controller class that contains method prototypes or basic implementations of methods for further overriding.

```
#include <Interactor.h>
```

Inheritance diagram for Interactor:



Protected Member Functions

- void initializeIterator ()
- void initializeSimulator ()
- void initializeIterator (string suffix, string fileName, int numOfGenes)
- void initializeSimulator (int numOfGenes, int simulateType, string suffix)
- string startIterator ()
- string startSimulator ()

Private Member Functions

- virtual void getSimulatorParams (int &numOfGenes, int &simulateType, string &suffix)=0
- virtual bool getIteratorParams (Iterator::IteratorStatus &status, Iterator::RandomTypes &type, string &file↩ Name, string &suffix, int &numOfGenes)=0
- virtual string getSuffix ()=0
- virtual void testAllOnSimulator ()=0
- virtual void testAllOnIterator (Iterator::IteratorStatus status)=0

3.4.1 Detailed Description

An abstract controller class that contains method prototypes or basic implementations of methods for further overriding.

3.4.2 Member Function Documentation

3.4.2.1 virtual bool Interactor::getIteratorParams (Iterator::IteratorStatus & *status,* Iterator::RandomTypes & *type,* string & *fileName,* string & *suffix,* int & *numOfGenes*) [private], [pure virtual]

Function prototype for getting Iterator parameters.

Parameters

status	Iterator status, SIMULATED or PROVIDED.

See Also

Iterator::IteratorStatus

Parameters

type	Iteration type.

See Also

Iterator::RandomTypes

Parameters

fileName	The input filename. i.e "overall" for "overall.txt"
suffix	File suffix added to output file
numOfGenes	Number of genes corresponding to the input file

Returns

True if the Iterator parameters are default values

Implemented in Menu.

3.4.2.2 virtual void Interactor::getSimulatorParams (int & *numOfGenes*, int & *simulateType*, string & *suffix*) `[private]`, `[pure virtual]`

Function prototype for getting Simulator parameters.

Parameters

numOfGenes	Number of genes to simulate
simulateType	Type of simulation, default is SPARSE
suffix	Suffix added to the end of output file

Implemented in Menu.

3.4.2.3 virtual string Interactor::getSuffix ( ) `[private]`, `[pure virtual]`

Prototype for the method used to get file suffix from user

Returns

The file suffix provided by the user

Implemented in Menu.

3.4.2.4 void Interactor::initializeIterator ( ) `[protected]`

Initialize the owned Iterator object.

3.4.2.5 void Interactor::initializeIterator (string *suffix*, string *fileName*, int *numOfGenes*) `[protected]`

Overloaded version, initialize the Iterator with specified params.

Parameters

suffix	File suffix added to the end of the file
fileName	The file name of input
numOfGenes	Number of genes corresponding to the input file

Note

This method is called by the InitializeIterator method internally.

3.4.2.6 void Interactor::initializeSimulator ( ) [protected]

Initialize the owned Simulator object.

3.4.2.7 void Interactor::initializeSimulator (int *numOfGenes*, int *simulateType*, string *suffix*) [protected]

Overloaded version, initialize the Simulator with specified params.

Parameters

numOfGenes	Number of genes the user wants to simulate
simulateType	The type of simulation, the default is SPARSE
suffix	The file suffix of the output file

Note

This method is called by the InitializeSimulator method internally.

3.4.2.8 string Interactor::startIterator ( ) [protected]

Starts the Iterator object, will be overridden in concrete implementation.

Returns

A message indicating the status of the Iterator

3.4.2.9 string Interactor::startSimulator ( ) [protected]

Starts the Simulator object, will be overridden in concrete implementation.

Returns

A message indicating the status of the Simulator

3.4.2.10 virtual void Interactor::testAllOnIterator (Iterator::IteratorStatus *status*) [private], [pure virtual]

Test all prepared iteration cases, either PROVIDED or SIMULATED

Parameters

status	Input file type. PROVIDED or SIMULATED.

Implemented in Menu.

3.4.2.11 virtual void Interactor::testAllOnSimulator ( ) [private], [pure virtual]

Test all prepared simulation cases.

Implemented in Menu.

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

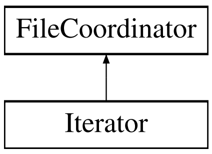
- Interactor.h
- Interactor.cpp

3.5 Iterator Class Reference

This class encapsulates the functionality of an inverse iterator.

`#include <Iterator.h>`

Inheritance diagram for Iterator:



Public Types

- enum IteratorStatus { SIMULATED, PROVIDED, ERROR, READY }

  The status of the Iterator, what kind of input file it is dealing with.
- enum RandomTypes {
  QUADRATIC, UNIFORM, CUBIC, MAXIMUM,
  OPTIMAL, DEFAULT }

  The algorithms used to conduct the inverse iteration.

Public Member Functions

- Iterator (IteratorStatus status=PROVIDED, RandomTypes types=OPTIMAL)
- void iterate ()
- bool sanityCheck ()
- virtual fileStates readFile (string fileName, int numOfGenes)

Static Public Attributes

- static vector< string > RANDOM_TYPE

Private Member Functions

- double iterateOnce ()
- double deterministicIterate ()
- double calculateEntropy (vector< double > &inputWeights)
- int chooseEntry (map< int, double > &column, RandomTypes type=QUADRATIC)
- virtual void saveVector (vector< double > &stats, saveTypes type)
- virtual void saveIteration ()

Additional Inherited Members

3.5.1 Detailed Description

This class encapsulates the functionality of an inverse iterator.

The Iterator inherits from FileCoordinator to use the file-related manipulations. The methods used for iteration are separated into two categories : Random and Deterministic. The currently used "Optimal" algorithm is a deterministic algorithm. The algorithm and input file type for iteration could be set when initializing the object, like :

```
Iterator* myIterator = new Iterator(IteratorStatus::PROVIDED, RandomTypes::OPTIMAL);
```

It can also be set later by invoking the "setType" method, like:

```
myIterator->setType(RandomTypes::MAXIMUM);
myIterator->setIteratorStatus(IteratorStatus::SIMULATED);
```

To let the Iterator object work, clients only need to call the public method "iterate" after configuration.

```
myIterator->iterate();
```

3.5.2 Member Enumeration Documentation

3.5.2.1 enum Iterator::IteratorStatus

The status of the Iterator, what kind of input file it is dealing with.

Enumerator

- ***SIMULATED*** Iterator will iterate on SIMULATED data.
- ***PROVIDED*** Iterator will iterate on PROVIDED real data.
- ***ERROR*** Iterator is in error status.
- ***READY*** Iterator is ready to start working.

3.5.2.2 enum Iterator::RandomTypes

The algorithms used to conduct the inverse iteration.

Although called "RandomTypes", this enumeration actually contains both random and deterministic algorithms. Moreover, clients can add self-developed algorithms into the enumeration.

Enumerator

- ***QUADRATIC*** A random algorithm, refer to the report for details.
- ***UNIFORM*** A random algorithm, refer to the report for details.
- ***CUBIC*** A random algorithm, refer to the report for details.
- ***MAXIMUM*** Deterministic algorithm, refer to the report for details.
- ***OPTIMAL*** Deterministic algorithm, refer to the report for details.
- ***DEFAULT*** Ending mark of the enumeration.

3.5.3 Constructor & Destructor Documentation

3.5.3.1 Iterator::Iterator (Iterator::IteratorStatus *status* = PROVIDED, Iterator::RandomTypes *type* = OPTIMAL)

Construct a new Iterator object, and set its IteratorStatus and RandomTypes. If no parameters provided, the default construction will use PROVIDED and OPTIMAL.

3.5.4 Member Function Documentation

3.5.4.1 double Iterator::calculateEntropy (vector< double > & *inputWeights*) `[private]`

Helper function that calculates entropy of a given output.

Parameters

inputWeights	A vector containing the weights of a specific iteration

Returns

The entropy of this iteration

3.5.4.2 int Iterator::chooseEntry (map< int, double > & *column*, RandomTypes *type* = QUADRATIC) [private]

Helper method to choose a non-zero entry in the iteration process. This method chooses entry based on Random or Deterministic methods. Clients are also required to override or create new method for entry-choosing when testing self-developed algorithms.

Parameters

column	The column of the transition matrix to choose entry from
type	The type of entry choosing

Returns

The row number of the chosen entry.

Note

This method is internally used by Iterator::iterateOnce and Iterator::deterministicIterate. General usage is like :

```
map<int, double>& column = copy.get(col);
int chosenRow = chooseEntry(column, type);
```

For detailed used, refer to Iterator.cpp

3.5.4.3 double Iterator::deterministicIterate ( ) [private]

Perform the iteration for deterministic algorithms.

Returns

The entropy of the iteration.

3.5.4.4 void Iterator::iterate ( )

Main method for iteration. This is a "Strategy Pattern" design. The iterate method will check the IteratorStatus and RandomType of the Iterator object and determine which method will be used on which file.\

It will also save the results in output files after iteration.

3.5.4.5 double Iterator::iterateOnce ( ) [private]

Perform one iteration for random algorithms.

Returns

The entropy of this iteration.

Note

For random algorithms, 1000 iterations will be performed and the iteration with least entropy will be selected. For implementation details, refer to Iterator.cpp.

3.5.4.6 Iterator::fileStates Iterator::readFile (string *fileName*, int *numOfGenes*) [virtual]

Read a transition matrix from input file.

Parameters

fileName	The file name of the file. i.e "overall" for "overall.txt"
numOfGenes	Number of genes corresponding to the input files

Returns

Whether the read-in process is successful.

See Also

FileCoordinator::fileStates

Reimplemented from FileCoordinator.

3.5.4.7 bool Iterator::sanityCheck ( )

A check of internal consistency of input by verifying that the column sums are ones.

Returns

True if the input file has no problem.

3.5.4.8 void Iterator::saveIteration ( ) `[private]`, `[virtual]`

Saves iteration information into current file.

Reimplemented from FileCoordinator.

3.5.4.9 void Iterator::saveVector (vector< double > & *stats*, FileCoordinator::saveTypes *type*) `[private]`, `[virtual]`

Saves a vector into the current file, depending on the parameters.

Parameters

stats	The vector data to save.
type	The saveType specifying the type of data. i.e ENTROPY, WEIGHTS

Reimplemented from FileCoordinator.

3.5.5 Member Data Documentation

3.5.5.1 vector< string > Iterator::RANDOM_TYPE `[static]`

A static container of names of iteration algorithms. This vector is initialized in Analyzer.cpp.

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

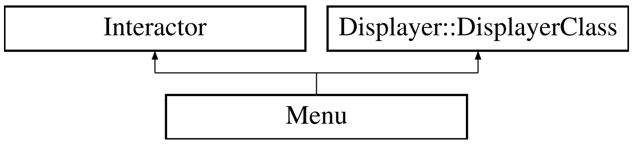
- Iterator.h
- Analyzer.cpp
- Iterator.cpp

3.6 Menu Class Reference

This is a concrete controller class for the application.

`#include <Menu.h>`

Inheritance diagram for Menu:



Public Member Functions

- virtual void showMenu ()
- virtual void getChoice ()

Private Member Functions

- virtual void displayMessage (string message)
- virtual void checkMatrix (tMatrix ∗transition)
- virtual void parseChoice ()
- virtual void exitProgram ()
- virtual string getSuffix ()
- virtual bool getIteratorParams (Iterator::IteratorStatus &status, Iterator::RandomTypes &type, string &file←Name, string &suffix, int &numOfGenes)
- virtual void getSimulatorParams (int &numOfGenes, int &simulateType, string &suffix)
- virtual void testAllOnSimulator ()
- virtual void testAllOnIterator (Iterator::IteratorStatus status)

Additional Inherited Members

3.6.1 Detailed Description

This is a concrete controller class for the application.

This class is a subclass of the "Interactor" that also implements the "Displayer" interface. This class serves as a controller that gets user choice using "Displayer" interface methods and process it using "Interactor" class methods.

3.6.2 Member Function Documentation

3.6.2.1 void Menu::checkMatrix (**tMatrix** ∗ *transition*) `[private]`, `[virtual]`

Prints the transition matrix to standard output and ask the user to ensure the correctness of the matrix.

Parameters

transition	A pointer reference to the matrix to be checked.

Implements Displayer::DisplayerClass.

3.6.2.2 virtual void Menu::displayMessage (string *message*) `[private]`, `[virtual]`

Specific implementation of abstract method in "Displayer" interface. This method prints a message to console output.

Parameters

message	The message to be shown.

Implements Displayer::DisplayerClass.

3.6.2.3 void Menu::exitProgram ( ) [private], [virtual]

Display a "Process Complete" message then directly exit the program.

Implements Displayer::DisplayerClass.

3.6.2.4 void Menu::getChoice ( ) [virtual]

Get the choice from the user by taking in input from c++ standard keyboard input.

Implements Displayer::DisplayerClass.

3.6.2.5 bool Menu::getIteratorParams (**Iterator::IteratorStatus** & *status*, **Iterator::RandomTypes** & *type*, string & *fileName*, string & *suffix*, int & *numOfGenes*) [private], [virtual]

Get the necessary parameters needed for initializing an "Iterator" instance. Information will be prompted to user from standard output.

Parameters

status	Status of the "Iterator", can be "PROVIDED" or "SIMULATED";
type	Type of algorithms used for decomposition;
fileName	The file name of the file storing the transition matrix;
suffix	Suffix obtained from user;
numOfGenes	The number of genes corresponding to the file with *fileName*

Returns

"True" if the "Iterator" is initialized in "Default" mode.

Note

This function is an implementation of a function prototype from "Interactor" class. Implementation should be different depending on different types of user interaction. This is how it is used :

```
if (getIteratorParams(status, type, fileName, suffix, numOfGenes)){
    myIterator = new Iterator();
}else{
     myIterator = new Iterator(status, type);
}
initializeIterator(suffix, fileName, numOfGenes);
```

Warning

This method must be implemented, or the initialization process cannot complete.

Implements Interactor.

3.6.2.6 void Menu::getSimulatorParams (int & *numOfGenes*, int & *simulateType*, string & *suffix*) [private], [virtual]

Function prototype for getting Simulator parameters.

Parameters

numOfGenes	Number of genes to simulate
simulateType	Type of simulation, default is SPARSE
suffix	Suffix added to the end of output file

Implements Interactor.

3.6.2.7 string Menu::getSuffix ( ) [private], [virtual]

Get the file suffix from user by prompting a message to standard output.

Returns

The suffix entered by the user.

Implements Interactor.

3.6.2.8 void Menu::parseChoice ( ) `[private]`, `[virtual]`

A method to parse the choice from user. Deciding which mehod to call upon each choice.

Implements Displayer::DisplayerClass.

3.6.2.9 void Menu::showMenu ( ) `[virtual]`

The method that shows the menu to user. This implementation will print the menu to the console output. The information of the menu is stored in the namespace "Displayer".

Implements Displayer::DisplayerClass.

3.6.2.10 void Menu::testAllOnIterator (**Iterator::IteratorStatus** ***status***) `[private]`, `[virtual]`

Test all prepared iteration cases, either PROVIDED or SIMULATED

Parameters

status	Input file type. PROVIDED or SIMULATED.

Implements Interactor.

3.6.2.11 void Menu::testAllOnSimulator ( ) `[private]`, `[virtual]`

Test all prepared simulation cases.

Implements Interactor.

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

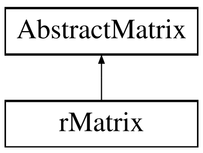
- Menu.h
- Menu.cpp

3.7 rMatrix Class Reference

This class holds the data needed for representing a boolean matrix.

`#include <rMatrix.h>`

Inheritance diagram for rMatrix:



Public Member Functions

- rMatrix (int n=1)
- rMatrix (vector< int > &positions)
- virtual void print (ostream &output)

Private Attributes

- vector< int > matrix

  A vector<int> storing the positions of ones in each column.

Additional Inherited Members

3.7.1 Detailed Description

This class holds the data needed for representing a boolean matrix.

The instances of this class can be initiated with number of genes, and a vector of rows. It also implements the print method.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 rMatrix::rMatrix (int *n* = 1)

Default constructor, creates an instance with random positions of ones.

3.7.2.2 rMatrix::rMatrix (vector< int > & *positions*)

This constructor takes in a vector<int> as an input and generate and instance of boolean matrix with the corresponding row positions being one.

3.7.3 Member Function Documentation

3.7.3.1 void rMatrix::print (ostream & *output*) [virtual]

Print the matrix in a n by n matrix form to the output stream

Parameters

output	The stream for output

Implements AbstractMatrix.

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

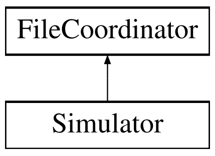
- rMatrix.h
- rMatrix.cpp

3.8 Simulator Class Reference

This is a class that handles simulation process.

`#include <Simulator.h>`

Inheritance diagram for Simulator:



Public Member Functions

- string simulate ()

Private Member Functions

- void generateTransition (int n)
- void generateTransition (int n, int m, ostream &output)
- virtual string saveMatrix ()

Additional Inherited Members

3.8.1 Detailed Description

This is a class that handles simulation process.

The Simulator class serves to simulate transition matrix with a given number of genes. There are two types of simulation: SPARSE and FORWARD. The SPARSE simulation will control the number of non-zero entries in each column below the number of genes.

The FORWARD simulation will simulate 10 BNs and their corresponding weights and construct a final transition matrix. But the result of this simulation is not guaranteed to be sparse, especially when the number of genes are small.

3.8.2 Member Function Documentation

3.8.2.1 void Simulator::generateTransition (int *n*) `[private]`

The method for generating SPARSE transition matrix.

Parameters

n	The number of genes to simulate

3.8.2.2 void Simulator::generateTransition (int *n*, int *m*, ostream & *output*) `[private]`

The method for generating FORWARD transition matrix.

Parameters

n	The number of genes to simulate
m	The number of BNs used in simulation. The default value is 10
output	An output stream to save the BNs and their weights

3.8.2.3 string Simulator::saveMatrix ( ) `[private]`, `[virtual]`

Saves the transition matrix to current file.

Returns

Returns the full path of the file.

Reimplemented from FileCoordinator.

3.8.2.4 string Simulator::simulate ( )

The main method available for clients. After setting the number of genes and type of simulation, the client needs only to call this method to simulate and save results in output files.

Returns

A message indicating location and type of simulation.

See Also

Interactor::startSimulator

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

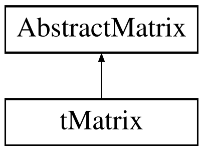
- Simulator.h
- Simulator.cpp

3.9 tMatrix Class Reference

Internal data structure representing a matrix that has constant column sums.

```
#include <tMatrix.h>
```

Inheritance diagram for tMatrix:



Public Member Functions

- tMatrix (const rMatrix &base)
- double module ()
- void set (int i, int j, double value)
- map$<$ int, double $>$ get (int col) const
- tMatrix & operator+= (const tMatrix &rhs)
- virtual void print (ostream &output)

Static Public Member Functions

- static tMatrix times (rMatrix lhs, const double k)

Private Attributes

- vector$<$ map$<$ int, double $>$ $>$ matrix

Additional Inherited Members

3.9.1 Detailed Description

Internal data structure representing a matrix that has constant column sums.

This matrix inherit the "print" method from AbstractMatrix to print the matrix as $n \times n$ matrix to an output stream.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 tMatrix::tMatrix (const rMatrix & *base*)

Construct a tMatrix using an rMatrix instance. The matrices are the same, but this instance supports tMatrix operations.

Note

Typical usage : tMatrix transition(R);

3.9.3 Member Function Documentation

3.9.3.1 map< int, double > tMatrix::get (int *col*) const

Get the colth column of the matrix.

Parameters

col	Column number;

Returns

The colth column as a map from int to double

3.9.3.2 double tMatrix::module ( )

Calculates the Frobinous module of the matrix as : $\sqrt{\sum_{i,j} a_{ij}^2}$.

Returns

The module of the matrix

3.9.3.3 tMatrix & tMatrix::operator+= (const tMatrix & *rhs*)

Defines addition between transition matrices.

Note

Typical usage :

```
transition += anotherTransition;
```

3.9.3.4 void tMatrix::print (ostream & *output*) [virtual]

Print the matrix in a n by n matrix form to the output stream

Parameters

output	The stream for output

Implements AbstractMatrix.

3.9.3.5 void tMatrix::set (int *i*, int *j*, double *value*)

Set the ith row jth column entry to the given value.

Parameters

i	Row number;
j	Column number;
value	The value to set.

3.9.3.6 tMatrix tMatrix::times (rMatrix *lhs*, const double *k*) [static]

Defines scalar multiplication on boolean matrix.

Parameters

lhs	The boolean matrix to multiply on;
k	The scalar;

Returns

A tMatrix instance as the result of the scalar multiplication

Note

Typical usage :

```
tMatrix transition = tMatrix::times(R, k);
```

3.9.4 Member Data Documentation

3.9.4.1 vector<map<int, double> > tMatrix::matrix [private]

A vector of map<int, double> storing the entries of the matrix. The matrix[j][i] represents the (i, j)th entry of the matrix.

Note

Only non-zero entries are stored, zero entries cannot be found in any of the maps.

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

- tMatrix.h
- tMatrix.cpp

4 File Documentation

4.1 AbstractMatrix.h File Reference

This is an abstract class/interface for matrix.

`#include <iostream>`

Classes

- class AbstractMatrix
  This class defines some of the basic operations we need for our matrix.

4.1.1 Detailed Description

This is an abstract class/interface for matrix.

4.2 Displayer.h File Reference

This is an interface for the "View".

Classes

- class Displayer::DisplayerClass

  Abstract interface for direct interation with end users.

4.2.1 Detailed Description

This is an interface for the "View".

This interface abstracts methods needed for the "View". Any controller class should implement this interface. Besides, the namespace Displayer also provides some interaction messages that could be presented to end users.

4.3 FileCoordinator.h File Reference

An abstract class for file manipulation.

```
#include <string>
#include <fstream>
#include "tMatrix.h"
```

Classes

- class FileCoordinator

  Abstract class for file manipulations.

4.3.1 Detailed Description

An abstract class for file manipulation.

This file contains the declaration of the abstract class responsible for file manipulation.

Author

Liang RuoChen

Version

2.1 09/04/2014

4.4 Interactor.h File Reference

An abstract controller class.

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include "Simulator.h"
#include "Iterator.h"
```

Classes

- class Interactor

  An abstract controller class that contains method prototypes or basic implementations of methods for further overriding.

4.4.1 Detailed Description

An abstract controller class.

This controller has reference to Simulator and Iterator objects that do the background computation.

Author

Vincent

Version

2.1 22/04/2014

4.5 Iterator.h File Reference

A file containing class definitions for Iterator.

```
#include "FileCoordinator.h"
#include <set>
```

Classes

- class Iterator

  This class encapsulates the functionality of an inverse iterator.

4.5.1 Detailed Description

A file containing class definitions for Iterator.

Apart from Iterator, this file also includes definitions for IteratorStatus and RandomTypes, which are important enumeration types for simulation and iteration.

4.6 Menu.h File Reference

This file contains declaration of the class Menu.

```
#include "Interactor.h"
#include "Displayer.h"
```

Classes

- class Menu

  This is a concrete controller class for the application.

4.6.1 Detailed Description

This file contains declaration of the class Menu.

Author

Liang RuoChen

Version

2.1 09/04/2014

4.7 rMatrix.h File Reference

This is the boolean matrix class.

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

Classes

- class rMatrix

  This class holds the data needed for representing a boolean matrix.

4.7.1 Detailed Description

This is the boolean matrix class.

Author

Liang RuoChen

Version

2.1 09/04/2014

4.8 Simulator.h File Reference

This file contains class definitions of Simulator class.

```
#include "FileCoordinator.h"
```

Classes

- class Simulator

  This is a class that handles simulation process.

4.8.1 Detailed Description

This file contains class definitions of Simulator class.

4.9 tMatrix.h File Reference

This file contains declaration of transition matrix class.

```
#include <map>
#include <iomanip>
#include "rMatrix.h"
```

Classes

- class tMatrix

  Internal data structure representing a matrix that has constant column sums.

4.9.1 Detailed Description

This file contains declaration of transition matrix class.

The class tMatrix is defined in this file, it represents the transition matrix whose solumn sums are one.

Author

Liang RuoChen

Version

2.1 09/04/2014

Index