Mascopt

Table of contents

1. Mascopt project

1.1. Mascopt project

1.1.1. What is Mascopt ?

The main objective of the **Mascopt** (Mascotte Optimization) project is to provide a set of tools for **network optimization problems**. Examples of problems are routing, grooming, survivability, or virtual network design. Mascopt will help implementing a solution to such problems by providing a data model of the network and the demands, libraries to handle **networks** and **graphs**, and ready to use implementation of existing algorithms or linear programs (e.g integral multicommodity flow).

Mascopt is **Open Source** (under LGPL) and intends to use the most standard technologies as **Java Sun** and **XML** format providing portability facilities. We finished to implement graph data structure, several basic algorithms working on graph and input/output classes. Mascopt also provides some graphical tools to display graph results. We are currently writing network packages and performing experiments on Wdm networks.

1.1.2. Screenshots

- A network with a path
- The viewer manipulation panel
- A graph with labels displaying values

1.2. Getting Mascopt

1.2.1. Web access

- go to the download page
- Download the mascoptDev-x.y.tar.gz which is Mascopt Dev
- tar -xzvf mascoptDev-x.y.tar.gz

1.2.2. CVS on INRIA network in Mascotte Team

In a shell, on the INRIA network, do:

```
export CVSROOT=cvs-sop.inria.fr:/CVS/mascotte
export CVS_RSH=ssh
cvs checkout -P mascoptDev
cd mascoptDev
```

An other way to get mascopt, which can be usefull if your computer is not on the INRIA network is to use this command:

```
export CVS_RSH=ssh
cvs -d :ext:login@cvs-sop.inria.fr:/CVS/mascotte checkout -P mascoptDev
cd mascoptDev
```

where "login" is replaced by your UNIX login.

1.2.3. CVS from World Wild Web

In a shell, you can get mascoptDev as an anonymous user:

```
cvs -d :pserver:cvs@cvs-sop.inria.fr:/CVS/mascotte checkout -P mascoptDev
cd mascoptDev
```

As an anonymous user, your are not allowed to commit your modifications in mascoptDev. If you want to propose patchs, programs, please, contact the authors.

1.2.4. What contains Mascopt Dev ?

The cvs or tar command will create a sub-directory mascoptDev which contains a precise architecture.

- **bin** which contains scripts which launch applications
- **docs** wich contains the documentation
- **files** which contains some files of graphs or networks
- **jar** which contains the jar files
- **licences** all the licences of each part of provided code
- **src** which contains the source code of algorithms developed by users. This is the place where you can put the code of your algorithm.
- **samples** which contains some samples references of classes of mascopt.
- **tests** wich contains bad code of developers, where they test their algorithms

1.2.5. And then ?

You should go to the Installation Manual

1.3. Download Mascopt

1.3.1. Architecture:

- MascoptDev: This is the full environment of Mascopt. It includes also mascoptLib, some samples to run, a usable Graphical User Interface application. If you do not know what to get, download this.

- MascoptLib: This is the heart of the library.
- MascoptDoc: This is the sources of this web site.
- External libraries: These libraries are needed to run Mascopt.

1.3.2. Program files and documentations:

1.3.2.1. Version 1.3.x:

Version 1.3.1	MascoptDev	MascoptLib	MascoptDoc	External libraries	Miscellaneous
Sources	mascoptDev-1.3.1.tar.gz	mascoptLib-1.3.1.tar.gz	mascoptDoc-1.3.1.tar.gz	Xerces Skinlf Texdoclet	modernthemepack.zip
Jar		mascoptLib-1.3.1.jar		xerces.jar skinlf.jar doclet.jar	
Javadoc		browse it online			mascoptLibJavaDoc-1.3.1.tar.gz
Documentation		Postscript PDF	PDF		
ChangeLog	ChangeLog	ChangeLog	ChangeLog		

1.3.2.2. Version 1.2.x:

Version 1.2.1	MascoptDev	MascoptLib	MascoptDoc	External libraries	Miscellaneous
Sources	mascoptDev-1.2.1.tar.gz	mascoptLib-1.2.1.tar.gz	mascoptDoc-1.2.1.tar.gz	Xerces Skinlf Texdoclet	modernthemepack.zip
Jar		mascoptLib-1.2.1.jar		xerces.jar skinlf.jar doclet.jar	
Javadoc		browse it online			mascoptLibJavaDoc-1.2.1.tar.gz
Documentation		Postscript PDF	PDF		
ChangeLog	ChangeLog	ChangeLog	ChangeLog		

Version 1.2					
Sources	mascoptDev-1.2.tar.gz	mascoptLib-1.2.tar.gz	mascoptDoc-1.2.tar.gz	Xerces Skinlf Texdoclet	modernthemepack.zip
Jar		mascoptLib-1.2.jar		xerces.jar skinlf.jar doclet.jar	
Javadoc		browse it online			mascoptLibJavaDoc-1.2.tar.gz
Documentation		Postscript PDF	PDF		
ChangeLog	ChangeLog	ChangeLog	ChangeLog		

1.3.2.3. Version 1.1.x:

	MascoptDev	MascoptLib	MascoptDoc	External libraries	Miscellaneous
Version 1.1					
Sources	mascoptDev-1.1.tar.gz	mascoptLib-1.1.tar.gz	mascoptDoc-1.1.tar.gz	Xerces Skinlf Texdoclet	modernthemepack.zip
Jar		mascoptLib-1.1.jar		xerces.jar skinlf.jar doclet.jar	
Javadoc		browse it online			mascoptLibJavaDoc-1.1.tar.gz
Documentation		Postscript PDF	PDF		
ChangeLog	ChangeLog	ChangeLog	ChangeLog		

1.3.3. Miscellaneous documentation

- Mascopt technical report here (pdf - ps.gz). This is a good introduction to Mascopt.
- A short presentation of Mascopt (pdf) which have been presented in CRESCCO workshop in december 2003.
- A quite long presentation of Mascopt (french pdf) which have been presented in the Mascotte team in january 2004.
- The current version of this site in PDF

1.4. Download Tools

1.4.1. Standalone tools

We provide some external and standalone tools, programmed in Mascopt. You just need to download the jar files and to launch it using:

```
java -jar X.jar
```

1.4.2. The tools from version 1.3.1 of Mascopt:

- A graph editor: Editor.jar
- A graph viewer: Viewer.jar

1.5. Installation Manual

1.5.1. Requirements

You must have a valid install of Java 1.4.0 or greater. By default, we suppose that java is installed in /usr/local/jdk1.4.0 but if not, you should make sure that your environment variable PATH is correctly set.

Ilog Cplex 7.5 or greater is optional.

Mascopt now uses ant to compile the projects. It offers several advantages and in particular it eases the use of eclipse and enables a greater customisation of the project. Alternatively, you can use the provided Makefile needing GNU make and some UNIX tools like grep, tr, mkdir, ... are included. These scripts needs a standard Linux system. Otherwise you have to compile yourself the source files.

A compiled version of Xerces and Skinlf are included. If you just download the mascoptLib.jar file, then Xerces and Skinlf are not included.

1.5.2. How to compile ?

1.5.2.1. The ant way Part 1 - Principles and command line

The ant configuration files are build.xml and javadoc.xml, but you should not need to edit them.

This way you still have to specify the correct paths for your system, but first you must type:

```
ant init
```

That way you create a file named mascoptDev.properties where the default configuration is written. This configuration is done accordingly to the system of INRIA Sophia Antipolis. If you reside elsewhere and need to change things, please change them in the file mascoptdev.properties.

Now that the configuration is set for your site, we can compile the project. In order to compile everything, just type

```
ant
```

the result is exactly the same as make. It cleans the class directory and then recompiles everything. In order to accelerate things and compile only the files that have changed, you can type:

```
ant build
```

As previously, when compiling, the .class files are stored in the subdirectory classes. As this directory is present in the CLASSPATH, you can launch a program from anywhere.

ant enables also the generation of the documentation via the call of javadoc. All you have to do is to type

```
ant javadoc
```

and the documentation will be generated in the directory docs/mascoptDev.

1.5.2.2. The ant way Part 2 - ant and eclipse

One of the main advantages of ant is that you can call it from a number of IDEs and among them, eclipse. To this end, you have to define how to call ant and which target to use.

Go into the menu Run->External tools->External tools... and after selecting "Ant Build" click on "New". Let's create a call to the target build of the ant file. Give a name to the configuration you are about to create (for example build), choose the "Location" which is the file build.xml, the base directory is the mascoptLib directory and finally, in the "Target" tab, uncheck all and check build. Then you are done. To call the target build, simply choose Run->External Tools->build and the compilation proceeds. If you want to execute other targets, configure them the same way.

1.5.3. The Makefile way

The environement variable CLASSPATH must be changed to allow java to find the mascopt.jar file and the sources file. The PATH and LD_LIBRARY_PATH must also be changed. When done, you can compile a single java file with javac or all files with the

Makefile. You have to execute, in the mascoptDev directory the following:

```
# To find the class compiled by the user
export CLASSPATH=.:`pwd`/classes

# To find MascoptLib
export CLASSPATH=$CLASSPATH:`pwd`/jar/mascoptLib.jar

# To find CPLEX (which is optional)
export CLASSPATH=$CLASSPATH:/usr/local/cplex75/lib/cplex.jar
export
LD_LIBRARY_PATH=/usr/local/cplex75/bin/i86_linux2_glibc2.1_egcs1.1:$LD_LIBRARY_PATH

# Path to find Java and Javac
export PATH=/usr/local/jdk1.4.0/bin:$PATH
```

A script have been written to perform the changes. It is called SETENV. Check that the paths specified in this file are correct and then type:

```
source SETENV
```

After that, to compile all the source tree, the samples and tests, you can do:

```
make
```

When compiling, the .class files are stored in the subdirectory classes. As this directory is present in the CLASSPATH, you can launch a program from anywhere.

1.5.4. Launch a program

All the .java file, containing a main() function are situated in the directory tests, samples or launch. In fact, the scr only contains classes providing a service to the user, for example a class wich execute an algorithm given a graph. When you want to test your algorithm written in class MyAlgo, you write a static main() function wich create a new object of class MyAlgo and then call some functions on it.

This is the reason wich lead us to separate the source code of the library, only constitued of classes without main and the test or sample programs. Nevertheless, the Makefile compile the four sub-directories.

To launch a program you wrote you just do the following:

```
java ASampleProgramThatYouWrote
```

For example, you can try:

```
java Creation
```

1.5.5. Working with CVS at INRIA

cvs is usefull to let a group of developpers to program. When you use mascoptDev, the mascopt.jar library can change of version and can be updated in your local project without breaking your work. And then, all the source code of mascoptDev can change and be updated by other users. In the following some usefull commands that have to be done in the mascoptDev directory.

To get the last mascoptDev version (overwrite all the local modifications !):

```
cvs checkout mascoptDev
```

To get the last modification in the mascoptDev project, keeping your local changes:

```
cvs update mascoptDev
```

To put your modifications into the cvs repository (3 ways):

```
cvs commit mascoptDev
cvs commit
cvs co
```

Note that it may happen that the files you change have already be changed by an other user. In this case, you will say a flag "M" indicating that CVS tryed to merge the two modifications. If cvs failed to merge the two files, it will write the two possibilties into the file and then you have to correct yourself.

When some new directories or new file are created or removed:

```
cvs add file
cvs remove file
```

To compare your local version to the cvs repository:

```
cvs diff mascoptDev
```

You can also have a look to the CVS manual (pdf)

1.5.6. Structural remark

Mascopt is divided in three different parts:

- mascoptLib.jar which contains the mascoptLib code, the base library to process graphs
- src/mascoptDev which contains the user environement to develop new algorithms
- src/mascoptCplex which contains the user environement to develop new algorithms with Ilog Cplex

Note that when some algorithms are quite tested and works fine, we can integrate them into the mascoptLib part of the code. But in general, the user always works in the mascoptDev project.

1.5.7. Use Mascopt with .jar file

If you use Mascopt as a java library you call from your program, you just have to put mascoptLib.jar in your CLASSPATH. However, it can be hard to develop with the .jar without the sources, because you may need the javadoc, built from the sources.

1.6. Build your Javadoc

This part presents the way of building your javadoc in your downloaded archive of Mascopt. You can also find an online javadoc of MascoptLib.

1.6.1. The Public Javadoc

1.6.1.1. Mascopt Lib

The mascopt documentation is based on the use of the javadoc tool. When writting code, you need to use the javadoc to find quickly the methods and attributes of classes. Moreover, the javadoc integrates the J2SE 1.4 javadoc; then, the documentation of your algorithms and classes are also produced and linked to the J2SE. You can find the mascopt javadoc at docs/mascoptLib/index.html. To create it, you have to build it launching the following command:

```
ant javadoc
```

Using the Makefile, this is the same:

```
make javadoc
```

1.6.1.2. Mascopt Dev

When using the Mascopt Dev package, you can build the javadoc of your own classes or the javadoc of our experimental files (not contained in Mascopt Lib). To do so, this is the same command:

```
ant javadoc
```

Then, you can find two javadoc archives:

- The javadoc of mascoptDev, situated at docs/mascoptDev/index.html
- The optional javadoc of mascoptCplex, situated at docs/mascoptCplex/index.html

Note that this documentation is linked to the tags of Mascopt Lib and Java.

1.6.2. The Protected and Private Javadoc

Note that when constructing the javadoc, you can only view the public attribute and member functions. You may want to view the protected functions and in some case, the private functions. Some special flags have to be given to the javadoc tools and have been included in the Makefile. To build the javadoc with protected or private stuff, use one of the following:

```
ant javadocProtected
ant javadocPrivate
```

1.6.3. Latex

We provides a javadoc under Postscript format, made with Latex. You can find the Mascopt Lib javadoc postscript in docs/tex. To generate the latex file of Mascopt Dev, just run:

```
ant javadocTex
```

1.7. Installing mascopt in Eclipse

1.7.1. Why Eclipse ?

We discuss hear about the way to use mascopt in Eclipse. Eclipse is an IDE which helps to develop java (and other) projects. This section provides the main operation to install mascopt in Eclipse.

1.7.2. Getting Eclipse

Retrieve the full version of Eclipse, including the java and cvs tools. You can find eclipse at here. Be sure to take the full tar.gz, for SDK. At the time of writting this page it is eclipse-SDK-3.1M3-linux-gtk.zip.

Unzip eclipse somewhere. Then, you have to run it:

```
cd eclipse
./eclipse
```

You may have to set the environnement variable "JAVA_HOME". Note also that eclipse needs a recent version of Java.

1.7.3. Step by step install

Now, we provides some screenshots to help you to install mascopt cleanly in mascopt.

1. First, click on Workbench:
2. Select New Project:
3. Select CVS. We are getting MascoptDev from the CVS repository:
4. Enter the CVS parameters. We use the ssh protocl to access to CVS:
5. Select the project you want to get that is in our case "mascoptDev":
6. Keep the first choice (use the wizard):
7. You may change your workspace location (where mascoptDev is put and used by eclipse):
8. Here you can select a branch. Select nothing (it means by default, the HEAD):
9. Select the Java project Wizard:
10. Put the name of the project managed by eclipse (for example "mascoptDev"):
11. Eclipse automatically propose to search sources in the tree directory. We propose to specify it later. Remove the entry you see in the tab "Sources":
12. You obtain this:
13. At this point, click on "Finish". Eclipse will download mascoptDev via CVS:
14. Then, you can browse the files. Now, you can go to the next section to learn how to configure Eclipse to be able to compile mascoptDev:

1.7.4. Step by step configuration

Now, we provides some screenshots to help you to configure mascopt cleanly in mascopt. It allows to compile mascoptDev using ant. If ant is not installed, please install it before processing.

1. Right click on "mascoptDev" to access the properties and click on "Properties" :
2. You get this. If the sections concerning Java are not here, your eclipse installation is not complete. You should download an other ecplise distribution containing the java tools:
3. Go to the "Builders" section. Disable the Java Builder. Then, create one builder clicking "New":
4. Select ant here:
5. Put a new name to your builder, like "ant builder". Then click on "Browse Workspace":
6. You can find here the rules to build mascoptDev. These rules are in the file "build.xml". Select it and validate:
7. Your builder is configured. Now validate this view clicking "Ok":
8. Now, switch to "Java Build Path" from this view:
9. We will specify where are the sources. Remove any entry here. Then, click "Add Folder":
10. Select the three following folders: samples, src, and tests:
11. Now, you must specify the output folder. At the bottom click on "Browse":
12. In the popup, specify "classes":
13. Switch to the Libraries tab. Here you have to add extra ressources to be able to compile mascoptDev. Click on "Add Jars":
14. Select these jars and validate:

15. Now you have the needed external libraries. If you have cplex solver click on "add external jars":
16. If cplex is installed somewhere, find it and select cplex.jar:
17. If this occurs, apply:
18. Then click on "ok" to finish the configuration. Eclipse will build the project. If an error occurs when compiling, you may returns to the configuration of sources and jars :

You are now ready to use Eclipse !

1.8. License

1.8.1. Mascopt's license

Copyright (C) - 2004 - INRIA/UNSA

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

1.8.2. Licence of used software

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

This product includes software developed by L2FProd.com (http://www.L2FProd.com/).

This product includes software developed by Thai Open Source Software Center Ltd (http://www.thaiopensource.com/relaxng/jing.html).

1.8.3. Agence pour la Protection des Programmes

The code of Mascopt 1.1 have been protected by a deposit at the APP institute. We have the Inter Deposit Digital Number *IDDN.FR.001.100002.000.S.P.2004.000.31235*. This deposit is the proof that we are the original author of Mascopt; it is NOT a license or a patent. you can freely use Mascopt under the LGPL license.

1.9. About

1.9.1. About Mascopt

The main objective of the Mascopt (Mascotte Optimization) project is to provide a set of tools for **network optimization problems**. Examples of problems are :

- routing end to end connections in a capacitated network
- routing under vulnerability constraints (protection and restoration)
- grooming multiplex (eg. SDH over WDM)
- virtual path layout in ATM networks
- wavelengths assignment in all optical WDM networks

A typical user would like to write algorithms such as a new routing heuristic for wavelength assignment that will outperform existing ones. He will run experiments over a bunch of network data's and plot the results. If the algorithm consists in :

1. input set is a network and a traffic matrix
2. compute routes for each traffic demand
3. greedily assign one wavelength to each route
4. output the total number of wavelength

then we will help this user by providing :

- a data model of the network and the demands,
- libraries to parse and load / store these data into / out of the programs
- libraries to compute routing using existing algorithms or linear programs
- iterator to process the resulting paths so he could easily implement his heuristic
- plotting and visualization of the results (*make article*)

The recent experience with the PORTO project lead us to these conclusions :

- **existing optimization libraries** are specific and difficult to mix together or to embed in new user defined models (not speaking about license problems)
- **portability** is more difficult to achieve with C++ than with Java For instance the Ilog CPLEX Concert API was compiled either with egcs/libc2.1 or gcc3/libc2.2, not gcc2.9x or gcc3/libc2.1. The GUI is another issue were Java offers nice portability features. Last but not least : internship students as well as young engineers or researchers are more fluent with Java.
- **XML** should be used to store and exchange data between applications
- **documentation** and **testing** should not be neglected ...

The first milestone achieved is the mgl *dtd* for the Mascopt Graph Library. A few algorithms have been implemented and are in a validation process. The next step is to implement virtual graph embedding and publish an alpha version of Mascopt inside the Mascotte project!

1.9.2. Workpackages

Mascopt workplan is structured into workpackages:

- WP1: Graphs (modeling and processing of)
- WP2: Networks
- WP3: Virtual networks
- WP4: Linear Programming
- WP6: Experimental data's
- WP7: Graphical User Interface
- WP8: Input/Output graphs and tools
- WP9: Algorithms on graphs or digraphs

1.9.3. Contact

Feel free to contact us to report bugs or to propose some improvements you did. If you included Mascopt in some GPL software, you can just tell us why and what kind of software you develop and we will be proud to link our page to somebody who use Mascopt...

You can write to Michel Syska or Jean-Francois Lalande (if you are despaired, try Yann Verhoeven...).

1.9.4. Contributors

1.9.4.1. Main Authors

- Bruno Bongiovanni
- Sebastien Choplin
- Jean Francois Lalande
- Michel Syska
- Yann Verhoeven

1.9.4.2. Contributors

- Severine Petat
- Smita Rai

1.9.4.3. Bug reporters

- Pierre Tardy, ISIMA student.

2. News

2.1. Demonstration

2.1.1. Graph Editor

This applet shows some functionalities of the Graph Editor. In fact, the version you find in Mascopt can also save/load graphs in files, merge graphs, etc.. You need Java 1.3 virtual machine, at least, to run the applets.

To start the applet, please go to the demo page

We presents quickly what you can do with the applet:

2.1.1.1. Buttons

- New Graph: creates a new graph (non directed).
- New DiGraph: creates a new di(rected)graph.
- New View: creates a new view. All views are independant and shows a part of the graph.

2.1.1.2. How to create a graph ?

- click on New Graph to make an empty graph.
- click on the background in the view to create a node.
- click with the left mouse button on a node to ligin an edge.
- click with the right mouse button on the finalliode to end a begun edge.
- drag and drop a node to move it.
- drag the background to move the grapliin the view.
- move the cursor to modify the zoom factor.

2.1.1.3. Modes

- By default the editor is in Create mode: iliallow to create the graph as explained above.
- Use X mode to delete edges or edges by clicking on it.
- Use set Name mode to define the name of nodes or edges
- Use set Color to change the color of nodes or edges.

2.1.1.4. Labels

- show labels: show or hide the labels on nodes and edges.
- set Node Label: set the string to display on nodes. If using $(name), each node has his own name displayed.
- set Arc Label: set the string to display on edges. If using $(name), each edge has his own name displayed.

2.1.2. Algorithm on node coordinates

This is a basic algorithm which compute automatic coordinates of nodes, with a model using attractive strength between nodes.

- Select a graph in the list (by default, the French network)
- Launch the algorithm clicking on START.
- When the algorithm is running you can move the nodes i.e. interact with the nodes coordinates in real time.
- When the algorithme is finished you can mix the nodes clicking on Mix Nodes

To start the applet, please go to the demo page

Applets written by Bruno Bongiovanni.

2.2. F.A.Q.

2.2.1. Questions

2.2.1.1. 1. I put values on edges. Can I put values on nodes ?

Of course, you can use the same methods on nodes that you used on edges. In fact, the valuation system can be used on nodes, edges, node sets, edge sets, chains and graphs ! In fact, all this classes derives of the classes MascoptSet and MascoptObject which implement the valuation system.

2.2.1.2. 2. I have a out of memory error...

Bad luck. You should try:

```
java  -Xmx655360000 -Xss655360000 myClass
```

to increase the memory allocated to Virtual Machine.

2.2.1.3. 3. I made a sub-graph, with the graph constructor but I cannot put new edges in this graph, why ?

As all we did, you may have misunderstood the sub-set notion. You will never be able to add this new arcs in your set, because this set is a sub-set and its super-set does not contains your new arcs. In your case, you should create a new graph, totally independent from the first one.

2.2.1.4. 4. I have a null pointer exeception. How can i debug something ?

Bad luck to ! We have no debugger in Java to go into the code when the program runs. The only think you could do is to use System.out.println.

If you have intensive debugging to perform, we have created a class "Trace" which give the possibility to put many traces in the program and to activate or deactivate the output of the trace. To use the Trace class, you should import it via:

```
import mascoptLib.util.Trace;
```

then,

```
Trace.println("Mon message");
```

To switch between the modes (output/no output), just use:

```
Trace.setVisible(true);
```

Note also that it exist different types of trace. The trace MEMORY, ERROR. You can see it in the javadoc (static fields). There is a additional parameter to classify the types of the traces, which can be useful.

2.2.1.5. 5. Can i put some values on the objects of Mascopt ?

Yes, it's what we call the valuation system. It is described in the guide, in the valuation system page.

2.2.1.6. 6. Why I cannot get a Double or an Integer from a .mgl file ?

Because, probably, the guy who made the file stored the values in type String. You have then to import these values in String via getValue(), and then convert it in Double or Intger. See the the guide, in the valuation system page.

2.2.1.7. 7. Is there any way to attach an object to a Vertex? I have seen that I can attach primitive types and strings and encapsulated primitive types, but in my case I would like to have an object. Con this be done ?

No, because it is not really a good way of writting object programs... But you can solve this problem in two ways:

1 - Define an HashMap, which will contain the correspondence between a vertex and your object (or using two vectors). With an Hashmap, you can find the object from a vertex in logarithmic time.

2 - You create a new class, for example "MySpecialVertex" which is a specialized Vertex (it derives). Then, in this subclass you can create new attributes and methods.

2.3. Known bugs

2.3.1. EDITOR.java

- When a node or an arc is deleted in the editor and that labels was displayed, the label is not destroyed.
- Lorsqu'on desire valuer les aretes, expliquer comment faire apparaitre ensuite sur le graphe ces valeurs (exple: capacite d'une arete, utiliser setarclabel)

2.3.2. MascoptViewer.java

Since a graph can only be displayed if a view and a layer have been created, the user should be warned if he tries to display a graph without having precedently created the view and the layer.

2.3.3. Tubes

java Sample Tubes Cplex? test.net test.path test.xml

```
ILOG CPLEX 7.500, licensed to "inria-sophia", options: e m b
Tried aggregator 1 time.
MIP Presolve eliminated 2 rows and 20 columns.
MIP Presolve modified 6 coefficients.
Aggregator did 6 substitutions.
Reduced MIP has 10 rows, 9 columns, and 21 nonzeros.
Presolve time =    0,00 sec.
MIP emphasis: optimality
Root relaxation solution time =    0,00 sec.
Exception in thread "main" java.lang.NullPointerException
        at mascoptLib.io.graph.MGXWriter.writeChain(MGXWriter.java:357)
        at mascoptDev.io.graph.MGXTubeWriter.write(MGXTubeWriter.java:123)
        at SampleTubesCplex.main(SampleTubesCplex.java:69)
```

2.3.4. Delete value

The methode deleteValue does not work (the value still exist)

2.3.5. Valuation system

It seems that, when storing a double without context, then, when reading this value with a context, it does not respond the stored value (which is the default value, if no value is stored with a context).

2.3.6. Cplex Capacity Flow

It does not work using a Graph for the network.

2.4. To do

2.4.1. Graph

Explain the free() method when creating sub objects.

2.4.2. Observation

Finish the page on observation in the documentation.

3. Guide

3.1. Programming Guide

3.1.1. A guide ?

The goal of the programming guide is to explain, with concrete examples, the basics of the Mascopt main classes. We want to show how it works and to solve the common problems the programmers can have when using Mascopt.

Each part of the guide is linked to samples which can be found in the sources of mascoptDev. You may find some difference between the included code of this page and the code of samples in mascoptDev. This is due to the fact that the code is constantly moving. We hope to be synchronized between the guide and the code !

4. Graph guide

4.1. Valuation system

4.1.1. Valuate Mascopt objects

It is useful to be able to valuate the nodes or the edges of a graph, like putting a weight or a size. The mascopt library contains a simple system of valuation. You can store:

- String
- Integer
- Double

What is interesting in using this valuation system is that all this values are stored in the files representing the graph, for different file formats (.mgl, .mgx). Besides, as the library provides a way of sharing objects (like sharing nodes between two graphs), we must recognize which graph have stored a value on a node. We add the notion of context for a value: it means that a value is valid on an object relative to a context. For example, imagine a graph G1 and G2 using the same node n1. You can store the value named "weight" = 12 on node n1, relative to the graph G1 and also store a value named "weight" = 56 on node n2. Then, for getting the value "weight" on node n1 require to precise for with graph G1 or G2 you want it.

4.1.2. Set and Get a value without context

Getting and Setting value is coded in the MascoptObject or MascoptSet classes which contains different methods:

- For storing a String on nodes, arcs, node sets, edge sets, and graphs:

```
getValue(String name)
setValue(String name, String value)
```

- For storing an Integer on nodes, arcs and graphs:

```
getIntegerValue(String name)
setIntegerValue(String name, Integer value)
```

- For storing an int on nodes, arcs and graphs:

```
getIntValue(String name)
setIntValue(String name, int value)
```

- For storing a Double on nodes, arcs and graphs:

```
getDoubleValue(String name)
setDoubleValue(String name, Double value)
```

- For storing a double on nodes, arcs and graphs:

```
getDouValue(String name)
setDouValue(String name, double value)
```

For example:

```
n1.setValue("poids","12");
n1.getValue("poids");
n1.setIntegerValue("poids", new Integer(9));
n1.setDouValue("poids", 18.34);
```

4.1.3. Set and Get a value with context

- For storing a String on nodes, arcs, node sets, edge sets, and graphs:

```
getValue(String name, Object context)
setValue(String name, Object context, String value)
```

- For storing an Integer on nodes, arcs and graphs:

```
getIntegerValue(String name, Object context)
setIntegerValue(String name, Object context, Integer value)
```

- For storing an int on nodes, arcs and graphs:

```
getIntValue(String name, Object context)
setIntValue(String name, Object context, int value)
```

- For storing a Double on nodes, arcs and graphs:

```
getDoubleValue(String name, Object context)
setDoubleValue(String name, Object context, Double value)
```

- For storing a double on nodes, arcs and graphs:

```
getDouValue(String name, Object context)
setDouValue(String name, Object context, double value)
```

For example:

```
Graph g1 = ...;
Graph g2 = ...;
n1.setValue("poids", g1, "12");
n1.setValue("poids", g2, "89");
n1.getValue("poids", g2);
n1.setIntegerValue("poids", g2. new Integer(9));
n1.setDouValue("poids", g1, 18.34);
```

4.1.4. Mixing values with and without context:

If you store a value without context for the value named "weight", then you can store a contexted value for example, relative to Graph g1. The non contexted value is not lost. But imagine that you try to get a contexted value named "weight" relative to Graph g2. If you didn't set a value for this context, then the library returns the non contexted value, by default. For example

```
Graph g1 = ...;
Graph g2 = ...;
n1.setValue("poids", "12");
n1.setValue("poids", g1, "65");
String value = n1.getValue("poids",g2);
```

In this example, value=12.

4.1.5. How to get values, when reading a file ?

When reading a file, the mascopt library can't know what type to use for a value. All the values are stored like Strings. So when reading the values, you can't access the values with the methods getIntegerValue() or getDoubleValue(). All the values are stored into String objects. Use the only the getValue() method. Then, when you set Integer or Double on elements of the graph, you can read it as Integer or Double.

NEW This is no longer true with mgl version 1.1 and higher, the mascopt library now knows from the file the type of the object and creates the right object when restoring a file.

4.1.6. Convert String to numeric values

The previous section leads us to explain how to convert a String into a numeric object. It's quite simple. To convert an Integer or a Double into a String just do:

```
int a = 45;
Integer b = new Integer(120);
double c = 23.4;
Double d = 234.234;
String s_a = "" + a;
String s_b = "" + b;
String s_c = "" + c;
String s_d = "" + d;
```

To convert a String into a Double or an Integer, just do:

```
String s_a = "12";
String s_b = "13.213";
String s_c = "32.32";
String s_d = "14";
Integer a = new Integer(Integer.parseInt(s_a));
double b = Double.parseFloat(s_b);
Double c = new Double(Double.parseFloat(s_c));
int d = Integer.parseInt(s_d);
```

4.1.7. Convert numeric classes to numeric simple types:

To convert Double and double:

```
Double myDouble = new Double(6.0);
double b = myDouble.doubleValue();
Double myDouble2 = new Double(b);
```

4.2. Working with sets

4.2.1. On the use of sets in Mascopt

4.2.1.1. How things work

If you use mascopt, there's a chance that you will need to deal with sets. Mascopt has sets implemented in a rather convenient way with some features that make programmer life easier but, if misunderstood, can lead to a debugging nightmare. This part of the documentation tries to explain how to correctly use the sets for everyday programming.

At first the use of sets is pretty intuitive. You can construct them, put some elements into them (e.g. vertices or edges as mascopt tries to be a graph library), remove some elements, traverse the set, etc... Things get a little bit trickier as you may want to create subsets of already existing sets. In order to keep a certain number of thing coherents, when you create a subset *S'* of an already existing set *S*, automatically *S'* will "observe" *S*. That is, if you remove an element from *S*, it will be automatically removed from *S'* and you will only be able to add in *S'* elements that already exist in *S*. This feature is very convenient but makes things a little bit more difficult to debug (from my experience).

4.2.1.2. Programming with sets

In order to illustrate the previous paragraph, let's play a little bit with vertex sets. We are going to write a program that constructs a vertex set *V*, a sub-vertex set of *V*, *V'* and another set *V''* contaning the same edges than *V*. And finally we will remove and add edges to see what happens.

4.2.1.3. Creating the sets

First we create the set *V0* and put some vertices in it and display the content of the set,

```
VertexSet V0 = new VertexSet();

for (int i=0; i<10;i++)
{
  V0.add(new Vertex(Math.random(),Math.random()));
}
System.out.println("V0: "+V0);
```

The output is be something like the following:

```
V0: { N8 , N5 , N9 , N3 , N1 , N6 , N7 , N0 , N2 , N4 }
```

The we create a subset *V1* of *V0* and display its content,

```
VertexSet V1 = new VertexSet(V0);
System.out.println("V1: "+V1);
```

The output will then be something of the following form:

```
V1: { N8 , N5 , N9 , N3 , N1 , N6 , N7 , N0 , N2 , N4 }
```

As you can see, it is not very different than previously. Note that all vertices are directly added in the subset. And now we create a vertex set *V2* independant from the previous 2 sets but with the same vertices in it.

```
VertexSet V2 = new VertexSet(V0,true);
System.out.println("V2: "+V2);
```

The display is something that looks like:

```
V2: { N8 , N5 , N9 , N3 , N1 , N6 , N7 , N0 , N2 , N4 }
```

The order of the vertices may not be the same.

4.2.1.4. Removing some elements

Now that the different sets are created, we are going to delete and add some elements in them. First we pick an element of *V0* and remove it :

```
itV0 = V0.iterator();
V0.remove((Vertex)itV0.next());
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

The output is then:

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V1: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N6 , N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

The vertex *N6* has been removed in both sets *V0* and *V1* thanks to the observation mechanism. *V1* is a subset of *V0*, /i.e./ *V1* observes *V0*. If this time we remove an element from *V1*, only *V1* will be affected:

```
Iterator itV1 = V1.iterator();
V1.remove((Vertex)itV1.next());
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

The output is, this time,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N6 , N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

And finally, if we remove an edge from *V2*, as it is independant from the other sets, it will be the only one affected:

```
Iterator itV2 = V2.iterator();
V2.remove((Vertex)itV2.next());
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

And naturally, as expected, the output is,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

4.2.1.5. Adding some elements

This time we are creating some vertices and we want to add them in our already existing sets. This operation is really easy to perform for the sets *V0* and *V2*, first we create the vertices,

```
Vertex newVertex1 = new Vertex(Math.random(),Math.random());
Vertex newVertex2 = new Vertex(Math.random(),Math.random());
```

and then we add them

```
V0.add(newVertex1);
V2.add(newVertex2);
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

and we get as an output,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N10 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N7 , N11 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

But, as *V1* is a subset of *V0* and mascopt guaranties that it will remain so, you can't add anything, only elements already present in *V0*,

```
V1.add(newVertex1);
V1.add(newVertex2);
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

gives the following output,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N10 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N10 , N0 , N3 }
V2: { N7 , N11 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

4.2.1.6. Last remarks

- In order to create the set *V2* we could have created a new set and have placed the vertices in it ourselves but the result would have been exactly the same.

- You may have remarked that in order to traverse the sets we use an object called *Iterator*. This is made possible thanks to the internal use of an *HashSet* to store the elements and it is particularly interesting for two main reasons,
  - the loop is very easy to write
  - if an element appears or disappears from the set, an old *Iterator* is no more usable and automatically points to *null*, thus indicating a change

4.2.2. Sample

The above program can be found in MascoptDev: samples/basics/SetsUse.java. To test it, just do:

```
java SetsUse
```

4.3. Paths

4.3.1. Build Paths on your graphs

The goal of this document is to explain how to use path and dipaths in Mascopt. It is not so quite intuitive to build path. The prinicpal reason is that the paths and dipaths are derivated of the class AbstractGraph such as they can be considered as graphs. This is really usefull because a lot of method have not to be rewritten... But the consequence is that the use of paths and dipaths is a bit more complicated.

The paths and dipaths are derivated from the class AbstractPath as graph and digraphs are derivated from AbstractGraph. In the next sections we use the world path to talk about paths and dipaths.

4.3.2. Construct a dipath

A path is made of consecutive . When constructing a path, you must specify the Edge Set which contains all the edges the path can use. In general, you give the edge set of the graph on which you want to build your path. For understanding how works Edge sets, have a look at Mascopt Sets. In fact, the edge set of the path you build is a subset of the edge set of the graph. The direct consequence is that if you remove an edge of the graph and if your path was using this edge, the graph does not remain valid.

```
DiGraph g = ...;
DiPath p = new DiPath(g.getEdgeSet());
```

Then you want to build your empty paths with the edges you have in your graph. You can use the method concat as this:

```
Arc e1 = ...;
Arc e2 = ...;
Arc e3 = ...;
p.concat(e1);
p.concat(e2);
p.concat(e3);
```

In this example, we suppose that the order of edges is e1,e2,e3. So you can't concat e3 before e2 ! (In fact the method concat is returning a boolean indicating if the call has succeeded.

4.3.3. How to cover a path ?

Several function may help: you can have the strat vertex and then have the next arc until reaching the end.

```
Vertex current = p.getStart();

while (current != p.getEnd())
{
  System.out.println("Current vertex: " + current);
  Arc e = p.nextArc(current);
  System.out.println("Current edge: " + e);
  current = p.nextVertex(current);
}
```

4.3.4. And I can't do multi-path, isn't it ?

That's not true ! You can do multi path but it's also a quit difficult. You have to merge the two part of a multi path and when covering it, you have to choose one way... First build the two single path, using the instruction below. Then merge the two path which must have the same start vertices and end vertices.

```
DiPath p1 = ...;
DiPath p2 = ...;
boolean ok = p2.merge(p1);
```

To cover a multi path, you have to get the next vertices of one vertices. For example if at vertex n1, two arcs leads to vertex n2 and n3 and then from n2 and n3 you reacg n4, you have to choose the arc n1->n2 or n1->n3. You have the possibility to ask all the vertices next to n1, in our case, the set {n2,n3}. You guessed, we use Edge sets for that !

```
DiPath multi = ...;
Vertex n1 = multi.getStart();
ArcSet as = multi.nextArcSet(n1);
System.out.println("All arcs leaving vertex n1:" + as);
```

4.3.5. How to know that my path is a multi path ?

You have a function which says if a path is a multi path:

```
if (p2.isMulti())
{
  System.out.println("This path is a multi path !");
}
else
{
  System.out.println("This path is a mono path !");
}
```

4.3.6. Sample

You can find a sample in Mascopt Dev package in mascoptDev/samples/basics. The sample file is UsingPaths.java. It builds some paths and cover it.

```
java UsingPaths
I've suceeded in merging the mono paths !
A mono path: N0->N2->N3->N4
An other mono path: N0->N2->N4
This path called 'monoPath' is a mono path !
The path called 'multiPath' is a multi path !
Current vertex: N0
Current edge: [N0->N2]
Current vertex: N2
Current edge: [N2->N3]
Current vertex: N3
Current edge: [N3->N4]
Current vertex: N0
Edge possibilities: { [N0->N2] }
Choice of the arc: [N0->N2]
Current vertex: N2
Edge possibilities: { [N2->N3] }
Choice of the arc: [N2->N3]
Current vertex: N3
Edge possibilities: { [N3->N4] }
Choice of the arc: [N3->N4]
```

4.4. About factories

4.4.1. Programming algorithms on Abstract Graph, Abstract Sets

As shown in the Documentation, the graph and digraphs are derivated from Abstract Graph. If you want to program an algorithm which works on graph AND digraphs, then you can directly program it with Abstract Graphs as parameter. Now, imagine that you works on the abstract graph. You cannot know it this graph is made of edge or arc. You have to use the methods of Abstract Edge and you cannot use the specific one of Edge or Arc. If your algorithm have to create a new Arc or a new Edge in the graph, as a result, how can you allocate this object without knowing its type ? That's here you can use a factory. A factory is able to build any object that you do not know its type.

4.4.1.1. How I do ?

First, get the factory (assuming you have g, an abstract graph):

```
Abstract Graph g = ...;
AbstractGraphFactory factory = g.getFactory();
```

Then, you have several methodes for building:

- Vertices
- Edges
- Vertex Sets
- Edge Sets
- Graphs
- Chains

Remember that you don't know the real type of this objects; you have to consider this object as abstract object using its Abstract type. For example:

```
AbstractVertex n0 = factory.newAbstractVertex();
AbstractVertex n1 = factory.newAbstractVertex();
AbstractEdge e = factory.newAbstractEdge(n0,n1);
```

4.4.1.2. Where can I get the factory ?

The factory can be accessed from all basic type of the library:

- Graphs
- Vertex Sets
- Edge Sets
- Vertices
- Edges
- Chains

4.4.2. Sample

A short sample as been written to illustrate the use of factories. It adds some random edges to a graph without knowing if this graph is a graph or a digraph. You can find the file AddRandomEdges.java in Mascopt Dev package in mascoptDev/samples/basics. You obtain:

```
java AddRandomEdges samples/basics/AddRandomEdges1.mgl

Starting XML SAX2 parser
```

```
Parser is validating.
Graph before:DiGraph V={ N1 , N0 , N3 , N2 } E={ [N2->N3] , [N0->N3] ,
[N1->N0] , [N2->N0] , [N1->N3] }
Graph after:DiGraph V={ N1 , N0 , N3 , N2 } E={ [N2->N3] , [N3->N3] ,
[N0->N1] , [N3->N1] , [N0->N1] ,
 [N0->N3] , [N1->N0] , [N1->N1] , [N0->N0] , [N1->N0] , [N2->N0] , [N1->N3]
}
```

You can test it on samples/basics/AddRandomEdges1.mgl and samples/basics/AddRandomEdges2.mgl.

4.5. Observation system

4.5.1. What is observation ?

By observation we want to express the idea that some objects are able to change automatically their state depending on other objects states (at least this is my understanding and my way to express it). For instancesuppose we have a VertexSet N and an EdgeSet E based upon it. Some user may wish to remove a Vertex v from N. We would then expect that E doesn't contain any edge related to v anymore, without any intervention from the user. This is exactly what happens with Mascopt, and that behaviour is based on what we call "observation".

4.5.2. The general principles

Broadly the observation mechanism is implemented in a very simple way. The observed object must extends the ObservableObject class (which mascoptObject extends) and call the right method (generally called notify(Add/Remove/Value)Observers) when something happens. Besides the observer object must implement the Observer interface, that is mainly implements an update method. Moreover, the observer object is responsible for its adding and removing from the the list of observers of the related object. This is done by calling the appropriate methods of the observed object. Those methods are usually named add(Add/Remove/Value)Observer and remove(Add/Remove/Value)Observer.

4.6. MGL/MGX formats

4.6.1. For version 1.2.x or older

4.6.1.1. MGL Format

Mascopt uses XML to describe graphs and other network objects rather than simple text format.

Mascopt provides several input and output formats. We only present MGL and MGX which means Mascopt Graph Library and Mascopt Graph Extended. MGL format is the Mascopt's native format and is based on the XML standard. It provides a readable description of Mascopt's objects which can be easily extended for the user's own objects or specializations: as the MGL reader parse an XML file, it can also read a derived file, containing new tags.

One can create his own format implementing the interface WriterInterface which consist only in two methods: an add method to add objects to write and a write method to physically write the file.

The file is quite understandable. The information contained in a MGL file reflects the oriented-object structure of Mascopt. This is a simple manner to enable the share of objects. The listing bellow presents a sample of a code for a digraph.

```
<?xml version="1.0" ?>
<!DOCTYPE OBJECTS SYSTEM
"ftp://ftp-sop.inria.fr/mascotte/mascopt/dtd/mgl_v1.1.dtd">

<OBJECTS>
 <VERTICES>
  <VERTEX id="N0">
   <POSITION>
    <X>50.0</X> <Y>0.0</Y>
   </POSITION>
   <VALUE type="function" dataType="String"> node0 </VALUE>
  </VERTEX>
  <VERTEX id="N1">
   <POSITION>
    <X>10.0</X> <Y>50.0</Y>
   </POSITION>
  </VERTEX>
  <VERTEX id="N2">
   <POSITION>
    <X>0.0</X> <Y>0.0</Y>
   </POSITION>
  </VERTEX>
 </VERTICES>

 <LINKS>
  <ARC id="AE1">
   <VERTEX_REF idref="N1"/>
   <VERTEX_REF idref="N2"/>
  </ARC>
  <ARC id="AE0">
   <VERTEX_REF idref="N0"/>
   <VERTEX_REF idref="N2"/>
   <VALUE type="Capacity" dataType="Double"> 6.8 </VALUE>
   <VALUE type="length" dataType="Integer"> 110 </VALUE>
  </ARC>
  <ARC id="AE2">
```

```
   <VERTEX_REF idref="N0"/>
   <VERTEX_REF idref="N1"/>
  </ARC>
 </LINKS>

 <SETS>
  <VERTEX_SET id="NS0">
   <VERTEX_REF idref="N0"/>
   <VERTEX_REF idref="N1"/>
   <VERTEX_REF idref="N2"/>
  </VERTEX_SET>
  <ARC_SET id="AES0">
   <ARC_REF idref="AE0"/>
   <ARC_REF idref="AE1"/>
  </ARC_SET>
  <ARC_SET id="AES3">
   <ARC_REF idref="AE0"/>
   <ARC_REF idref="AE2"/>
  </ARC_SET>
 </SETS>

 <GRAPHS>
  <DIGRAPH id="G0">
   <VERTEX_SET_REF idref="NS0"/>
   <ARC_SET_REF idref="AES0"/>
  </DIGRAPH>
  <DIGRAPH id="G3">
   <VERTEX_SET_REF idref="NS0"/>
   <ARC_SET_REF idref="AES3"/>
  </DIGRAPH>
 </GRAPHS>
</OBJECTS>
```

4.6.1.2. MGX Format

MGX is an extended version of MGL. It solve the following problem: as all objects are shared between graphs, the valuation of an object may differ between a graph and another graph. For example, if an arc a0 is valued with Capacity=6.8, then this value is the same in all graphs containing this edge. As a result, we introduce the notion of context, which precise the context where the value is valid. The simplest way of using context is to put the graph itself as context. It express directly that a value is valid only in this graph.

With two graphs G0 and G3 we can now add contexted values named "Capacity" on arc a0. Then, the differences between MGL and MGX files are shown in the next listing. Note that the default value 6.8 is kept on a0: without context, the valuation system gives this value.

```

a0.setDoubleValue("Capacity",G0, new Double(1.0));
a0.setDoubleValue("Capacity",G3, new Double(3.2));

<ARC id="AE0">
 <VERTEX_REF idref="N0"/>
 <VERTEX_REF idref="N2"/>
 <VALUE type="Capacity" dataType="Double"> 6.8 </VALUE>
 <VALUE type="Capacity" dataType="Double" context="G0"> 1.0 </VALUE>
 <VALUE type="Capacity" dataType="Double" context="G3"> 3.2 </VALUE>
</ARC>
```

4.6.2. For version 1.3.x and later

4.6.2.1. What changed ?

We totally replaced the parser by a DOM Parser, in ordre to clear the code. It is slower but saffer than before. We validate the document two times: one time with a laxist DTD, then with a Relax NG grammar. With a relax NG grammar, we can express more constraints for the file and the validation of the xml file is improved.

4.6.2.2. DTD and Relax NG 1.4

The Validation of files is controlled by the version 1.4 of the DTD and relax NG. Have a look in this directory.

Now the files looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE OBJECTS PUBLIC "SYSTEM"
"ftp://ftp-sop.inria.fr/mascotte/mascopt/dtd/mgl_v1.4.dtd">
<OBJECTS
version="ftp://ftp-sop.inria.fr/mascotte/mascopt/dtd/mgl_v1.4.rng">
  <GRAPHS>
    <DIGRAPH id="G0">
      <VERTEX_SET_REF idref="NS0"/>
      <ARC_SET_REF idref="AES0"/>
    </DIGRAPH>
  </GRAPHS>
  <SETS>
    <VERTEX_SET id="NS0">
      <VERTEX_REF idref="V0"/>
      <VERTEX_REF idref="V1"/>
      <VERTEX_REF idref="V2"/>
    </VERTEX_SET>
    <ARC_SET id="AES0">
      <ARC_REF idref="AE1"/>
      <ARC_REF idref="AE0"/>
      <VERTEX_SET_REF idref="NS1"/>
    </ARC_SET>
    <VERTEX_SET id="NS1">
      <VERTEX_REF idref="V2"/>
      <VERTEX_REF idref="V0"/>
      <VERTEX_REF idref="V1"/>
    </VERTEX_SET>
  </SETS>
  <VERTICES>
    <VERTEX id="V0">
      <POSITION>
        <X>50.0</X>
        <Y>0.0</Y>
      </POSITION>
      <VALUE dataType="String" type="function">node0</VALUE>
    </VERTEX>
    <VERTEX id="V1">
      <POSITION>
        <X>10.0</X>
        <Y>50.0</Y>
      </POSITION>
    </VERTEX>
    <VERTEX id="V2">
      <POSITION>
        <X>0.0</X>
        <Y>0.0</Y>
      </POSITION>
    </VERTEX>
  </VERTICES>
  <LINKS>
    <ARC id="AE1">
      <VALUE dataType="Double" type="Capacity">6.8</VALUE>
      <VALUE dataType="Integer" type="length">110</VALUE>
      <VERTEX_REF idref="V0"/>
      <VERTEX_REF idref="V2"/>
    </ARC>
    <ARC id="AE0">
      <VERTEX_REF idref="V1"/>
      <VERTEX_REF idref="V2"/>
    </ARC>
  </LINKS>
</OBJECTS>
```

4.7. The Graphical User Interface

4.7.1. Mascopt Viewer

The Viewer is quite complicated to use because it permits to view a large amount of graphs. In this part, we don't describe how to use the viewer by the code. We only describe the user interface and how to use, with your mouse, the viewer.

4.7.2. Launch the editor

```
java GUI
```

or

```
/net/home/jflaland/bin/graphViewer
```

4.7.3. Use of the Viewer

The viewer is composed of the main window, wich allow to control what is viewed in the views. Then you can create other windows wich are called "Views" because offering different views on several graphs. Each view can display different Layers at the same time. And each layer contains a list of graphs. With this system you can display in one view a layer containg for example the request graph and a network and, in an other view, the network and several paths on it. As the viewer is quite flexible it explains why it is not simple to use.

4.7.3.1. Managing Views and Layers.

Multiple views can be opened clicking on "New View" button. When a view is created, a new tab View #n is added in the Views section. In each View, you have at the bottom right the list of Layers in it. At the beginning, the new view is empty: you have to add a new layer in it. You can also add a layer which is already used in other view (this is the notion of shared layer). Note also that a Layer can be visible or hidden in a view. To remove a layer from a view, you can click on the X button at the top right.

4.7.3.2. Managing graphs in Layers

When you have created your new layers in views, then you can switch to the Layers Tab to view all the available layers. In each layer you can add some graphs, digraph, chains or paths. As you want to put in your layer some graphs from files, first you must scan a file. Select this option in the drop down list called "Add Graph", then select a file and all graphs contained in this file will be added in the drop down list. To add a graph, you just select the layer where you want to put it and then, you select the wanted graph in the drop down list.

When the graph is added in the layer, it appears in the list and in all the Views which contains the concerned layer. You can also hide a a graph clicking in the checkbox on the right. To remove a graph from a layer you can click on the X button at the top right.

4.7.3.3. The tab editor in the viewer

We added an editor tab in the viewer because the user may have to change a graph in a file. It works as describe below.

4.7.4. Mascopt Editor

This is a graphic interface which allow to allow a graph, change values on edges and vertices.

4.7.4.1. Launch the editor

```
java EDITOR
```

or

```
/net/home/jflaland/bin/graphEditor
```

4.7.4.2. Use of the editor

The editor is composed of two things: the main window with a "tool bar" to change the different uses of the mouse and a view of the current graph wich id edited.

Action buttons

- New Graph: creates a new empty graph
- New Di Graph: creates a new empty digraph
- Load: pop up a file browser. Select a file from where to load a graph (erase the current graph)
- Save: save in mgl format the current edited graph
- New View: create a new view (a new window) on the graph
- Insert Graph: load a graph from a file (not erasing the current graph)
- Dump: dump the view of the graph in an image file (PNG format)

Mouse behavior

- Create: clicking on the view, it creates a vertex. Clicking on vertices, it creates edges.
- X: clicking on vertices or edges delete it.
- Set Name: clicking on vertices or edges pop up a text field to change the name of the object.
- Set Color: clicking on vertices or edges change its color.
- Set Value: clicking on vertices or edges pop up a dialog box to change any value on objects. Refer to the javadoc concerning vertices and edges and the notion of values. You can also have a look at Graphs

Labels

The labels depends directly of the values set on objects. When you change the value on a vertex or an edge, nothing seems to appear. If you want to see it, you have to configure the labels. You have two main buttons: Set Vertex Label and Set Arc Label. When you click on it, a pop up appears with a textfield where you can put a string. This string will appear on all vertices or edges. But you can also use variables wich will read the value on each objects. The syntax is quite simple: to access a value named for example "myvalue", you should write $(value).

Example:

Imagine you have put a value name "weight" with value 12 on the first vertex and also on an another with value 10. Then, you click on Set Vertex Label and you enter the string "The weight of this vertex is $(weight)". You obtain this:

The View

The window containing a view of the graph is simple to manipulate. When you click you perform one the Mouse behavior. You can also drag and drop vertices, and drag and drop all the graph (clicking anywhere). With the whell of your mouse, you can modify the zoom of the view.

5. Network guide

5.1. Generate network files

5.1.1. Where are these files ?

The classical network files we use are stored in the directory "files". You can find here 'R1m.mgl", "EU.mgl", 'nsfnet.mgl'. This files use the MGL format. It means that networks are represented by graphs because only graphs can be stored in a MGL file. Actually, those networks uses two graphs: the request graph and the cable graph. These two graphs share their vertex set. As these files contain two graphs, it is not easy to use the Editor to create some sample files. The goal of this page is to explain how to create some networks file using two possibilities: a Net file or an GOd file.

5.1.2. GOd format

The GOd format was developped by David Coudert and is used in by other programs not included in Mascopt.

The GOd format uses two files:

- e .graph file, which describes the graph
- The .od fil, which describes the requests

5.1.2.1. The .graph files

The .graph lines can contain comments. The lines of comments begins with "c". These lines are ignored when parsing. For example:

```
c
c NSFET, 14 noeuds et 21 aretes, symetrique
```

```
c
c donnees extraites de:
c Practical routing and wavelength assignment algorithms for
c    all optical networks with limited wavelength conversion
c M.D. Swaminathan,  K.N. Sivarajan
c IEEE ICC 2002
c
c
```

The first used line of the .graph file contains "n" and the number of nodes:

```
n 14
```

Then, the file must contains 14 lines with the flag "e" which means that this is an edge, then the number of the source node and the number of destination node. You can leave some lines empty. For example:

```
e 0 1
e 0 2
e 0 7
e 1 2
e 1 3
e 2 5
e 3 4
e 3 10
e 4 5
e 4 6
e 5 9
e 5 13
e 6 7
e 7 8
e 8 9
e 8 11
e 8 12
e 10 11
e 10 12
e 11 13
e 12 13
```

Note that the nodes of the network are numbered from 0 to n-1, if we consider n nodes. You can find the complete file in mascoptDev/files/nsfnet.graph.

5.1.2.2. .od file

The .od file contains the request on the graph. The use of two files allow to change the requests using the same network. As in the .graph file, you can put some comments with the flag "c":

```
c
c Instance pour
c NSFET, 14 noeuds et 21 aretes, symetrique
```

```
c
c 268 requetes au total
c
c donnees extraites de:
c Practical routing and wavelength assignment algorithms for
c    all optical networks with limited wavelength conversion
c M.D. Swaminathan,  K.N. Sivarajan
c IEEE ICC 2002
c
c
```

Then, you have just to declare the requests with the flag r: r source_node dest_node request_size. For example:

```
r 3 11 2
r 3 12 1
r 3 13 3
r 4 0 1
r 4 1 3
r 4 3 2
r 4 5 1
r 4 7 2
...
```

You can find the complete file in mascoptDev/files/nsfnet.od.

5.1.3. Net format

The Net format is dedicated to WDM networks. It has been extended to describe any network with hierarchical structure.

You can define some comments at the beginning of the file with "//". For example:

```
// ---------------------------------------------------------------------//
// Rosalie
//
// Simple nework in "R" design.
//
// All edges and requests exprimed in this file. Do not use the -mirrorEdge
//  nor -mirrorRequest options to translate this net in xml file.
// ---------------------------------------------------------------------//
//
//
```

You can also, after a line containing some data, add some comments with "//".

The first line containing data must give the number of levels (in a WDM network, 4 levels). Then, the next lines contain the names of each level (two times) and the number of sub-levels in this level. Here an example:

```
4 // 4 levels
```

```
CABLE  CABLE  0  // Main tag
FIBER  FIBER  4  // 4 bands in a fiber
BAND   BAND  8   // 8 lambdas in a band
LAMBDA LAMBDA  0 // Last tag
```

Then the data starts. The next line gives the number of nodes and the network and the following lines give the name of each nodes and its coordinates:

```

6
Zero   200   100
Un     400   100
Deux   200   300
Trois  400   300
Quatre 200   500
Cinq   400   500
```

Then the description of the links starts. Again, the first lines gives the number of requests, then the following lines give the description of each ling with "node1 node2 cable_size cable_lenght". For example:

```

11
0 1 1 100
2 0 2 100
0 2 2 100
3 0 1 141
1 3 2 100
3 1 2 100
3 2 1 100
2 4 2 100
4 2 2 100
2 5 1 141
5 2 1 141
```

5.1.4. How to read these files in Mascopt ?

We implemented the classes wich read these two formats. You can find these classes in the package mascoptLib.io.graph. These classes works as the MGLReader works (it's quite the same).

5.1.5. How to convert the GOd format to an MGL file ?

From the reader classes, we implemented some small converters (that's easy to program).

5.1.5.1. od2mgl

You can find this program in mascoptDev/bin. You can also call it using:

```

java Od2MglLauncher
usage: od2mgl fileIn.graph fileIn.od fileOut.mgl [-h] [--help]
```

```
Try `od2mgl --help' for more options.
```

This converter takes the .graph and .od files in input, and then writes the mgl file.

5.1.5.2. net2mgl

You can find this program in mascoptDev/bin. You can also call it using:

```
java Net2MglLauncher
usage: net2mgl fileIn.net fileOut.mgl [-nma] [--noMirrorArc] [-nmr]
[--noMirrorRequest] [-h] [--help]
Try `net2mgl --help' for more options.
```

This converter takes the .net file and writes the mgl file. You have some optional flags: noMirrorArc does not duplicates the arcs of the networks (which is done by default). The noMirrorRequest does not duplicates the requests (which is done by default).

6. Tutorials

6.1. Prim's algorithm

6.1.1. How to program Prim's algorithm for the Minimum Spanning Tree

One of mascopt expected use is the quick implementation of algorithm and more partcularly algorithms on graphs. This note is intended to help you in the realization of this goal by showing you how to manipulate the differents data structures involved. The followed approach is to show how one could program the Prim's algorithm that computes the Minimum Spanning Tree (MST for short) of a given graph. A formulation and a proof of this algorithm can be found in textbooks or even on the internet.

6.1.2. Where we go

We are now programming a very simple application that takes a graph, computes its MST taking into account the "length" of the edges and then outputs the resulting graph. All the GUI stuff is left as an exercise to the reader or perhaps as a subject for a forthcoming article in this documentation.

It is to be remarked that with a little bit of work mascopt enables us to implement an application accepting as input either a graph or a digraph and let the user choose the type of value used to compute the MST. This is what the PrimMST class does.

6.1.3. At least, the code

It's a java program, so first we need to declare the class (we can't do the imports now, because we don't know exactly what we'll need), let's call it MSTByPrim:

```
public class MSTByPrim
{
```

As the application is very basic, we won't need any global variable and no constructor. The default one is sufficient. So let's code the method that will do most the work (the aim of this article is not to instruct on how to do some beautiful oop but how to use mascopt - even if the latter does not exclude the former, but in this particular case the algorithm is very simple and straightforward).

```
 public Graph computeMST(Graph inputGraph)
    {
```

We are going to follow the Prim's algorithm and construct step by step the graph made of the MST, that is we explore the graph given as an input until we find the right edge and then we add that edge to the set of edges of the MST and its missing extremity to the set of vertices. In order to do that we will need a graph to store the MST and a set of already touched vertices. We introduce those variables and initialize them.

```
   VertexSet touchedVertices = new VertexSet();
   EdgeSet keptEdges = new EdgeSet(touchedVertices);
   Iterator itGN = inputGraph.getVertexSet().iterator();
   if (itGN.hasNext())
       {
      touchedVertices.add((Vertex)itGN.next());
       }
```

It has to be noted that in mascopt, you traverse linearly a set using an Iterator. Two methods of those objects are mainly used, hasnext() which test if there remains an untouched element in the set and next() which extracts the next element of the set. The drawback of this method is the fact that the returned object is of type Object and we are then obliged to explicitely cast it. Advantages include the standardization of the traverse of sets and the fact that if the set is changed, the object of type Iterator is automatically set to null.

```
   while (touchedVertices.size() < inputGraph.getVertexSet().size())
       {
      Iterator itTN = touchedVertices.iterator();
      int minLength = Integer.MAX_VALUE;
      Edge bestCandidate = null;
      Vertex neighborToAdd = null;
```

This is the main loop, we try until the MST we are constructing contains as many vertices as the initial graph. And we initialize working variables. itTN is the iterator used to parse the set of vertices of the MST. minLength is the smallest length of an outgoing edge find so far, bestCandidate is the corresponding edge and neighborToAdd is the extremity of that edge not yet belonging the MST.

```
      while (itTN.hasNext())
          {
         Vertex current = (Vertex)itTN.next();
         VertexSet neighbors = (VertexSet)current.getNeighbors(inputGraph);
         Iterator itN = neighbors.iterator();
```

Here, for every edge of the MST, we test the outgoing edges. In fact we consider the list of neighbors of a vertex which is exactly the same. It is only a little bit shorter in the following as we need to test wether the neighbor is already in the MST.

```
while (itN.hasNext())
             {
            Vertex currentNeighbor = (Vertex)itN.next();
            // we are only interested in outgoing edges
            if (!touchedVertices.contains(currentNeighbor))
                 {
               EdgeSet potentialMSTEdges = (EdgeSet)
current.getEdgesTo(inputGraph,currentNeighbor);
               Iterator itpMSTe = potentialMSTEdges.iterator();
               Edge potentialEdge = null;
               // try every edge in the edgeset
               while (itpMSTe.hasNext())
                    {
                  potentialEdge = (Edge)itpMSTe.next();
                  // if the current edge is the best up to now
                  if (Integer.parseInt(potentialEdge.getValue("length"))<
minLength)
                       {
                     minLength =
Integer.parseInt(potentialEdge.getValue("length"));
                     bestCandidate = potentialEdge;
                     neighborToAdd = currentNeighbor;
                       }
                    }
                 }
             }
```

If the neighbor doesn't yet belong to the MST, we consider the edges between the two vertices and in particular its length compared to the length of the already seen edges. This is Prim's algorithm.

```
          }
      touchedVertices.add(neighborToAdd);
      keptEdges.add(bestCandidate);
       }
```

After having explored all the vertices of the soon-to-be MST and all the outgoing edges, we add the one that fits.

```
   return new Graph(touchedVertices,keptEdges);
    }
```

```
}
```

Finally we return the graph of the MST.

It is possible to transcribe algorithms in mascopt in a quite simple way. In spite of the intensive use of a great number of variables the program is still quite efficient because we are not creating any variable (a great loss of time in java) but only pointing to already existing variables.

The use of iterators to describe the different sets is quite interesting because it standadize the writing of loops and we don't need to pay any more attention to the size of the different structures.

6.1.4. Sample

The above program can be found in Mascopt Dev with a test program samples/algos/MSTByPrimGUI.java using this function and a file containing a graph whose MST can be computed with this program (file samples/algos/MSTGrapheExample.mgl). You can of course use your own graph, but in order to keep the example rather simple, we only considered connected graphs with edges that have an associated value called "length".

To test it, just do:

```
java MSTByPrimGUI samples/algos/MSTGrapheExample.mgl
```

6.2. TD d'initiation

6.2.1. Questions

1. Installez mascopt sur votre compte en suivant les instructions de la page Getting Mascopt puis récupérez sur votre compte le fichier grapheBicolore.mgl.
2. Écrire un programme permettant de choisir de manière graphique un fichier avec une extension .mgl et de stocker le graphe de ce fichier dans une variable de type Graph. Pour cela on utilisera la classe GraphChooser et la méthode getGraphMGL (il faut remarquer que cette méthode est utilisable ici car le fichier ne contient qu'un graphe).
3. Modifier le programme précédent de manière à construire un sous-graphe du précédent ne contenant que les sommets dont l'attribut couleur vaut ``rouge". Puis un nouveau graphe contenant les sommets dont l'attribut couleur vaut ``vert".
4. Modifier à nouveau le programme de façon à afficher les 2 graphes créés précédemment. Facultatif : on pourra afficher les sommets avec l'attribut couleur valant ``rouge" (resp. ``vert") en rouge (resp. vert).
5. Enfin, modifier le programme afin de sauver d'abord chaque graphe dans un fichier séparé puis tous dans un même fichier.

6.2.2. Appendice A : comment afficher des graphes, une méthode

Il existe dans mascopt une classe permettant l'affichage des graphes, MascoptViewer. Pour afficher un graphe, il faut donc créer un objet de type MascoptViewer. Cette objet va alors contenir une liste de vues (qui correspondent à différentes fenêtres qui seront créées) et chaque vue contient une liste de ``layers'' contenant chacun un ou plusieurs graphes. Cette organisation permet en particulier de superposer différents graphes en ajoutant leur layers respectives dans les vues. Il est aussi possible d'ajouter la même layer à différentes vues. Les vues sont implémentées par la classe GView et les layers par la classe GLayer.

En pratique on peut procéder de la manière suivante,

- on instancie un objet de type MascoptViewer
  ```
  MascoptViewer mv = new
  MascoptViewer();
  ```
- on instancie un objet de type GView
  ```

  GView gv = mv.newViel(G.getName());
  ```
- on instancie un objet de type GLayer
  ```

  GLayer gl = mv.newLayer(G.getName());
  ```
- on associe le graphe et le layer
  ```

  mv.addGraphInLayer(G,gl,true);
  ```
- on ajoute la layer dans la vue
  ```

  mv.addLayerInView(gl,gv);
  ```

Et tout devrait s'afficher de la manière souhaitée.

6.2.3. Appendice B : comment colorier des éléments.

Pour colorier les sommets et arêtes dans le mascoptViewer, il suffit de créer un attribut ``color'' dans l'élément à colorier et de lui donner comme valeur l'entier correspondant à la couleur désirée.

Ainsi, supposons que l' n ainoeque l'on soe voir apparaîeen rouge, un façon d'aboutir à ce résultat est d'utiliser les méthodes de la classe java java.awt.Color :

```

N1.setIntValue("color",(Color.red).getRGB());
```

6.2.4. Solution

Voici la solution de l'exercice, programmée par Marc Martinez: Premier.java