UNIVERSITE LIBRE DE BRUXELLES
Faculty of Engineering
Department of Computer & Network Engineering


# The Visual BioMaze Framework 1.2.1 Tutorial

Tutorial Version 2.0, June 2005
Designed to be used with Eclipse 3.1

Skhiri dit Gabouje Sabri
sskhirid@ulb.ac.be

# Contents

# Chapter 1

# Introduction

Extracting and visualizing information from biochemical databases is one of the most important challenges in biochemical research. The huge quantity and high complexity of the data available force the biologist to use sophisticated tools for extracting and interpreting accurately the information extracted from the database. These tools must define a graphical semantics associated to the data semantics in accordance with biologist usages. The aim of the Visual BioMaze Framework is to display complex biochemical networks in a readable and understandable way. In this tutorial we define the notion of *customizable representation model*, which allows the biologist to change the graphical semantics associated to the data semantics. The approach is also *generic* since our graphical semantics is common to several kinds of biochemical networks. we also provide *interchangeable graph layout* algorithms, giving the user the possibility to choose an existing algorithm, or to define his own. We will see that Visual BioMaze can be used for representing any kind of graphs with a particular semantics. We explain how these notions can be applied in the BioMaze project[1]. Finally, a more technical part presents the implemented graph layout algorithms and their adaptation to biochemical networks. In this document we call basic *contribution* either the representation model or the set of graphical constraint or the graph layout algorithm. We will define all these notions.

## 1.1 The BioMaze - Visual BioMaze Project

### 1.1.1 BioMaze

A major challenge of the post-genomic era is to determine the functions of all the genes and gene products at the genome level. In order to improve the prediction of such functions it is important to take in account the information about the different organization levels of the living cell. In particular, it is necessary to consider the set of physical and functional interactions between genes and proteins. Such interactions form networks of cellular processes, called biochemical networks, which include metabolic networks, regulatory networks for gene expression, and signal transduction.

The huge quantity of data already available and its continuous growth, the need to integrate such information, as well as the necessity of sophisticated software tools for manipulat-

[1]The partners of the Biomaze project (`http://cs.ulb.ac.be/research/biomaze/vbm`) are Université Libre de Bruxelles, Université Catholique de Louvain-la-neuve and Facultés Universitaires Notre-Dame de la Paix de Namur.

ing it represent true challenges for the research in bioinformatics. New tools for integrating, querying, extracting, analyzing, and visualizing biochemical databases are essential for the pharmaceutical and biotechnology industry, in particular for the design of new drugs and vaccines. Such highly sophisticated tools must be designed by multi-disciplinary teams, and require recent results in computer science in areas such as operational research (graph algorithms, constraint logic programming, automatic learning, form recognition, etc.), databases (huge schema management, object-oriented interfaces, evolution, meta-data, etc.), and visualization (multi-resolution, multi-representation, complexity management, etc.).

The aim of the BioMaze project is to develop a set of tools including:

1. an information system allowing to represent information about biochemical networks, and including functions for evolution management, generation, and documentation;

2. an open system of specialized software components to exploit biochemical data, including extraction, analysis, navigation, and visualization; and

3. a Web interface given access to the services providing by those specialized components.

The underlying information system of BioMaze is aMAZE [14].

### 1.1.2 Visual BioMaze

Visual BioMaze (VBM) [28] is the visualization framework of the BioMaze project. Although it was developed under BioMaze, the visualization framework can be used separately. The aim of the Visual BioMaze project is to provide a generic graph visualization tool, i.e., a framework able to show any kind of graphs, having any kind of associated graphical representation and any kind of graph layout algorithms.

When a graph has to be shown, the visualization framework has to use two important parameters: the representation model and the graph layout algorithm. We call representation model the graphical semantics used to represent the data embedded in graph, i.e., the graphics associated to each type of node and each type of arc. The graph layout algorithms will compute the position of each node of the graph. A tool which aims to provide generic viewer of graphs has to respond to two fundamental specifications: it has to be able to associate each type of node with a specific graphical representation and it has to be able to load dynamically graph layout algorithm. The Visual BioMaze framework fulfills these two requirements.

As a consequence, the Visual BioMaze tool can cope with any kind of graph independently of the its semantics and the user can choose both a suited representation model and a suited graph layout algorithm. With respect to the BioMaze project, this means that VBM can cope with any kind of biochemical graphs and it is able to represent them in the same drawing. An example of a network composed of a signal transduction part and a metabolic part arrives when a liver cell receives the signal that the sugar rate increases in the blood (transduction) and its response in which the insulin acts (metabolic pathway). Since users write a query in BioMaze which retrieves such a graph, the VBM framework is able to represent these two kind of biochemical networks in the same drawing. Such a result is not available yet in other biochemical tools.

# Chapter 2

# Installing the Visual BioMaze Framework

Visual BioMaze has been implemented as an Eclipse [17] feature. Eclipse is a rich client platform for the development of highly integrated tools. This means that you must first install Eclipse and then update the platefrom with our tool. Visual BioMaze needs an Eclipse the version 3.0 or higher.

Although you can use the Visual BioMaze Framework as any graph viewer, this version 1.X.X needs the installation of the aMaze[14] [27] [26] project. You can see how to install it on `http://www.northbears.org/thesnowbook` and the aMaze web site on `http://www.amaze.ulb.ac.be` for more information.

## 2.1 Eclipse Installation

1. Go to the Eclipse web site: `http://www.eclipse.org`
2. Enter the **Downloads** section
3. Select the site: **North America Main Eclipse Download Site**
4. Select the build: Eclipse-SDK-**3.1.X**, Attention, the Visual BioMaze framework has been tested fully compliant with the Eclipse 3.1 or higher but not for lower version
5. Follow the installation procedure
   (a) Check the availability of Java 1.4.1 (or higher) on your machine
   (b) Download the Eclipse SDK on your platform (Windows, Linux, Mac OS X)
   (c) Open the archive and install the application

## 2.2 GEF Installation

The Visual BioMaze needs the installation of the Graphical Editing Framework (GEF).

1. Start the Eclipse WorkBench

2. In the menu bar, select **Help→Software Updates→Find and Install** to launch the Install Wizard
3. On the features Updates Page
   (a) Check Search for new features to install
   (b) Press next
4. On the **Update sites to visit** page
   - Check the *Eclipse.org update site* to include it in the search
   - Select GEF-SDK 3.0.1 or higher in the list contained by *Eclipse.org update site*
   - Press next

Restart Eclipse and verify that everything is installed. To do so, select **Help→About Eclipse platform** and press the **Plug-in details**. Check the following entries:

- Eclipse.org Draw2D
- Eclipse.org Draw2D Documentation
- Eclipse.org Graphical Editing Framework
- Eclipse.org Graphical Editing Framework Documentation
- Eclipse.org Graphical Editing Framework SDK

## 2.3 Visual BioMaze Installation

1. Start the Eclipse WorkBench
2. In the menu bar, select **Help→Software Updates→Find and Install** to launch the Install Wizard
3. On the features Updates Page
   (a) Check Search for new features to install
   (b) Press next
4. On the **Update sites to visit** page
   (a) Add the *Laboratory of Computer and Network Engineering* update site
      - Press **New Remote site**
      - Fill **Name**:*CS Update site*
      - Fill **URL**:*http://cs.ulb.ac.be/research/biomaze/update*
      - Press OK
      - Check the *CS Update site* to include it in the search

For checking if the installation succeeded, restart Eclipse and see in **Help→Software Updates→Manage Configuration**. Verify if *Visual BioMaze Feature x.x.x* is present (see Figure 2.1).

***Note:*** Visual BioMaze needs the presence of the Snow plug-in, users can find any information about the installation of Snow on `http://www.northbears.org/thesnowbook`.

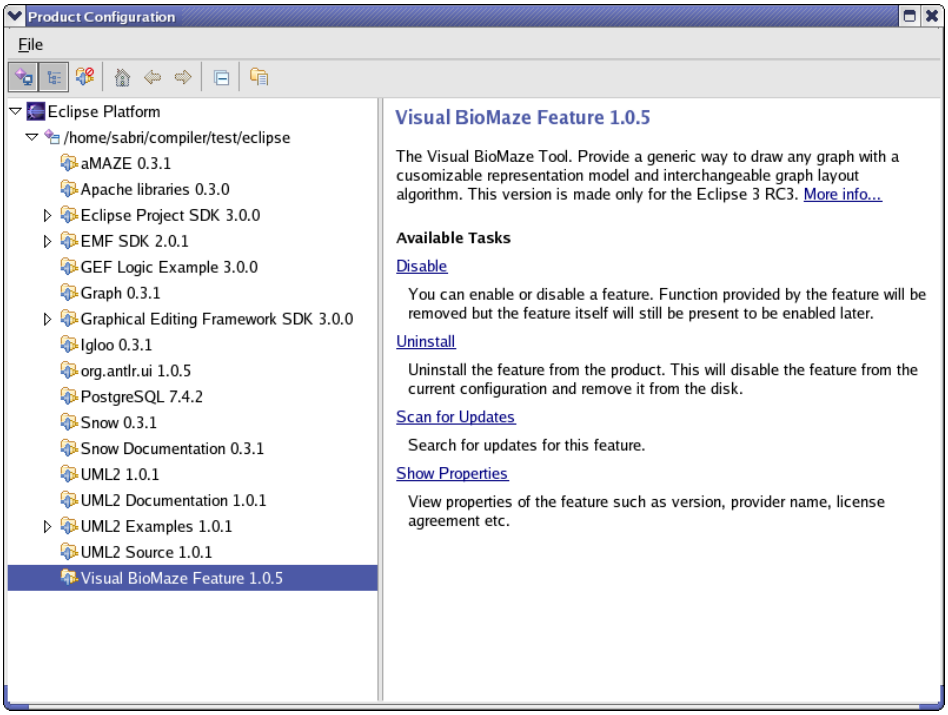



Figure 2.1: The Visual BioMaze Feature is present.

# Chapter 3

# Using Visual BioMaze

## 3.1 Open The Visual BioMaze

Once the plug-in is installed, you can open the Visual BioMaze View by:
**Window → Show View→Other→Visual BioMaze Views → Visual BioMaze x.x.x**
The display zone appears, see Figure 3.1.

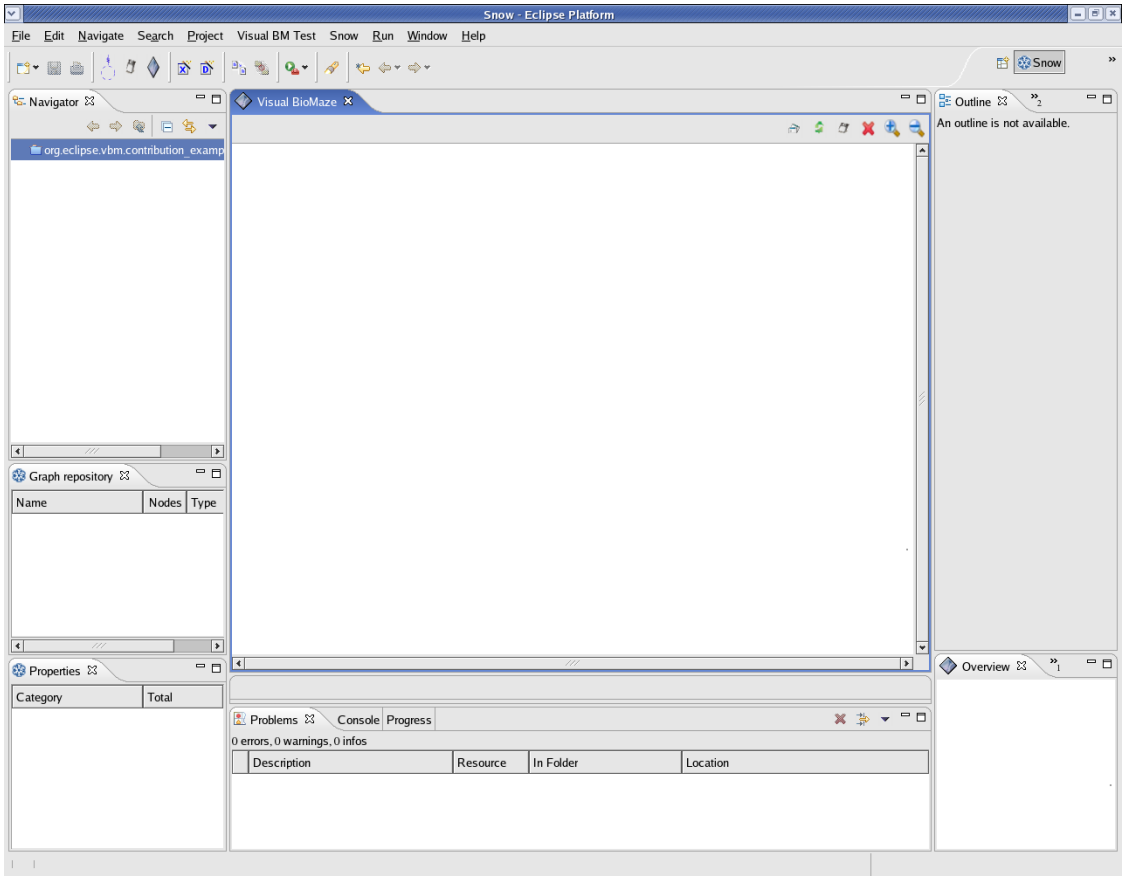



Figure 3.1: The Visual BioMaze drawing area.

## 3.2 Visualizing a result

If you want to visualize the result of an IGLOO query [1], the only thing to do is executing the query. If the Visual BioMaze view is opened the result will automatically be drawn.

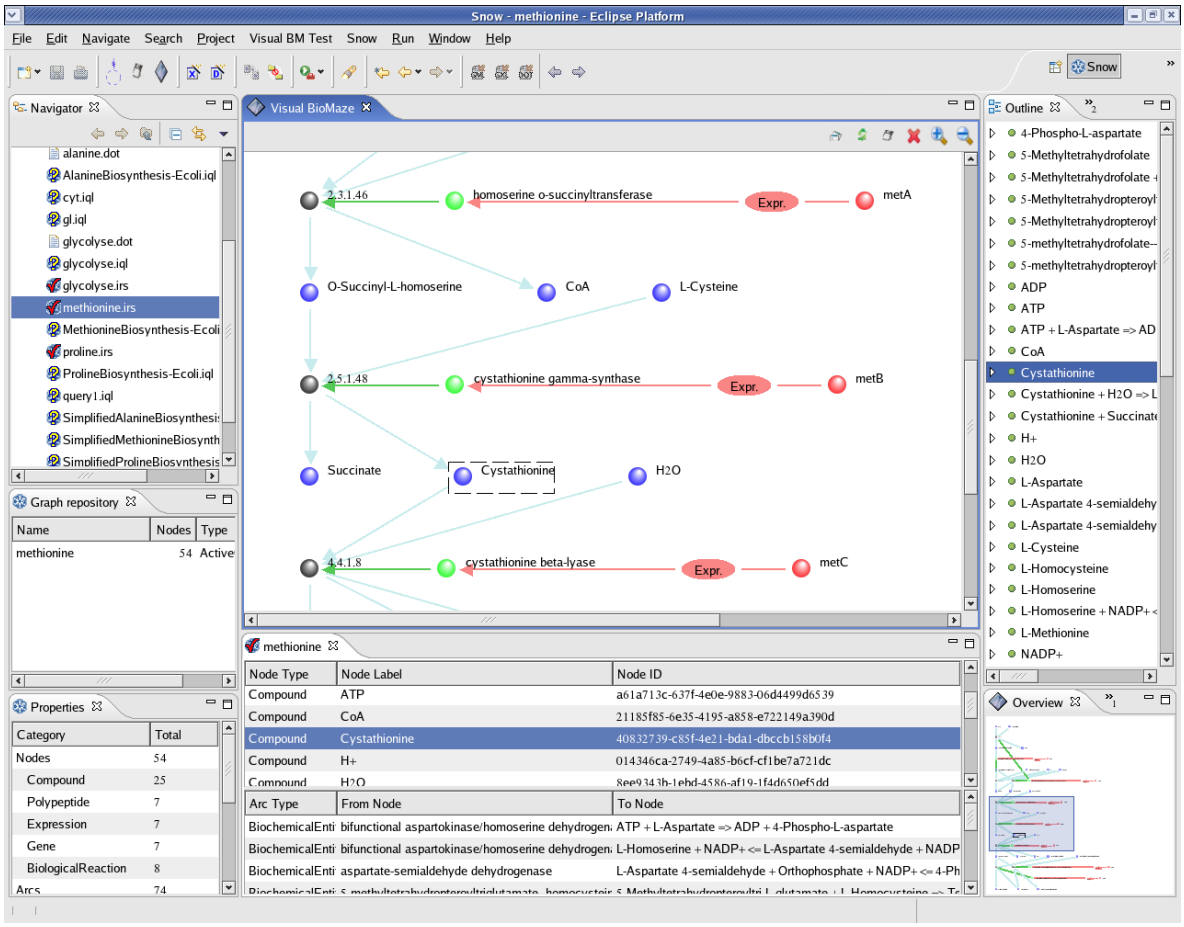



Figure 3.2: The snap shot shows how you can visualize a IGLOO result in the Visual BioMaze Framework.

## 3.3 Testing Visual BioMaze

We have introduce a test graph (the methionine biosynthesis) in the Tool. You can access it by the **Visual BM Test menu**. For displaying the menu, select **Window→Customize Perspective**, and in the **Commands** tab, check the **Visual BioMaze Menu**. In this menu, select **Show methionine in view**.

[1] For any information about IGLOO or Snow, see the documentation: `http://www.northbears.org` or the aMaze web site: `http://www.amaze.ulb.ac.be`

## 3.4 Assign graph-local preferences

In the following you will learn how to set a general graph layout algorithm, a representation model and a graphical constraint. These preferences are set for all graphs that will be drawn in Visual BioMaze. However, some graphs need particular representation model or particular graph layout algorithm. Then the Visual BioMaze framework provides a way to assign local preferences, and those will not impact the other graphs. Click on the local preference assignation button shows by Figure 3.3 and set the different parameter in the wizard. If you want to keep some parameters as the current graph layout algorithm, just press `Next`.

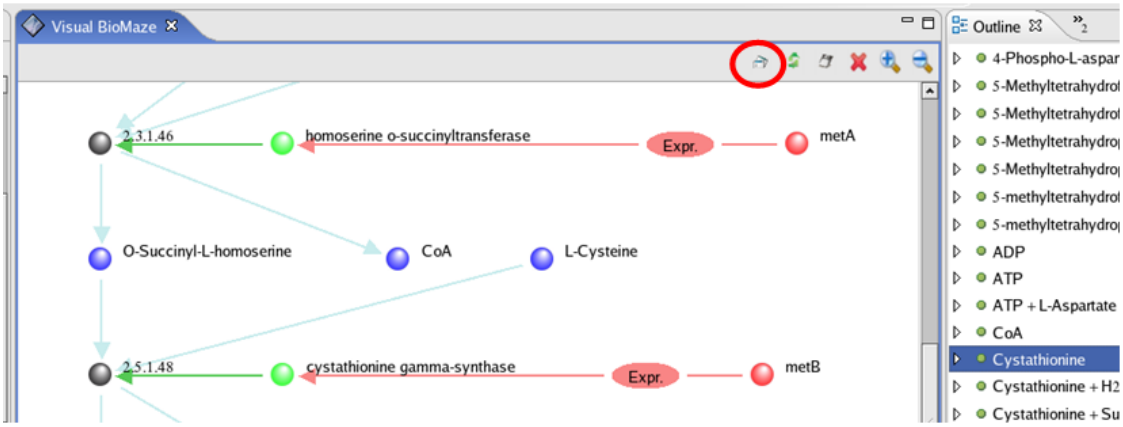



Figure 3.3: The Visual BioMaze framework provides a way to assign graph-local preferences.

Click on the button shown by Figure 3.3, the assign local preferences wizard page opens.The first page allows users to choose a suited Style. The style will be investigated in the chapter 7. Press finish for closing the wizard and applying the style on the current graph. If you want changing either a representation model, a set of graphical constraints or the graph layout algorithm, just press next.

## 3.5 The Visual BioMaze toolbar

Figure 3.4 shows the function of the different buttons of the Visual BioMaze toolbar.

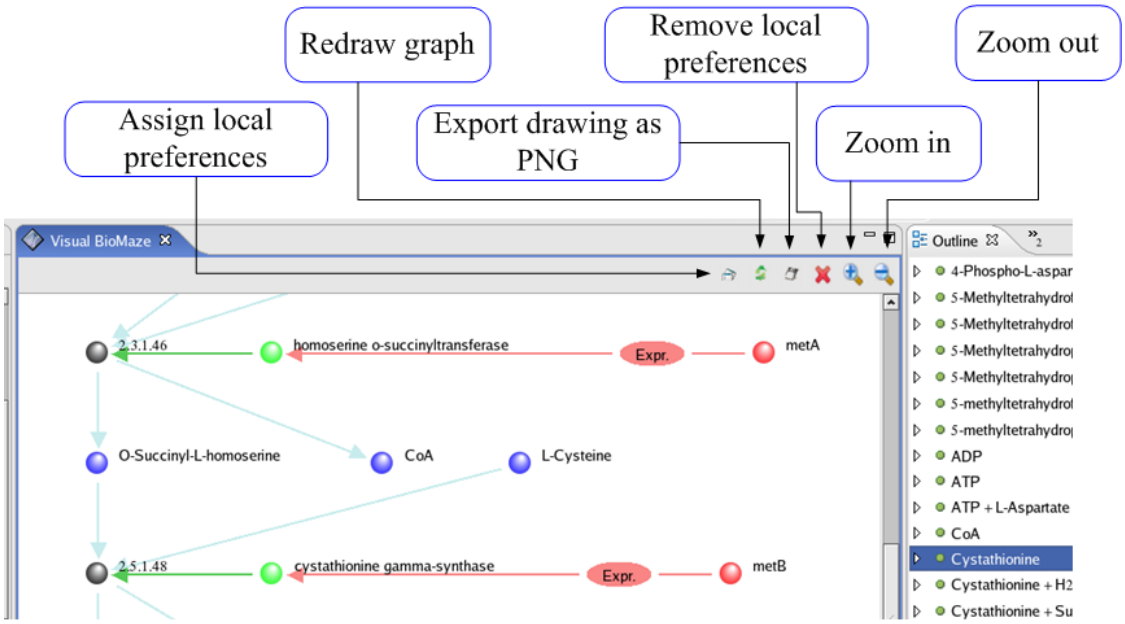



Figure 3.4: The Visual BioMaze toolbar.

# Chapter 4

# Customizable Representation Model

## 4.1 Introduction

An innovative aspect of our approach is the definition of representation models that are both *customizable* and *generic*. These two aspects are explained next. Choosing a representation model is of paramount importance because it has to convey the semantics represented by the data. Although researchers in bioinformatics do not agree about which representation model to use, two schools emerge: one represents metabolic pathways as usual in biochemistry books, as in BioCyc [11], while the second one proposes more expressive models for such pathways, like the Khon model [13], the Cook model [2] or the Maimon and Broming model [15].

The problem with simple models is that they are sometimes ambiguous: e.g., an arrow in the Kegg model is used both for representing the input/output of a reaction but also in phosphorylation (i.e., the addition of a phosphate group), or in other cellular transport interactions. On the other hand, although the expressive models convey a rich semantics, the problem is that they are difficult to understand and to read.

Therefore, each tool implements a particular representation model. In order to provide a flexible visualization tool, instead of choosing a particular model we introduced the notion of *customizable representation model*. This means that the same biochemical networks can be visualized with different representation models. The tool provides several predefined models but the users can also customize such models according to their research needs. Further, users may explore alternative representation models over the same biochemical network.

As a consequence, this approach allows two new interesting possibilities: (1) users can devise or adapt a model to highlight particular aspects in which they are interested (2) if the database schema evolves and if the information about which entities has evolved is known, a new representation model expanded with these new entities can be (semi-)automatically generated. This latter feature provides an elegant solution for the problem of visualizing an evolving database.

On the other hand, the Visual BioMaze Framework aims at representing three types of networks: metabolic pathways, signal transduction networks, and interaction networks. The three types of networks are represented as graphs, where the nodes are either biochemical entities (compound, gene, polypeptide) or interactions (reaction, control, signal). Our approach to represent the three types of graphs in an integrated visualization tool is to define a *general*

*representation model*. This model describes the graphical semantics associated to each entity and interaction node, regardless of the graph type.

As a consequence, we can cope with graphs having mixed types. An example of a network composed of a signal transduction part and a metabolic part arrives when a cell of liver receives the signal that the sugar rate increases in the blood (transduction) and its response in which the insulin acts (metabolic pathway). Since we allow typed subgraphs, the visualization module can recognize the different subgraphs and apply specific graphical constraints according to the subgraph type (coming soon in version 1.1.X).

## 4.2 Implementation

We use XML for implementing representation models. Such models are based on an XML Schema definition that prescribes the correctness of the model and provides validation features.

### 4.2.1 The model

The current representation model defines two objects types, `Element` and `Link` corresponding, respectively, to nodes and edges of biochemical graphs. An element defines a graphical semantics which can be associated to a node, while a link defines a graphical semantics which can be associated to edges between such elements. The code below, extracted from the Visual BioMaze default representation model, represents compounds by a circle of radius 15 in which an icon `compound.bmp` is displayed; the formula of the compound is shown outside the circle (see Figure 4.1).

```
<Element>
  <Id>Compound</Id>
  <Glyph>
    <Shape>
      <Circle>
        <Radius>15</Radius>
      </Circle>
    </Shape>
    <BorderColor>None</BorderColor>
    <FillColor>None</FillColor>
    <Text>
      <Present>true</Present>
      <Content>
         <Label>!Formula</Label>
         <Label>?LABEL</Label>
      </Content>
      <Position>Out</Position>
      <Alignment>Center<Alignment>
    </Text>
    <Icon>
      <Present>true</Present>
```

```
      <Location>name_space/compound.bmp</Location>
    </Icon>
    <BorderVisibility>true</BorderVisibility>
  </Glyph>
</Element>
```

The different tags are as follows. `Id` defines the ID of the glyph. `Glyph` defines the associated graphical semantics and is composed of the following elements.

1. `Shape`, defining the shape associated to the element, that can be `Box`, `Circle`, `Ellipse`, `Arrow`, and `Tshape`. The allowed shapes and their particular attributes are defined using XML Schema see chapter *Contributing to Visual BioMaze*). The `Arrow` and `Tshape` cannot be a node Glyph. They are restrained to head and tail glyph (explained below).

2. `Text`, defining the text associated to the element. As shown in Figure 4.1, the text associated to elements can be displayed outside the shapes (e.g., for compounds) `Out` or inside them (e.g., for reactions) `In`. This is defined by the `Position` tag that can take the values `In` or `Out`. Further, the text can be either a generic text or a particular attribute of the represented object (e.g., its identifier). This is expressed by introducing a `?` character (e.g., `?ID` in the above example). The possible values are

   - `?ID`: The identifier of the object
   - `?LABEL`: The label of the object
   - `?TYPE`: The type of the object

   But you can also ask to display a specific attribute of the object by the string `?<attribute>`. By example if you want displaying the attribute `ReactionEc.Label` of a reaction node you have to set the this tag to `!ReactionEc.Label`. If the attribute is not present[1] in the node the system will display a `X`.

   The text can be aligned according to the the `Alignment` tag. The possible values are:

   - `Center`
   - `Right`
   - `Left`

   Users can define alternative text. For instance in code above, if the `Formula` attribute is not present, the system will show the label. Similarly the user can define a list of possible texts, the first found will be show.

3. `BorderColor` and `FillColor` defining the colors of the border and the interior of the shape. The possible values are either color name or RGB values. The available color names are (for version 1.X.X):

   - `Black`
   - `Red`

[1] If you want that this attribute is present in the reaction node, you have to ask it in the IQL query with the keyword `FILL ReactionEc.Label`.

- `Green`
- `Cyan`

The RGB format is # $XXX$ $XXX$ $XXX$, where $XXX$ stands for 3 digits. For the `TShape` the `BorderColor` has to be set, if not the glyph will not be drawn.

4. `Icon`, defining if an image must be shown in the shape. The size of the image will be automatically re-adjusted to the shape size. The images are defined only by its name and its name space, all of them are localized in the image contribution directory ( see chapter *Contributing to Visual BioMaze*).In order to avoid names collisions, the Visual BioMaze framework uses the namespace to reference images, XML files, etc. Then, even if two contribution projects use the same image name for two different files, the framework can differentiate them. We have chosen to use the **project name** as name space. For instance if we develop a contribution project which its name is `org.eclipse.vbm.contribution_example`, the image has to be referenced by `org.eclipse.vbm.contribution_example/plus.gif`. The framework will automatically resolve the image location according to the name space.

5. `BorderVisibility`, defining if the shape border is visible. This provides an easy way to introduce new shapes to characterize an interaction. Indeed, if an interaction must be represented by a complex form that does not correspond to the predefined shapes, the user can introduce it as an icon and set the border visibility to false.

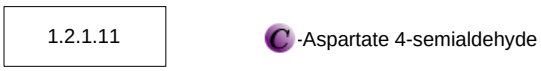



Figure 4.1: a) A compound is represented by a circle in which an icon is inserted, and b) A biochemical reaction is represented by a box.

The `Link` tag defines the graphical semantics associated to the edges between interaction types. The example below represents the links from catalysis to reactions by a line whose head is a circle of radius 18 in which is displayed the icon `catalyze.jpg` which stand in the contribution images directory (see chapter 8) and the text `+`, and without tail (see Figure 4.2).

```
<Link>
  <Between>
    <From>Catalysis</From>
    <To>Reaction</To>
    <Id>Catalysis</Id>
  </Between>
  <Effect>Continuous</Effect>
  <Color>Black</Color>
  <Head>
    <Present>true</Present>
    <Glyph>
      <Shape>
        <Circle>
```

```
          <radius>18</radius>
        </Circle>
      </Shape>
      <BorderColor>Black</BorderColor>
      <FillColor>None</FillColor>
      <Text>
        <Present>true</Present>
        <Content>
          <Label>+</Label>
          </Content>
        <Position>In</Position>
      </Text>
      <Icon>
        <Present>true</Present>
        <Location>name_space/catalyze.jpg</Location>
      </Icon>
      <BorderVisibility>true</BorderVisibility>
    </Glyph>
  </Head>
  <Tail>
    <Present>false</Present>
  </Tail>
  <width>1</width>
</Link>
```

The different tags are defined as follows:

1. `Between`, defining the interactions for which the edge is defined. This tag defines the source and target node.
2. `Effect`, defining the line style. The possible values are:
   - `Continue` A normal line
   - `Discrete` A doted line
3. `Head` and `Tail`: defining the glyphs for the extremities. These glyphs cannot be defined if the `Present` tag is set to `false`. The head and tail glyphs are important in biochemical networks visualization, since some interactions such as inhibition or catalysis are represented by a specific form at the edge extremities. The line width can also be customized.

### 4.2.2 Color Algorithm

Now, in one hand we have a representation model giving a set of glyphs, and in the other one we have the graph with nodes and arcs. The Visual BioMaze framework uses a list of color algorithms in order to assign a glyph to a particular node or arc. Users can develops their own color algorithms and load them in the system. This disposition enables a set of new functionalities: users can change the representation of a set of nodes in accordance with

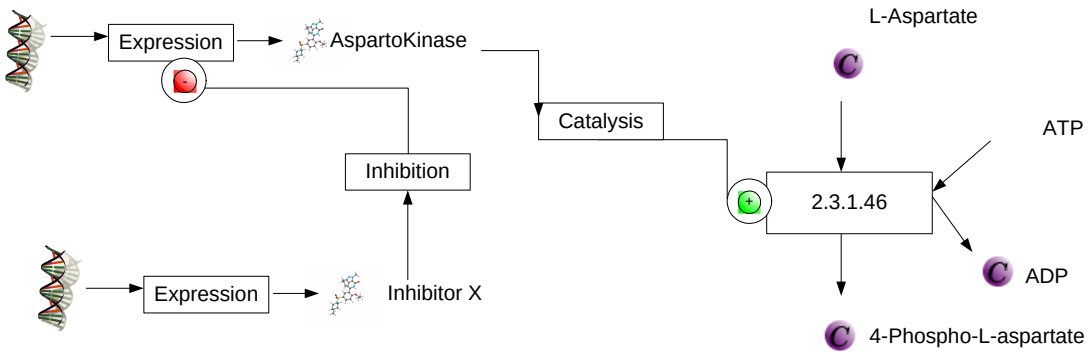



Figure 4.2: Example of a representation model. When representing links (e.g., from a catalysis to a reaction), graphical elements such as the head glyph, the line effect, the color, etc., are customizable by the representation model.

the value of certain attributes or in accordance with a particular path in the graph. User is now free to choose any graphical representation according to its need.

The first color algorithm which is applied is the default algorithm. This algorithm assigns a glyph to node if this glyph has an Id equals to the node-type. See chapter *Color Algorithms* for more information.

### 4.2.3 Undef and Any

The representation model provides a way to define a default glyph for nodes for which no representation is defined. Hence the glyph `Undef` defines the graphical semantics associated to this kind of nodes. Then, even if the default color algorithm does not match any glyph for a node, the `Undef` glyph will be loaded.

```
<Element>
 <Id>Undef</Id>
 <Glyph>
   <Shape>
     <Box>
       <Height>25</Height>
       <WIdth>30</Width>
     </Box>
   </Shape>
   <BorderColor>None</BorderColor>
   <FillColor>None</FillColor>
   <Text>
     <Present>true</Present>
     <Content>
        <Label>?LABEL</Label>
     </Content>
     <Position>Out</Position>
     <Alignment>Center<Alignment>
   </Text>
```

```
      <Icon>
        <Present>true</Present>
        <Location>undef.gif/Location>
      </Icon>
      <BorderVisibility>true</BorderVisibility>
    </Glyph>
  </Element>
```

The code above was extracted from the default representation model. If you define your own model do not forget to define the `Undef` node. If not, the `Undef` node will be handled by the Visual BioMaze Plug-in.

Writing a complete model is not easy, you have to define all the possible types between two nodes for all types of nodes. Further some IQL queries can create a new relation between two nodes, like between genes and a reactions, and typed `Catalysis`. For convenience, we have developed the `Any` keyword. You can define a graphical semantic associated to all arcs matching your definition.

```
<Between>
      <From>Any</From>
      <To>Any</To>
      <Id>Catalysis</Id>
</Between>
```

The example above shows a definition for a link between `Any` node but typed as `Catalysis`. The same can be done with `Any` types:

```
<Between>
      <From>Compound</From>
      <To>Reaction</To>
      <Id>Any</Id>
</Between>
```

What happen if the model contains a multiple matching ? The priorities are assigned as follows. The default color algorithm searches:

1. a perfect matching with $\langle from, to, type \rangle$
2. a link like $\langle \texttt{Any}, \texttt{Any}, type \rangle$
3. a link like $\langle from, to, \texttt{Any} \rangle$
4. a link like $\langle \texttt{Any}, \texttt{Any}, \texttt{Any} \rangle$

Otherwise, the default color algorithm gives a default link between the two considered nodes.

The definitions like $\langle from, \texttt{Any}, type \rangle$ or $\langle \texttt{Any}, to, type \rangle$ will be ignored. Figure 4.3 shows `Undef` nodes attached by `Undef` arcs.

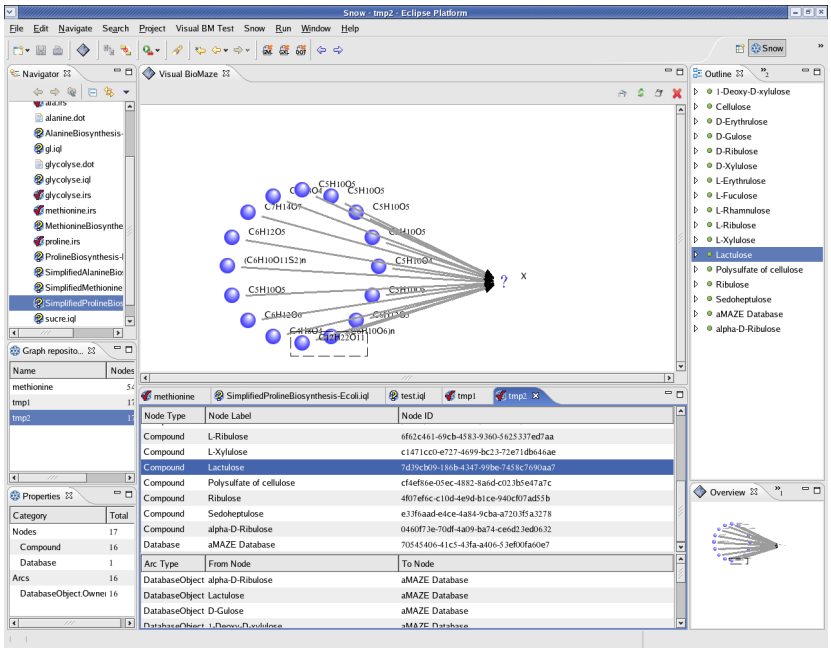



Figure 4.3: Example of undef arc and nodes. An undef arc is defined by the tuple: $\langle$`Any`, `Any`, `Any`$\rangle$. The house glyph represent the `Owner` typed arcs.

## 4.3 Apply any model

For applying a representation model follow these steps:

1. Go to the **Window** → **Preferences** → **Visual BioMaze Option** → **Advanced Configuration**

2. Check the **Representation Model Configuration page**

Select the model you want and if it is valid press the Apply button. If the model is not valid you cannot select it. The selection of the representation model is considered as advanced configuration because of non-contributor users will prefer use Style, which is the purpose of the chapter 7.

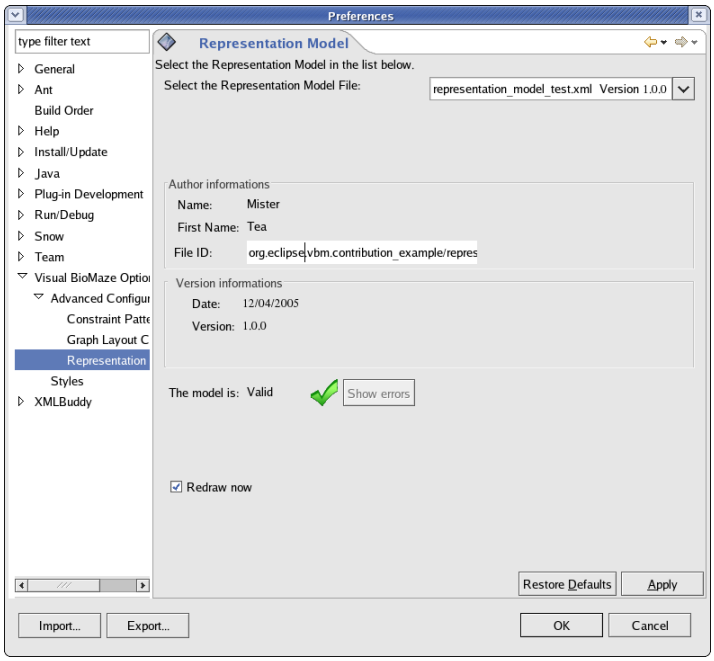



Figure 4.4: Only validated models are proposed to the user.

# Chapter 5

# Interchangeable Graph Layout Algorithms

## 5.1 Introduction

Many research efforts have been realized for many years in the area of graph layout and efficient graph layout algorithms have been developed. However, such algorithms are devised for general graphs and do not take into account semantic issues. These algorithms are inadequate for biochemical networks since these networks have associated particular semantics.

For example, a metabolic pathway follows a main direction that the visualization has to outline to represent it efficiently; further the regulatory pathway associated to it cannot be distributed around the pathway but also follows a specific direction. The same observation can be made about transduction signal graphs where the visualization has to outline the message direction.

Graph layout algorithms taking into account data semantics are currently a research domain, and there exists commercial tools that provide proprietary solutions. Our approach is to define a graph layout algorithm that is *interchangeable* giving the user the possibility to choose an existing algorithm, or to define his own.

## 5.2 Choosing an algorithm

Select **Window→Preferences→Visual BioMaze Option→Graph Layout Configura--tion→Advanced Configuration** (Figure 5.1). This page proposes all algorithms locally present. If you choose one, it will dynamically loaded and applied on graph to be visualized.

## 5.3 Creating your own Graph Layout algorithm

For creating your own algorithm, you have to create a blank Eclipse plug-in and extend our extension point.

### 5.3.1 Creating a blank plug-in

1. Select **New → Project → Plug-in Development →Plug-in Project**, press Next

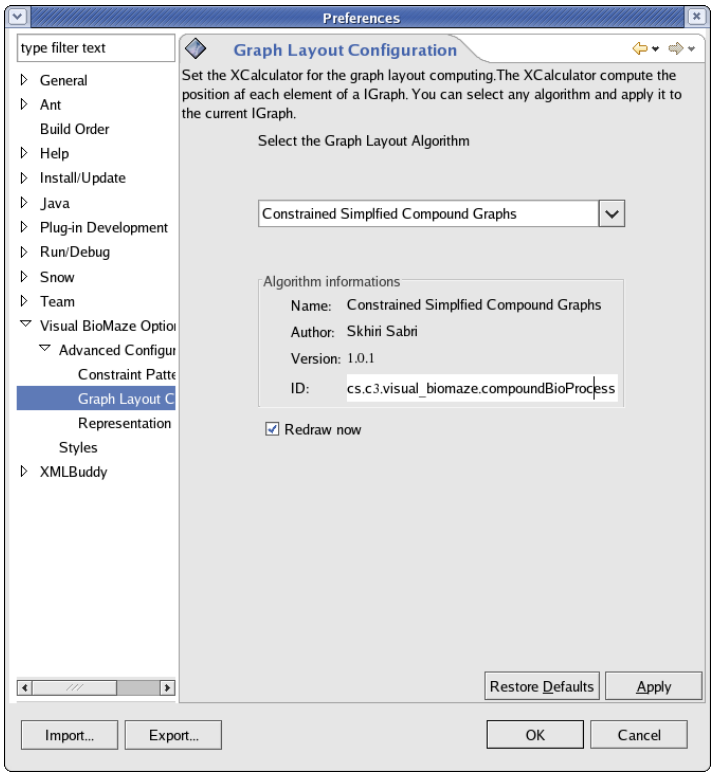



Figure 5.1: The preferences page of Graph Layout Configuration shows all the algorithms presents locally. The user can choose any one and apply it to the graph to be visualized.

2. Enter the project name, for example `org.eclipse.biomaze.gl_algorithm`, press Next
3. Check the **Create a Java project** box, and press Next
4. Enter your name in the **Provider Name** field, uncheck **Contribute to the UI**, and press Next
5. In the **Template page**, do not select anything, press Finish

### 5.3.2 Extending Visual BioMaze

Open the **MANIFEST.MF**

1. Select the **Dependencies** subtab
2. Press the Add button of the **Required Plug-ins** and add `be.ac.ulb.cs.c3.visual_bio-maze.master`, `be.ac.ulb.cs.c3.visual_biomaze.master` and `north.graph.core`.
3. Save the file
4. Select the **Extensions** subtab of the **MANIFEST.MF** file
5. Press the Add button and select `be.ac.ulb.cs.c3.visual_biomaze.host.GL_Calculator`

6. Click on the the GL_Calculator extension in the **All extensions** view, and fill the **id** field in the Extension Details view. This step is important ! Do not fortget it. For example set : `org.eclipse.biomaze.gl_algorithm.testGL`
7. Select the GL_Calculator extension, press the right mouse button, and select **New→Provider**
8. Select the provider tag and fill the **name** field in the Extension Details view
9. Select the GL_Calculator extension, press the right mouse button, and select **New→version** and fill the **version** in the Extension Details view
10. Select the GL_Calculator extension, press the right mouse button, and select **New→Calculator**
11. Select the **Calculator** tag and in the Extension Details view
    - In the **Class** field, press the **class** link
    - Set the class name, and press Finish

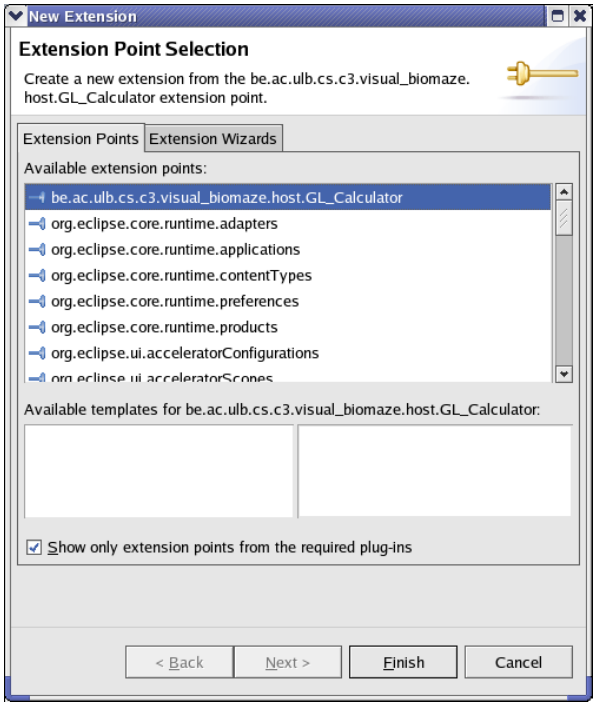



Figure 5.2: From the **Extensions** subtab of the `MANIFEST.MF` file and add a new Visual BioMaze extension.

### 5.3.3 Implementing your algorithm

Now that the plug-in is ready, the last thing to do is to implement your algorithm. Open the class generated by the PDE[1]. This class implements the `ICalculator` interface, the dedicated interface to the GL algorithm. Your algorithm has to be implemented in the

[1]Plug-in Development Environment

computeGL(IGraph, IEditableData, ConstraintSet, IVBMMonitor) method. For any information about IGraph or IEDitableData), check the javaDoc: http://cs.ulb.ac.be/research/biomaze/restricted/javaDoc. The ConstraintSet class represents the current reference constraint pattern XML file described in chapter 6. Finally the IVBMMonitor class provide a way to show feed back information about algorithm's state advancement.

For assigning a position to a node, you have to extract the Glyph[2] attribute and assign the position directly to the Glyph.

```
IGlyph glyph = node.getValue(repository, IGlyph.GLYPH);
glyph.setPosition(new Position(x,y));
```

Where the repository variable is the argument of the computeGL(IGraph, IEditableData, ConstraintSet, IVBMMonitor)method, and Position is a ulb.vbm.glyph.Position object.

It is done ! If you do not assign the position of some nodes, their default position is (0,0) and then, they will not be processed by the visualizer.

Often graph layout algorithms requires to specify bend points on edges. Visual BioMaze provide an easy way to set these kind of points. You have to extract the ILinkGlyph[3] attribute and assign the position of the bend point directly.

1. You have to create a virtual INode corresponding to the bend point. You can use the Basic node Builder of the north.graph.core plugin or your own INode builder.

   ```
   INode virtualNode = (INode) BasicNode.newBasicNode("bend_"+pos,IGLYPH.BENDPOINT);
   ```

2. Now you have to create a virtual glyph and put it as GLYPH attribute of the virtual node

   ```
   IGlyph virtualGlyph = new Glyph();
   virtualGlyph.setShape(new BendPoint(2,2));
   virtualNode.putAttribute(repository,IGlyph.GLYPH, virtualGlyph);
   ```

3. All the bend points corresponding to one edge have to be sort and store in a java.utils.List

   ```
   List edgeBendPoints = new ArrayList();
   edgeBendpoints.add(virtualNode);
   ```

4. Finally, you have to store the vector of bend points in the corresponding attribute of the LinkGlyph of the edge

   ```
   if(edge.containsAttribute(IGlyph.LINK_GLYPH){
     ILinkGlyph link = edge.getValue(repository, IGlyph.LINK_GLYPH);
     link.setBendPoint(bendNode);
   }
   ```

[2]The Glyph is the graphical representation of the node.
[3]The ILinkGlyph is the graphical representation of the edges.

The entire code:

```
// for one edge
List edgeBendPoints = new ArrayList();
INode virtualNode = (INode) BasicNode.newBasicNode("bend_"+pos,IGLYPH.BENDPOINT);
IGlyph virtualGlyph = new Glyph();
virtualGlyph.setShape(new BendPoint(2,2));
virtualNode.putAttribute(repository,IGlyph.GLYPH, virtualGlyph);
edgeBendpoints.add(virtualNode);
//...The same for all bend points associated to the edge (IArc)

if(edge.containsAttribute(IGlyph.LINK_GLYPH){
   ILinkGlyph link = edge.getValue(repository, IGlyph.LINK_GLYPH);
   link.setBendPoint(edgeBendPoints);
 }
```

Applying a graph layout algorithm could be very long. The Visual BioMaze framework provides services allowing: to report the state of the algorithm processing, to run the algorithm in the background and to cancel the algorithm. Thus, graph layout algorithm writers can define what the current task is with the `IVBMMonitor monitor` object:

```
// Step 3: minimizing edge-crossing
monitor.setTaskName("Minimizing edge-crossing");
monitor.setWorked(80);
```

The `setWorked(int val)` method accept an integer between 0 and 100, which represents the unit of the complete work, the algorithm can then update the progress view (launched when a graph drawing job was started). The supplied `IVBMMonitor` should be also used to check for cancellation requests made from the progress view. **It is the responsibility of the graph layout algorithm** to frequently check the cancellation status of the monitor and respond to a cancellation by exiting as soon as possible the `computeGL()` method. The following code report progress and respond to a cancellation:

```
int k = 1;
int n = nodes.size();
monitor.setTaskName("Assign Circular Layout");
for (Iterator iter = nodes.iterator(); iter.hasNext();) {
   /*Reporting and responding to cancellation*/
   if (monitor.isCanceled()) {
       return;
   }
   monitor.setWorked(k/n*100);

   /*Assign circular positions*/
   INode node = (INode) iter.next();
   IGlyph glyph = (IGlyph) node.getValue(repository, IGlyph.GLYPH);
   int x = (int) (1 * edgeLenght * Math.cos(2 * k * Math.PI / n)) +
```

```
                                                    2*edgeLenght;
    int y = (int) (1 * edgeLenght * Math.sin(2 * k * Math.PI / n))+
                                                    2*edgeLenght;
    glyph.setPosition(new Position(x, y));
    k++;
  }
```

### 5.3.4 Exporting your algorithm

The last step is exporting your plug-in.

1. Select your plug-in in the **Package Manager**
2. Press the right mousse button and select **Export**
3. Select **Deployable plugins and fragments**, press Next
4. Select the file name of the zip file that will be exported
5. Press finish
6. Unzip your plug-in in the Plugin directory in your ECLIPSE_HOME and restart Eclipse

Your plug-in is now available in the Visual BioMaze preference page. If not, something goes wrong during the exportation.

### 5.3.5 Debugging your algorithm

While your algorithm is under development, you do not need to export it at each test. You can select **Run→Run As→Run-time Workbench**. And if your plug-in is valid, it will be available in the preference page of the Run-time Workbench. As already said, the main advantage of graph layout framework of Visual BioMaze is to apply any algorithm on the graph to be visualized, but also distribute your algorithm.

## 5.4 Distributing your Graph Layout algorithm

Eclipse provides two ways to distribute your algorithm: via an update web site or via a deployable plugin. We advise to consult the Eclipse web site for any information. In conclusion, it means that everybody can contribute to the Visual BioMaze Framework with the addition of his own graph layout algorithm but can also distribute it easily to the community.

# Chapter 6

# Color Algorithms

## 6.1 Introduction

The Visual BioMaze framework provides a way to assign any particular graphical semantics to a nodes or a set of nodes. Let us imagine that we need to highlight a particular path in the visualized graph, e.g., the backbone of a metabolic pathway. We want to be able to overwrite the graphical representation of each node and arc of this path. This is exactly the purpose of the color algorithm. The color algorithm is the code that assigns or overloads a glyph to a node or a link glyph to an arc. Then, users can implement its own list of algorithms and load them into the framework in order to highlight particular information. Notice that the default color algorithm of the framework matches glyphs of the resulting model with nodes having the same node-type as the glyph-id.

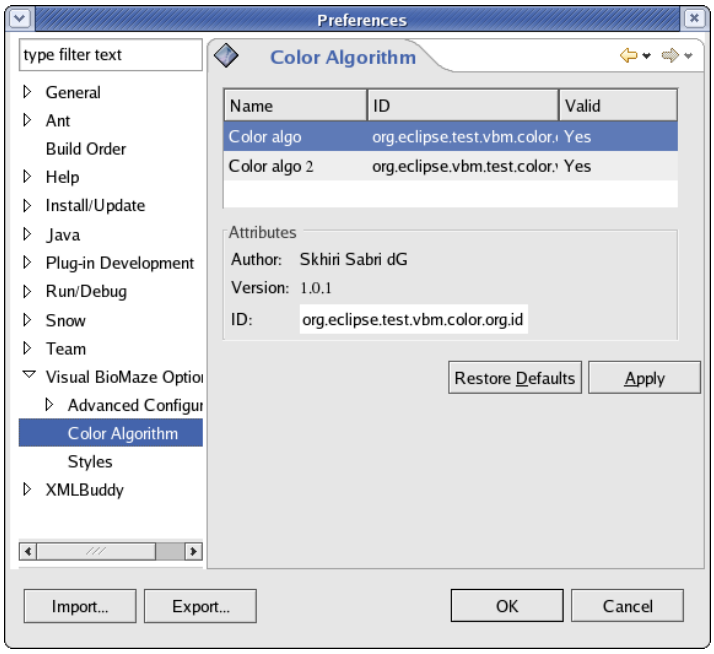



Figure 6.1: The preferences page of color algorithm shows all the algorithms presents on the system.

## 6.2 Creating your own color algorithm

For creating your own algorithm, you have to create a blank Eclipse plug-in and extend our extension point.

### 6.2.1 Creating a blank plug-in

1. Select **New** → **Project** → **Plug-in Development** →**Plug-in Project**, press Next
2. Enter the project name, for example `org.eclipse.biomaze.color_algorithm`, press Next
3. Check the **Create a Java project** box, and press Next
4. Enter your name in the **Provider Name** field, uncheck **Contribute to the UI**, and press Next
5. In the **Template page**, do not select anything, press Finish

### 6.2.2 Extending Visual BioMaze

Open the **plugin.xml**. Note that from Eclipse 3.1 the **plugin.xml** file is replaced by the **MANIFEST.MF** in the **META-INF** directory:

1. Select the **Dependencies** subtab
2. Press the Add button of the **Required Plug-ins** and add `be.ac.ulb.cs.c3.visual_bio--maze.master`, `be.ac.ulb.cs.c3.visual_biomaze.host` and `north.graph.core`.
3. Save the file
4. Select the **Extensions** subtab of the **MANIFEST.MF** file
5. Press the Add button and select `be.ac.ulb.cs.c3.visual_biomaze.host.colorAlgorithm`
6. Click on the the colorAlgorithm extension in the **All extensions** view, and fill the **id** field in the Extension Details view. This step is important ! Do not fortget it. For example set : `org.eclipse.biomaze.color_algorithm.test`
7. Select the color extension, press the right mouse button, and select **New→Provider**
8. Select the provider tag and fill the **name** field in the Extension Details view
9. Select the colorAlgorithm extension, press the right mouse button, and select **New→version** and fill the **version** in the Extension Details view
10. Select the colorAlgorithm extension, press the right mouse button, and select **New→Algorithm**
11. Select the **Algorithm** tag and in the Extension Details view
    - In the **Class** field, press the **class** link
    - Set the class name, and press Finish

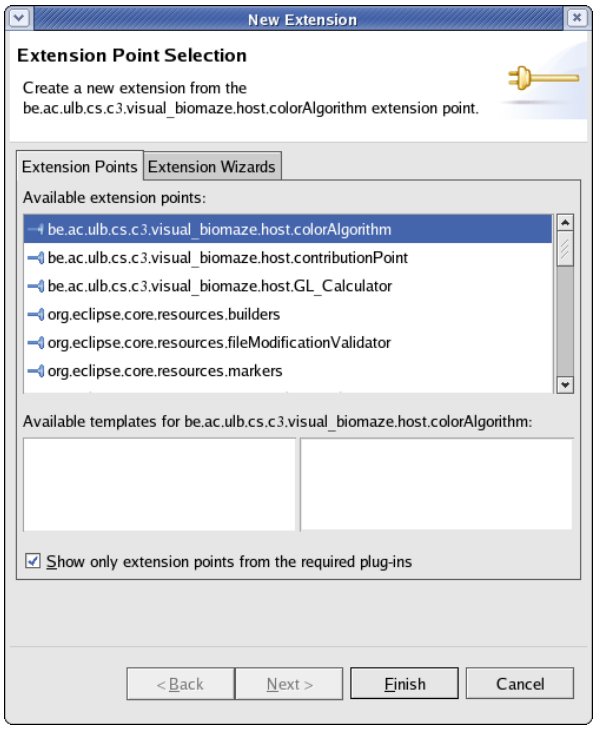



Figure 6.2: From the **Extensions** subtab of the `MANIFEST.MF` file and add a new Visual BioMaze extension.

### 6.2.3 Implementing your algorithm

Now that the plug-in is ready, the last thing to do is to implement your algorithm. Open the class generated by the PDE[1]. This class extends the `AbstractColorAlgorithm` class. Your algorithm has to be implemented in the `applyColorAlgorithm(IGraph, IEditableData` method. For any information about `IGraph` or `IEDitableData`, check the javaDoc: `http://cs.ulb.ac.be/research/biomaze/restricted/javaDoc`. The `AbstractColorAlgorithm` class contains two methods allowing user to overload the mapping of the default aglorithm for each node and arc:

- *overLoadGlyph(INode node, IEditableData repository, IGlyph newGlyph)*: overloads the glyph of the node by the new glyph
- *overLoadLinkGlyph(IArc arc, IEditableData repository, ILinkGlyph newLinkGlyph)*: overloads the link glyph of the arc
- *IGlyph extractGlyph(String nameSpace, String fileName, String id, INode node)*: extracts the glyph named by the *id*, from the file $fileName$ in the *nameSpace*. The notion of name space is explained in the chapter *Contributing to Visual BioMaze*. The node for which this glyph is destined must also be specified in order to resolve the label tag such as $?LABEL$.
- *IGlyph extractLinkGlyph(String nameSpace, String fileName, String id)*: extracts the link glyph named by the *id*, from the file $fileName$ in the *nameSpace*.

[1]Plug-in Development Environment

The code below shows an example of a color algorithm. It assigns the glyph *k_1_glyph* to all nodes *BiologicalReaction* and assigns the link glyph *k_1_link* to the outgoing arcs of these nodes.

```
public void applyColorAlgorithm(IGraph graph, IEditableData repository) {
       Set nodes = graph.getNodes();
       for (Iterator iter = nodes.iterator(); iter.hasNext();) {
        INode node = (INode) iter.next();
        if(node.getType().equals("BiologicalReaction")){
            IGlyph glyph = extractGlyph("org.eclipse.vbm.contribution.example
                                        .v6", "modeltest1.xml", "k_1_glyph", node);
            if(glyph!=null){
                this.overLoadGlyph(node, repository, glyph);
            }

           Set arcs = node.getOutArcs();
           for (Iterator iterator = arcs.iterator(); iterator.hasNext();) {
            IArc arc = (IArc) iterator.next();
            ILinkGlyph link = extractLinkGlyph("org.eclipse.vbm.contribution.
                                               example.v5", test1.xml", "k_1_link");
            if(link!=null){
                this.overLoadLinkGlyph(arc, repository, link);
            }
        }
       }
      }
```

### 6.2.4 Assigning your algorithms

The color algorithms are assigned via a style, see chapter *Styles*.

# Chapter 7

# Graphical constraints

## 7.1 Introduction

Any graph layout algorithm can be implanted in the Visual BioMaze Framework, some of them can take into account the particular semantic of a domain. For instance, in biochemical networks, the main pathway follows a main direction vertically while genetic regulation has to be drawn horizontally. These semantic rules are defined by the application domain. Similarly, a transduction signal pathway is not shown in the same way for a genetician as for a biologist. From a pure graphs point of view nothing can help us to know the domain-dependent rules that explain how to represent the graph.

Visual BioMaze provides a way in which users can define and express its own graphical constraints according to a particular application domain.

## 7.2 Definition

As representation models, the graphical constraints are XML files validated by XML schemas. Users can choose between several constraint files in the preference pages as shown by Figure 7.1. Users can define two types of constraint files: (1) Simple constraints which are used by the CSCGL algorithm (see chapter *Implemented algorithms*) and (2) the 8-directions constraints which are more expressive.

## 7.3 Simple constraints

The simple constraints are basic graphical constraints. They define a set of arcs which have to be drawn in a different direction than other arcs. They are constituted by the constraint pattern and by the topological constraints.

### 7.3.1 Constraint pattern

The constraint pattern is a set of typed-arcs. This set has to be matched by the graph layout algorithm with one or more subgraphs. For instance, the constraint pattern $\langle Expression, Traduction, Catalysis \rangle$ refers to a subgraph having three successive arcs typed respectively: Expression, Traduction and Catalysis. This constraint specifies that this subgraph must follow a different orientation from the main direction of the graph. The future

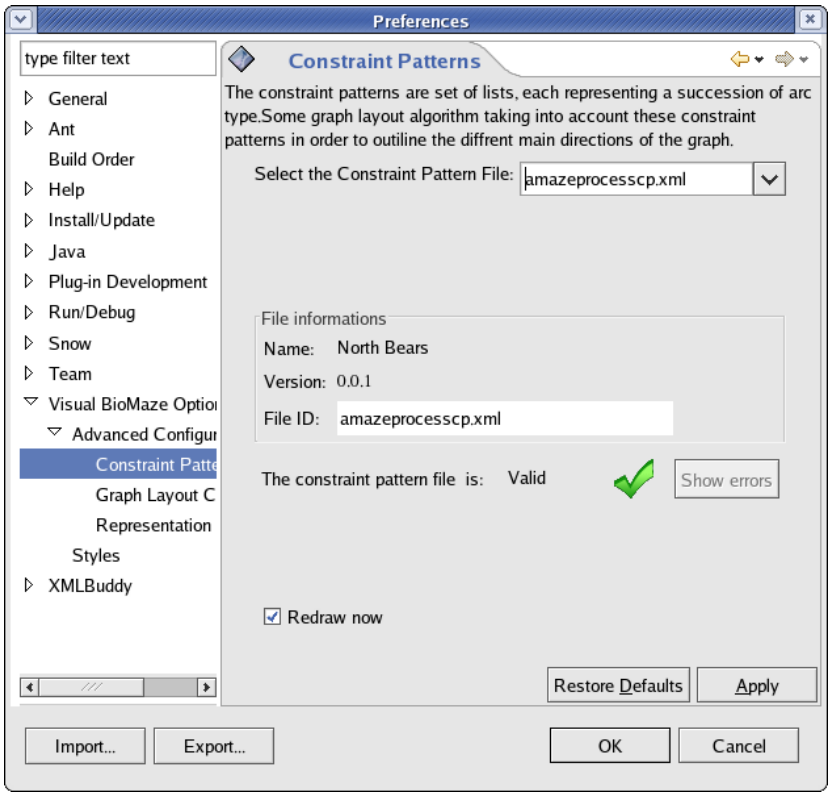



Figure 7.1: Users can choose between several constraints file.

versions of graphical constraint will contain explicitly the direction that the subgraph has to follow. As a result, the genetic regulation will be drawn differently from the main direction (See chapter *Implemented Algorithms*).

### 7.3.2 Topological constraint

The topological constraints define graphical constraints associated with topological properties. From Visual BioMaze 1.0.10, users can define one type of topological constraint: the cyclic constraint. These kinds of constraints define the behaviour to apply when the graph layout algorithm meets a cycle. Two behaviours can be applied: either cycles are drawn along a circle, as biochemical cycles in metabolic pathways, we call them *Topological cycles* either they are drawn normally but, then, we have to inverse a set of arcs (the backward edges).

The first tag $< TPCycle >$ defines the allowed arc types that can be found in a cycle. If a cycle is composed only of these types, it can be drawn along a circle otherwise they must be drawn normally. However we can not inverse any arcs, for instance in layered graph layout algorithm, if we inverse arcs belonging to the main direction we break it. The second tag $< BackwardArcs >$ defines the allowed arc types that the graph layout algorithm may inverse.

```
<ConstraintsList xmlns="http://cs.ulb.ac.be/research/biomaze"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://cs.ulb.ac.be/research/biomaze modelcp.xsd">
```

```
  <Author>Skhiri dit Gabouje Sabri</Author>
  <Version>1.0.0</Version>
  <ConstraintsSet>
    <ConstraintPattern>
      <typedArc>Expression</typedArc>
      <typedArc>Traduction</typedArc>
      <typedArc>Catalysis</typedArc>
    </ConstraintPattern>
    <ConstraintPattern>
      <typedArc>Expression</typedArc>
      <typedArc>Traduction</typedArc>
      <typedArc>Inhibition</typedArc>
    </ConstraintPattern>
    <TopologicConstraints>
      <Cyclic>
        <TPCycle>
          <typedArc>Substrate</typedArc>
          <typedArc>Product</typedArc>
        </TPCycle>
        <BackwardArcs>
          <typedArc>Inhibition</typedArc>
        </BackwardArcs>
      </Cyclic>
   </TopologicConstraints>
 </ConstraintsSet>
</ConstraintsList
```

## 7.4 8-directions constraints

These constraints are more accurate, they are composed by three types of constraints. If we want to be able to draw any kind of graphs according to specific domain semantics, we have to express the information "'how to draw a graph in an application domain?"' We have chosen to express these constraints as a set of *local constraints*, *cycle constraint*, and *backbone constraint*.

### Local constraints

A local constraint is defined by a set of tables as those shown by Tables 7.1 and 7.2. The first table defines for one reference node type, in this case *biologicalReaction*, the node types that must be placed around the reference node and their relative positions (Figure 7.3). The second table defines how to manage more than one constrained node in the same position. For each local constraint on a reference node (Table 7.1) we have to define one such a table.

Nodes involved in a local constraint can be the reference node for other constraints. A priority policy assigns higher priority to reference nodes contained in the backbone. The priority processing will be explained in Section **??**. The local constraints define also the kind of behavior to adopt if more than one node is matched in one constraint container. For

Table 7.1: Local constraints around the reference typed node

| Node type | Arc type | Pos. | Dir. | D.ratio |
|---|---|---|---|---|
| Compound | Reaction.in | NW | Forward | 5 |
| Compound | Reaction.out | SW | Reverse | 5 |
| Catalysis | Catalysis.Reaction | E | Forward | 10 |

instance, Table 7.2 states that if two nodes must be placed in NW of the reference node, those will be aligned vertically.

Table 7.2: Alignment policies

| S | N | E | W | NE | NW | SE | SW |
|---|---|---|---|---|---|---|---|
| Ver. | Horiz. | Ver. | Ver. | Ver. | Ver. | Horiz. | Ver. |

.

**Circular contraints**

Circular constraints define whether a cycle must be drawn along a circle or must be drawn by inversing the minimal set of backward edges [4]. They are constituted by a set $C_T$ of arc types specifying the allowed types for a circle. For cycles having at least one arc which is not contained in $C_T$ we inverse the backward edges, otherwise, we draw it along a circle.

**Backbone constraints**

These constraints define the set of nodes and arcs that constitute the main direction flow of the graph (Table 7.4).

Table 7.3: The set of nodes and arcs constituting the backbone of metabolic pathways.

| From | To | Arc type |
|---|---|---|
| Compound | BiologicalReaction | Substrate |
| BiologicalReaction | Compound | Product |

### 7.4.1 Cycle contraints

Cycle constraints define which cycles have to be drawn along a circle and which has to be drawn normally, i.e., by inversing backward edges [4]. This is exactly the same constraints as the topological constraints of simple constraints.

### 7.4.2 Backbones constraints

These constraints aim to define which nodes must be extracted from the graph in order to build the main direction flow. Then, they define the set of nodes and arcs which constitute this main direction (Table 7.4).

Notice that at the moment, only two algorithms uses graphical constraints:

- The Constrained Compound Graph Layout algorithm (3CGL) which needs a 8-directions constraint file

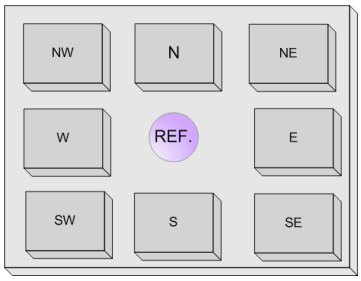



Figure 7.2: Local constraints define the disposition of nodes types around a node type reference.

Table 7.4: Defines the set of nodes and arcs which constitute this main direction flow.

| `From` | `To` | `Arc type` |
|---|---|---|
| Compound | BiologicalReaction | Substrate |
| BiologicalReaction | Compound | Product |

- The Constrained Simplified Compound Graph Layout algorithm (CSCGL) which needs a simple constraint file

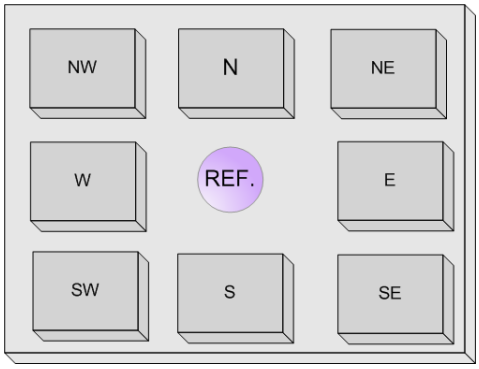



Figure 7.3: The local constraints define the disposition of nodes-types around a node-type reference.

# Chapter 8

# Styles

## 8.1 Introduction

As already said, an innovative aspect of the Visual BioMaze framework is the possibility to load and apply any representation model, any graph layout algorithm, any graphical constraint and any color algorithm. Most users do not consider important which algorithm or model to choose, most time they do not know the purpose of them. In order to simplify such choice we have defined the notion of *Style*. A style is a kind of profile in which a contributor has defined a specific representation model, a specific set of graphical constraint, a specific graph layout algorithm and a specific set of color algorithms. The aim of the style is bringing together the most suited parameters according to a graph-type and presents them to users in a simple way. However, users can still change any of these parameters, thus, the current style becomes a *custom style*, and it can be saved in the current configuration.

For instance, the style `vbm_style_process.xml` gathers the most suited representation model (`amazeprocess2.xml`), graph layout algorithm (Constrained Simple Compound Graph Layout), and graphical constraints (`amazeprocess.xml`) for the visualization of the aMAZE metabolic pathways. Figure 8.2 shows this style in the Visual BioMaze style properties view.

Recall that in this document we call basic *contribution* either the representation model or the set of graphical constraint or the graph layout algorithm.

In order to apply a style, the user has to open:
**Window→Preferences→Visual BioMaze Option→Style**. Figure 8.1 shows the preference page of style. The content of the style can be visualized in the Style properties view. This view can be opened by pressing the button **View Style** of the page. Users can verify if the content of the style is valid (Figure 8.2).

If the user modify the style by applying another contribution from the advanced configuration such as another graph layout algorithm or representation model or constraint pattern set, the current style becomes *customized* and then, can be saved by pressing the button **Save as** of the style preference page. This button opens a window on which the user must set the name of style, the name of the new contributor, and the name of the file. The user must define a relative file name. The style will be saved as `exp/filename.xml`

The styles provide another important property: users can define a list of representation models. Indeed, let us imagine a user wants to contribute to the Visual BioMaze framework by providing a new representation model for the metabolic pathway, but only for a specific subset of them: the genetic regulation. In this situation, the user must define the whole model

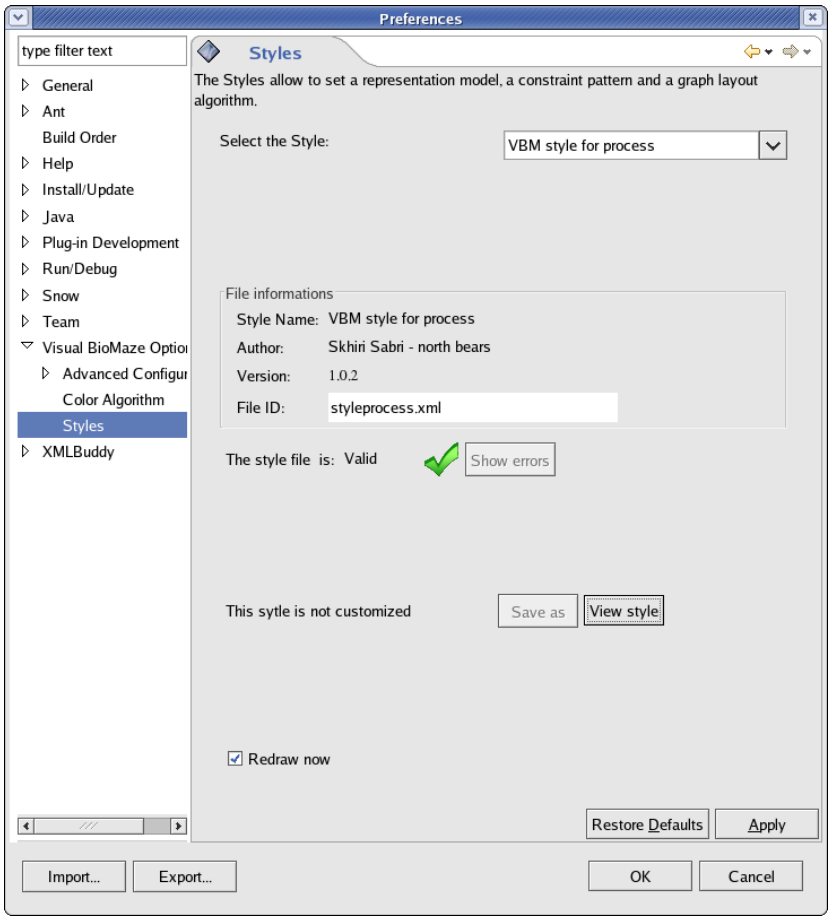



Figure 8.1: The preference page of Style. The user can select a style and apply it to all graphs to be visualized.

by describing not only the graphical representation associated to the genetic regulation but also the graphical representation associated to each other metabolic-types such as compound, biological reaction, polypeptide. This work can take a considerable amount of time. It is why the Styles allows using a sorted list of models. The given models are processed as they are sorted. The first defines a set of `glyph` and `linkGlyph` the graphical representation associated, to a node-type and to an arc-type respectively, by the default color algorithm. If the next models define other graphical representations for either same node-type or same arc-type, the first graphical representation is overwritten. This is the same rule for the next models. Then, the user can define as first model the aMAZE representation model for metabolic pathways and as next model its own representation model for the genetic regulation. As a result the user must only define the part of the representation model, which describes the genetic regulation. The default color algorithm of the framework will match glyph of the resulting model with node having the same type as the id of the glyph. Further, user can over load the assigned glyph by using a list of color algorithms, for instance to show particular information in the graph.

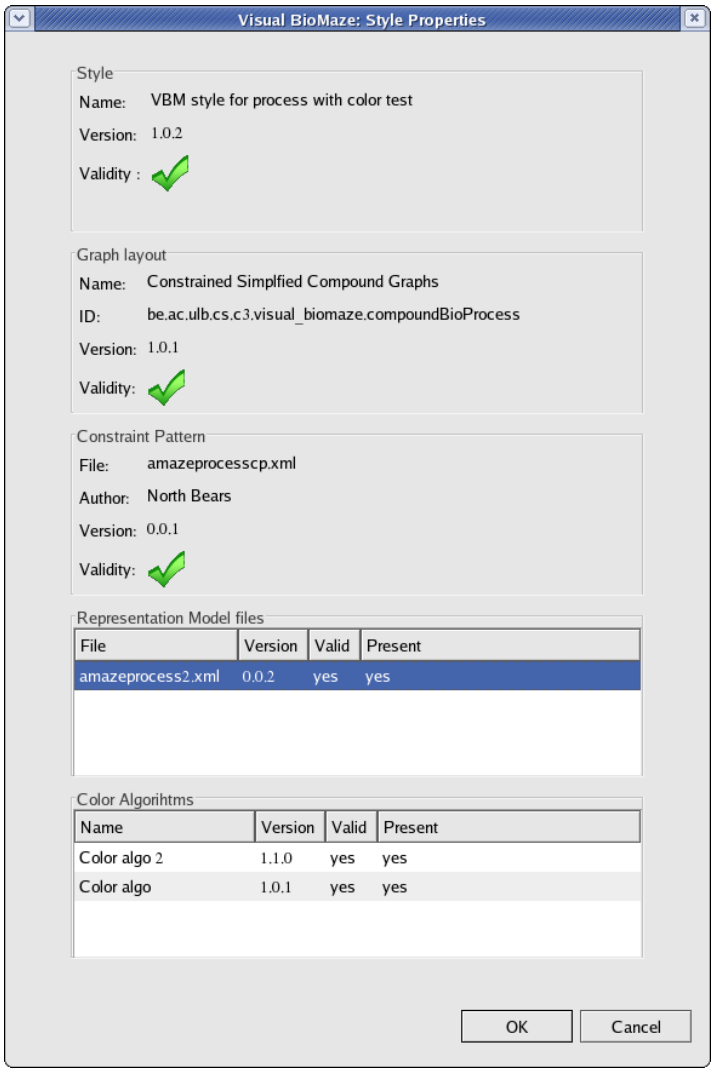



Figure 8.2: The Style properties view. This page describes the content of the style but also the validity of its content.

## 8.2 The xml Style file

As other basic contributions, the styles are xml files. Their structure is shown by the xml code below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Style xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                              xsi:noNamespaceSchemaLocation="styleschema.xsd">
  <Author>Zinedine Zidane</Author>
  <StyleName>Style for metabolic pathway</StyleName>
  <Version>1.0.1</Version>
  <Components>
      <ModelFilesList>
         <ModelFile>default.xml</ModelFile>
```

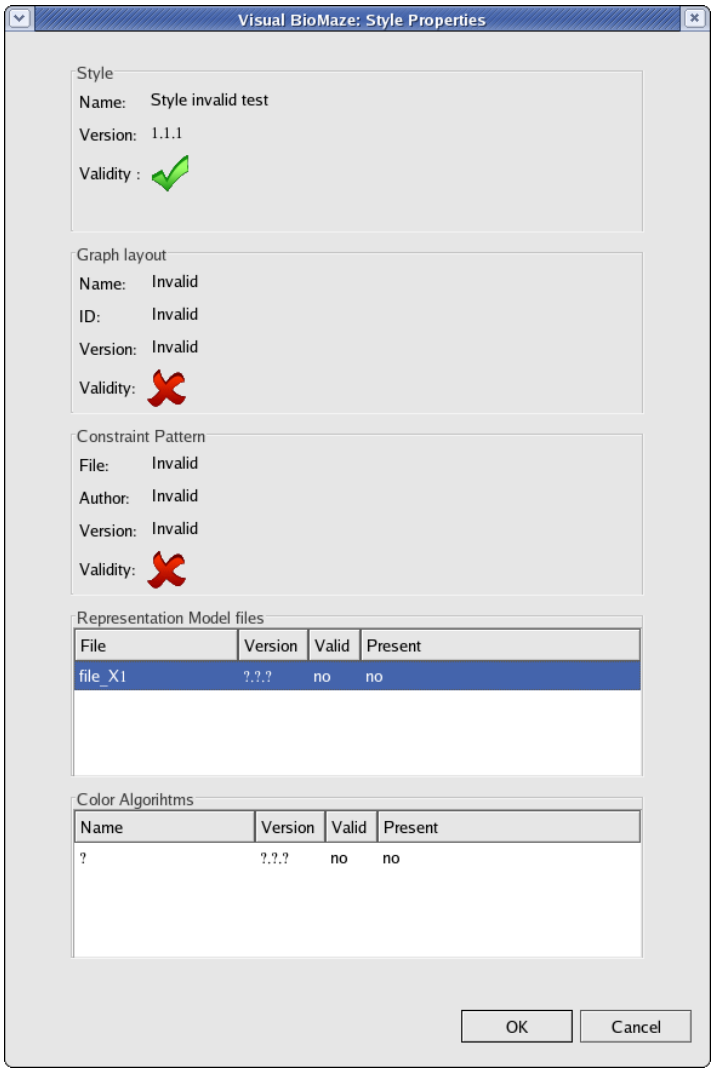



Figure 8.3: The Style properties view of an invalid style. If one of the components of the style is invalid this view highlights the wrong elements.

```
        <ModelFile>cs.ulb.ac.be/metabolic_pathway_model.xml</ModelFile>
        <ModelFile>cs.ulb.ac.be/compound_model.xml</ModelFile>
      </ModelFilesList>
      <ConstraintsFile>defaultcp.xml</ConstraintsFile>
      <AlgorithmID>be.ac.ulb.cs.c3.visual_biomaze.host.be.ac.ulb.cs.c3.
                                visual_biomaze.host.Calculator</AlgorithmID>
      <ColorAlgorithmsList>
        <ColorID>cs.ulb.ac.be.org.eclipse.id.color</ColorID>
      </ColorAlgorithmsList>
  </Components>
  <!-- Ceated by the Visual BioMaze Framework -->
</Style>
```

The tags are defined as following:

- `Author` defines the name of the contributor.
- `StyleName` is the name of the style.
- `Version` is the version of the style.
- `Components` defines the components of the style.
  - `ModelFilesList` defines the list of models. The names of models are preceded by the name space of the contribution. Then, we avoid collisions between different styles having the same file name. The model files as the `default.xml`, defined with no name space, refers to representation models which are provided by the distribution of Visual BioMaze. If user wants to use representation models present in the Visual BioMaze framework, their name space can be directly copy/paste from the representation model preference pages (Figure 8.4).
  - `ConstraintsFile` defines the XML constraint set file. As the model file, the file name is preceded by a name space. In this example, `defaultcp.xml` is not a contribution, it is the default constraint pattern file of Visual BioMaze. The name space of constraint pattern files can be directly copy/paste from the preference pages.
  - `AlgorithmID` defines which graph layout algorithm the style uses. The id is composed by the concatenation of the plug-in name which contains the algorithm (in the example `be.ac.ulb.cs.c3.visual_biomaze.host`) and the id of the extension point extended by the algorithm (`be.ac.ulb.cs.c3.visual_biomaze.host.Calculator`). This id can be copy/paste directly from the graph layout preference page, next to the label `ID`(Figure 5.1).
  - `ColorAlgorithmsList` defines a list of id representing a color algorithm.This id can be copy/paste directly from the color algorithm preference page, next to the label `ID`(Figure 6.1).

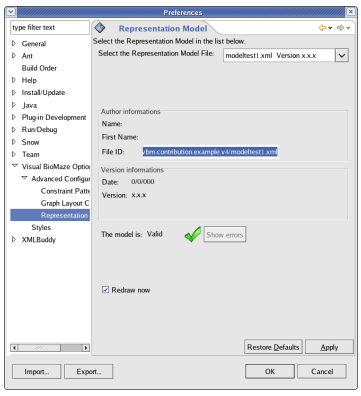



Figure 8.4: The name space of the contribution can be directly copy/paste from preference pages.

The chapter "'Contributing to Visual BioMaze "' explains how users can write their own contributions, further, we will see that the name space is the name of the contribution project.

# Chapter 9

# Contributing to Visual BioMaze

## 9.1 Introduction

The Visual BioMaze provides a set of tools which aim to help users to write new contributions. These tools are intregrated into the Eclipse platform. The operation consists in creating a new *Visual BioMaze Contribution* project and defining the different contributions that the user wants publish. Further, the contribution project has to be added to a specific extension point of Visual BioMaze. The next section explain step-by-step how contributing to Visual BioMaze.

In order to edit more easily the xml file, we advice the reader to donwload the free *XML Buddy* plug-in for Eclipse. For more information visit the web site `http://xmlbuddy.com/`.

## 9.2 How contributing to Visual BioMaze

First, the user must create a contribution project:

1. Open **File→New→Project**
2. Under the section **Visual BioMaze**, select **New Visual BioMaze Contribution Project**(Figure 9.1)
3. Press **Next**
4. Set the name of the project, for instance: `org.eclipse.vbm.contribution.example`, recall that the project name will be used as name space, then, choose a name which can be unique as an URL for instance `cs.ulb.ac.be`.
5. Press **Finish**

The project `org.eclipse.vbm.contribution.example` must appears in the user's worksapce (Figure 9.2).

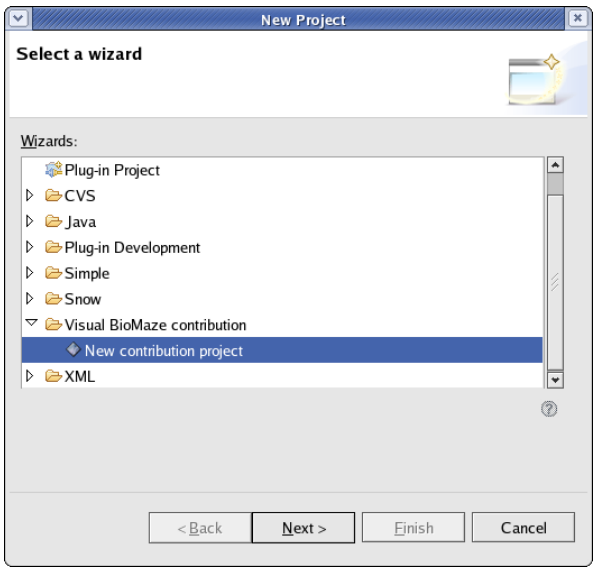



Figure 9.1: Select new project.

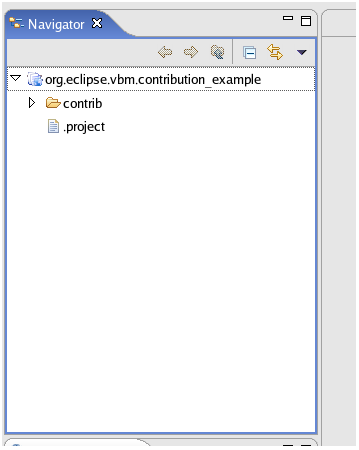



Figure 9.2: The contribution project is created in the workspace.

## 9.3 Downloading the Visual BioMaze Schema references

As already said, the contribution files are XML-structured files. The XML schemas which valid the structure of these contributions are contained in the Visual BioMaze Framework. In order to stay synchronize with the different versions of these schemas, users can download the Visual BioMaze Reference project which contains all reference schemas. When a new version of Visual Biomaze is released, users can directly update the reference project, and then, they have the same schemas that the new release. We have chosen the CVS system in order to facilitate the checkout of the reference project.

1. Select the *CVS Repository Exploring* perspective
2. In the *CVS Repositories* view, click on the right button of the mouse and select **New→Repository Location** (Figure 9.3)
3. Fill the fields as shown by Figure 9.4
   - **Host**: cs-devel.ulb.ac.be
   - **Repository path**: /cvsroot/vbm
   - **User**: cvspub
   - **Password**: leave blank
   - **Connection type**:pserver
4. Press **Finish**
5. Select the project *vbm_config_reference* in the `HEAD` branch
6. Checkout it (Figure 9.5)
7. Return to the Resource View, the reference project is now in your workspace (Figure 9.6)

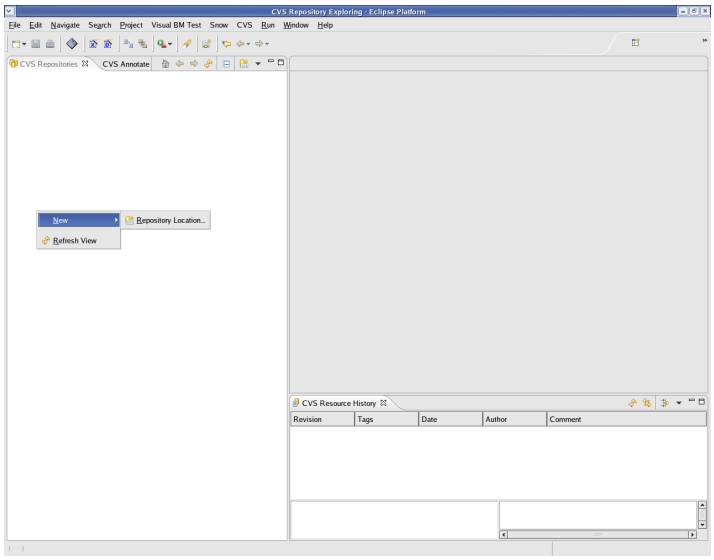



Figure 9.3: Create a new repository location.

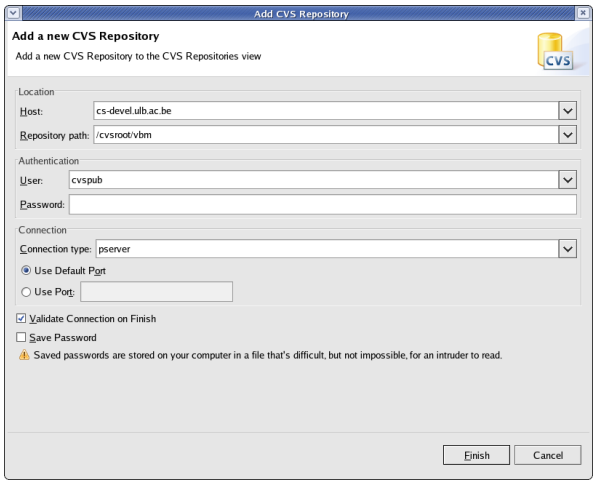



Figure 9.4: Fill the fields of the new repository.

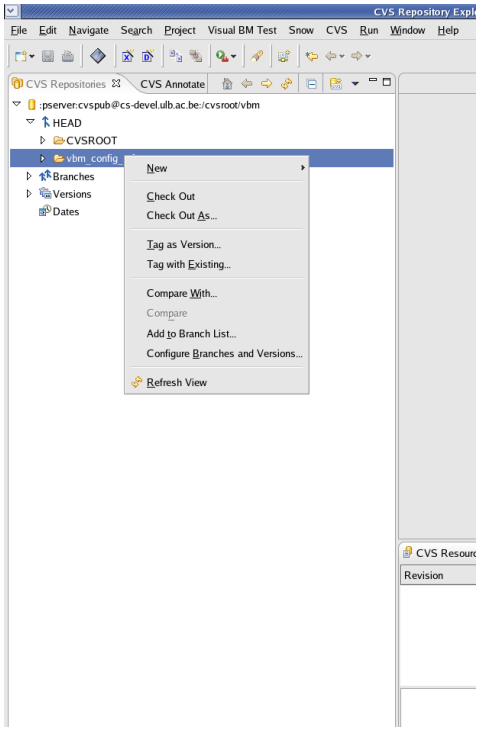



Figure 9.5: Checkout the refence project.

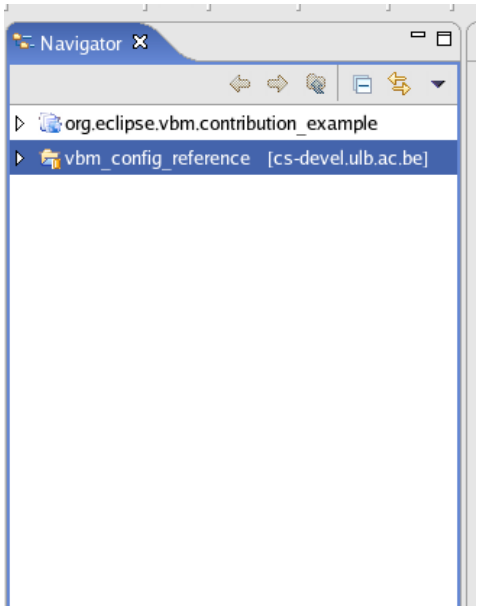



Figure 9.6: The reference project is now in workspace.

Now, let us create a contribution.

## 9.4 Creating a representation model file

1. Open **File→New→Other**
2. Under the section **Visual BioMaze**, select **New Visual BioMaze Contribution** (Figure 9.7)
3. Press **Next**
4. Select the project `org.eclipse.vbm.contribution.example`, set the name of the contribution, for instance `representation_model_test.xml`, and select the type of contribution, here **Representation model**
5. Press **Finish**

The file `representation_model_test.xml` is then created in the contribution project Figure 9.9). The generated XML model file contains one *element* which defines the graphical representation of the *undef* node-type, and one *link* which defines the representation of the $\langle any, any, any \rangle$ arc. For more information about the representation model, see the chapter Representation Model. The user can add element and link in this file. Fill the tags `Date` and `Numero` of `Version`, and the tag `Name` anf `FirstName` of `Author`. Now, the model is ready to be loaded.

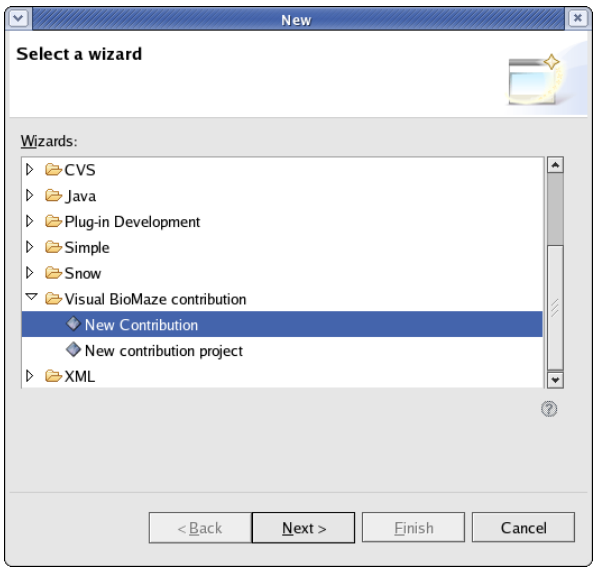



Figure 9.7: Create a new contribution

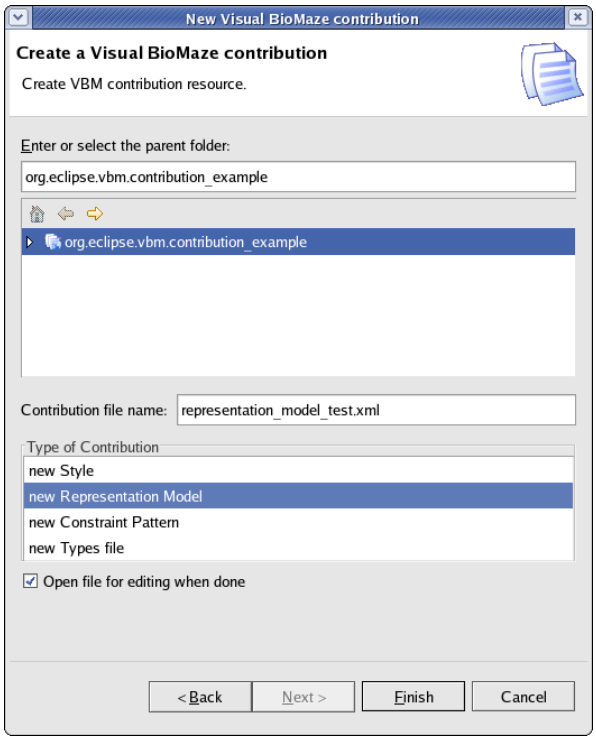



Figure 9.8: Select the project and the type of contribution.

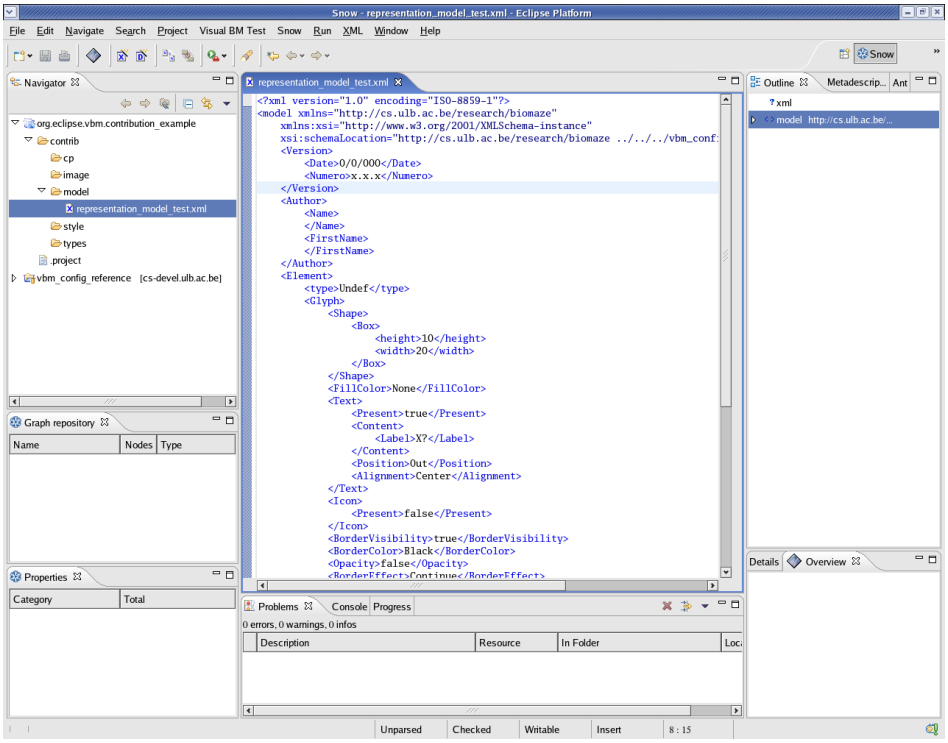



Figure 9.9: The representation model is created in the project

### 9.4.1 Testing the new model

Users can test any contributions with the *direct load* feature of Visual BioMaze.

1. Select the contribution project to test
2. Click on the right button of the mouse
3. Select **Load contribution**

All the contributions of the projects are now available in the preference pages as shown by Figure 9.10.

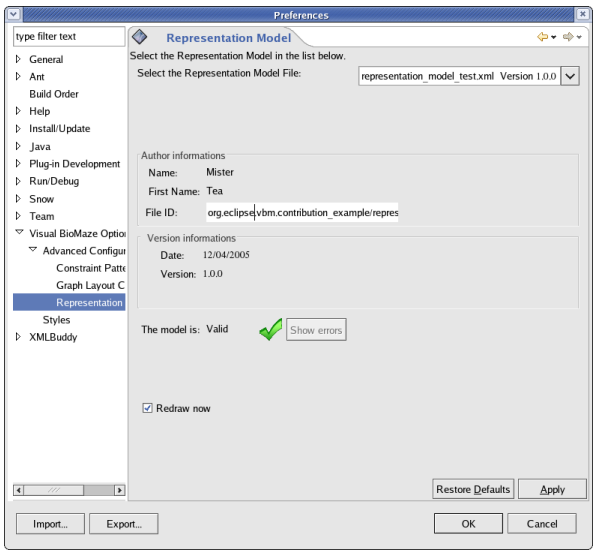



Figure 9.10: Visual BioMaze allows to load any conributions of a project directly in the system.

### 9.4.2 Including icons in representation models

As already sais in the chapter *Customizable Representation Model*, users can include some image as graphical representation of either nodes or head (or tail) of arcs. How users can reference these images ? First users have to import the image in the workspace and more particularly in the image directory of the contribution project.

1. **File→Import**
2. Select **File system** and browse the file system in order to fin the directory in which the image is contained
3. Select the image and set the field **Into folder:** with *org.eclipse.vbm.contribution_example/contrib/image*, i.e, in the image directory of the contribution project (Figure 9.11)

Now we can reference the image in our new representation model. In order to avoid name collision, the Visual BioMaze framework uses the namespace to reference images, XML files, etc. Then, even if two contribution projects use the same image name for two different files, the framework will be able to differentiate them. We have chosen to use the **project name** as name space. Then, in our example, the image has to be referenced by `org.eclipse.vbm.contribution_example/plus.gif`. The framework will automatically resolve the image location according to the name space.

```
<Icon>
   <Present>true</Present>
   <Location>org.eclipse.vbm.contribution_example/plus.gif</Location>
</Icon>
```

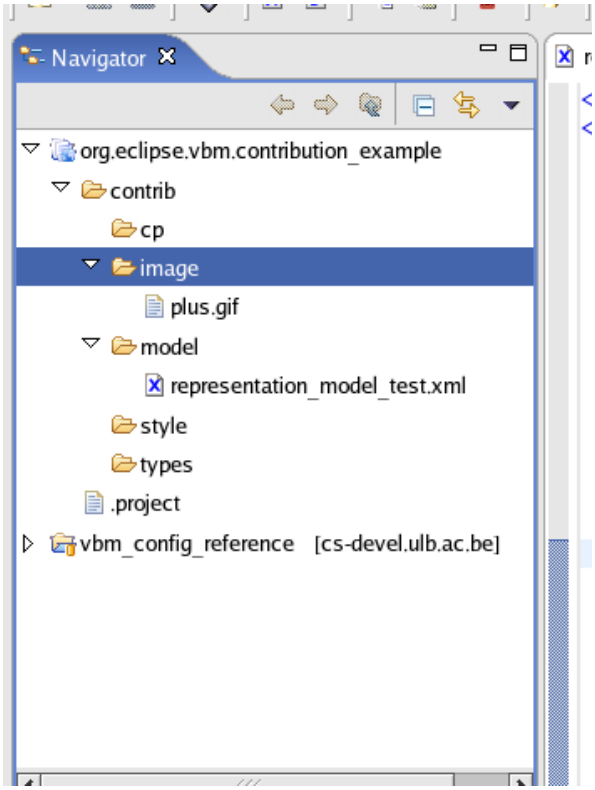



Figure 9.11: Import the icon `plus.gif` in the workspace.

## 9.5 Creating a graphical constraint

1. Open **File→New→Other**
2. Under the section **Visual BioMaze**, select **New Visual BioMaze Contribution** (Figure 9.7).
3. Press **Next**
4. Select the project `org.eclipse.vbm.contribution_example`, set the name of the contribution, for instance `constraint8dir_test.xml`, and select the type of contribution, here **new Constraint pattern** (Figure 9.12).
5. Press **Next**
6. select which kind of constraints you want to create: Simple constraints or 8-directions constraints (Figure 9.12).
7. Press **Finish**

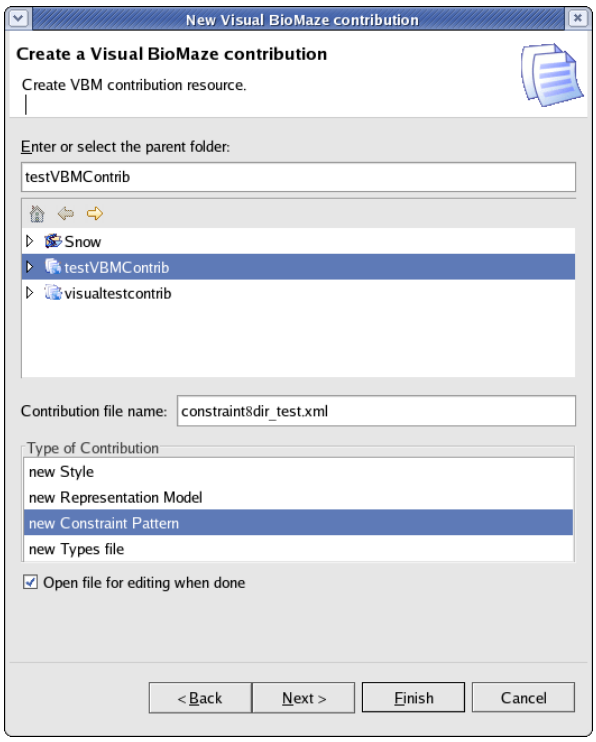



Figure 9.12: Create a constraint file.

The file `constraint8dir_test.xml` is then created in the contribution project. The generated constraint pattern file contains an empty *Constraint set* (Figure 9.14).. For more information about the graphical constraints, see the chapter Graphical constraints. Fill the tags `Author` and `Version` and other empty tags.

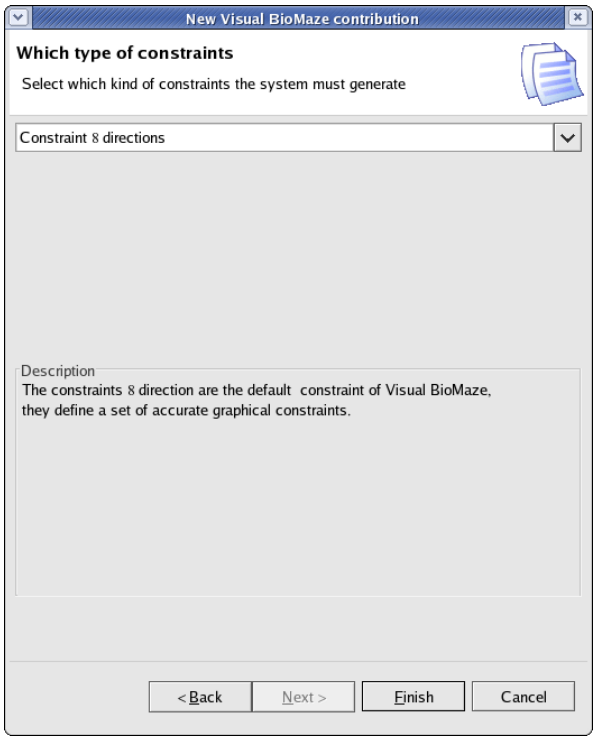



Figure 9.13: Select which kind of constraints must be created.

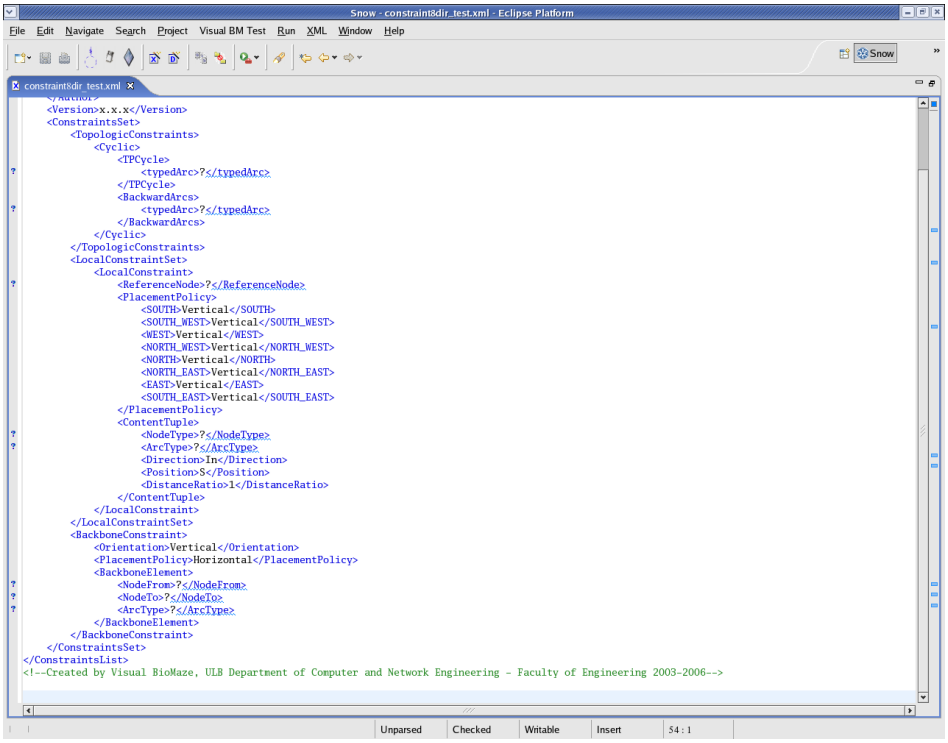



Figure 9.14: The created file.

## 9.6 Creating a Style

1. Open **File→New→Other**
2. Under the section **Visual BioMaze**, select **New Visual BioMaze Contribution** (Figure 9.7).
3. Press **Next**
4. Select the project `org.eclipse.vbm.contribution.example`, set the name of the contribution, for instance `style_test.xml`, and select the type of contribution, here new Style.
5. Press **Finish**

The file `style_test.xml` is then created in the contribution project. The generated style contains empty fields. For more information about the graphical constraints, see the corresponding chapter. Fill the file as following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Style xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                              xsi:noNamespaceSchemaLocation="../styleschema.xsd">
  <Author>Skhiri Sabri</Author>
  <StyleName>Style test for tutorial</StyleName>
  <Version>1.0.0</Version>
  <Components>
      <ModelFilesList>
         <ModelFile>default.xml</ModelFile>
         <ModelFile>org.eclipse.vbm.contribution_example/representation
                             _model_test.xml</ModelFile>
      </ModelFilesList>
      <ConstraintsFile>org.eclipse.vbm.contribution_example/constraint
                             _test.xml</ConstraintsFile>
      <AlgorithmID>be.ac.ulb.cs.c3.visual_biomaze.host.be.ac.ulb.cs.c3.
                                  visual_biomaze.host.Calculator</AlgorithmID>
      <ColorAlgorithmsList>
         <ColorID>rg.eclipse.vbm.contribution_example.id.color</ColorID>
     </ColorAlgorithmsList>
  </Components>
  <!-- Ceated by the Visual BioMaze Framework -->
</Style>
```

Then, you have created a style which contains:

- a representation model which is the superposition of the default representation model file and of your contribution.
- a constraint pattern file which is your, created in the contribution project.
- the default graph layout algorithm ID. The id's of each algorithm can be copy/paste directly from the graph layout preference page.

- a color algorithm which is yours. Similarly, the id's of each algorithm can be copy/paste directly from the graph layout preference page. Note that the presence of color algorithms is optional.

Note that the reference to your contribution files is made by adding the name of your contribution project. This disposition allows to differentiate the contribution of users, the name of the project is then used as name space by the Visual BioMaze framework. All the refenrences to an XML contribution file of the contribution project must be done with the name space.

### 9.6.1 List of deafault files

When users write a new style, they have not to create new representation models or new constraint pattern files, they can use the default files.

- Representation model: `default.xml`
- Constraint pattern: `defaultcp.xml`
- Graph layout algorithm: `be.ac.ulb.cs.c3.visual_biomaze.host.be.ac.ulb.cs.c3.visual_biomaze.host.Calculator`

## 9.7 Extending Visual BioMaze

The contribution system of Visual BioMaze uses the plug-in system of Eclipse. Then in order to registry the contribution, the user must define which extension point of Eclipse he will extend.

1. Select the contribution project, here: `org.eclipse.vbm.contribution.example`
2. Click on right button of the mouse and select **PDE Tools→Convert to plug-in**
3. Open the `MANIFEST.MF` file
4. Open the **Dependencies** sub-tab
5. In the required plug-ins section add `be.ac.ulb.cs.c3.visual_biomaze.host` and `be.ac.ulb.cs.c3.visual_biomaze.master` (Figure 9.16)
6. Open the **Extension** sub-tab
7. In the **All Extensions** add the extension `be.ac.ulb.cs.c3.visual_biomaze.host.contributionPoint` (Figure 9.17)
8. Select the extension point you have just created
9. Click on right button of the mouse and select **New→ContributionItem**
10. Select the contribution item you have just created
11. Fill the field id for instance `org.eclipse.vbm.contribution.example`

12. Select the type. You have the choice to load either some contribution (for instance one or more of the contributions we have created) or the whole project with all contribution defined. In this example we will load the whole project. Then select the type **Contribution Project** (FIgure 9.18)

13. Fill the field name, for instance `Contribution example`

14. The **Contribution** field describes the contribution to load, in our example, as we have chosen to load the entire project we have to select the `.project` file of the project. Then press the **Browse** button and select the `.project` file.

As we have chosen the `.project` file, we have to be sure that the file is embedded with the contribution. Then,**select the `build.properties` file of the contribution project and check the `.project` file in both, binary and build source** (Figure 9.15).

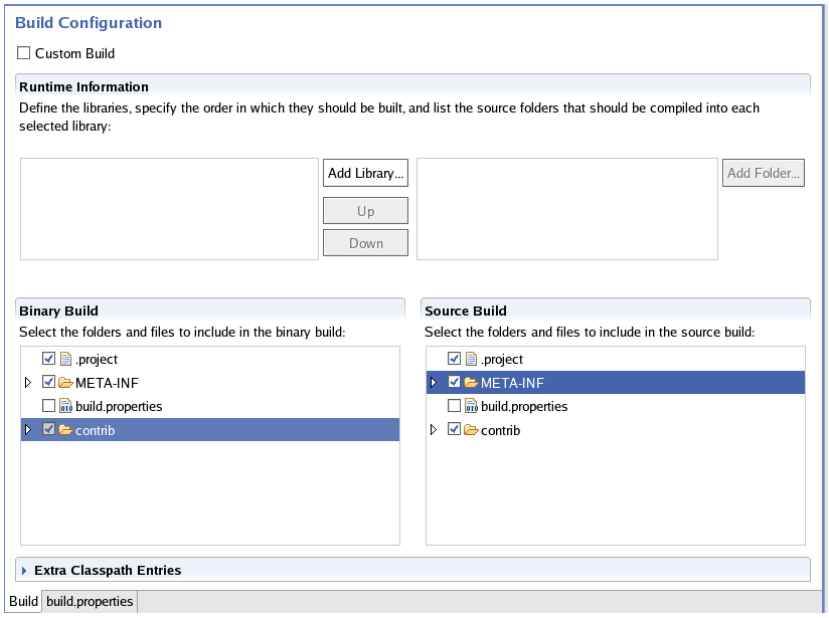



Figure 9.15: Check the .project file.

Now the contribution project is ready to be packaged and distributed! Note that the grpah layout algorithm can be defined in a contribution project. If a contribution style refers to it, it must specified the id: `org.eclipse.vbm.contribution.example`. concatened with `algorithm_ID` which is the of the algorithm extension point.

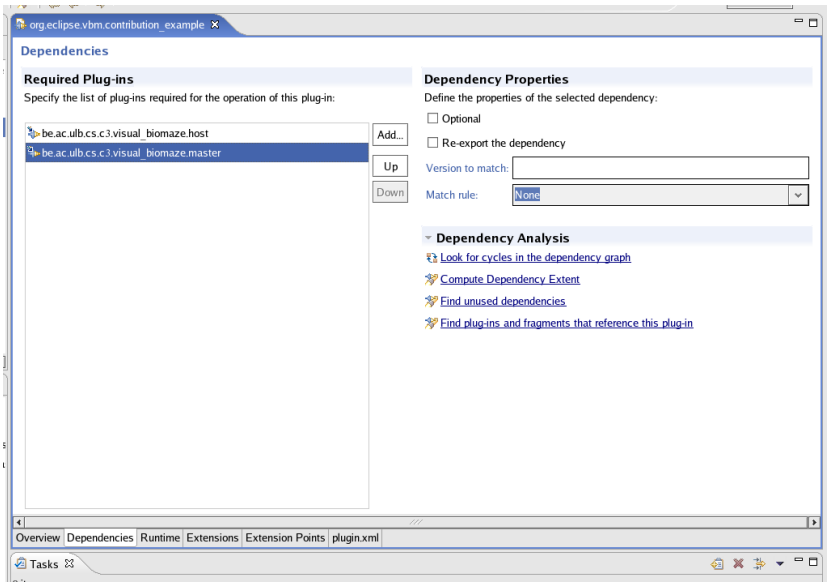



Figure 9.16: These two plug-in are required.

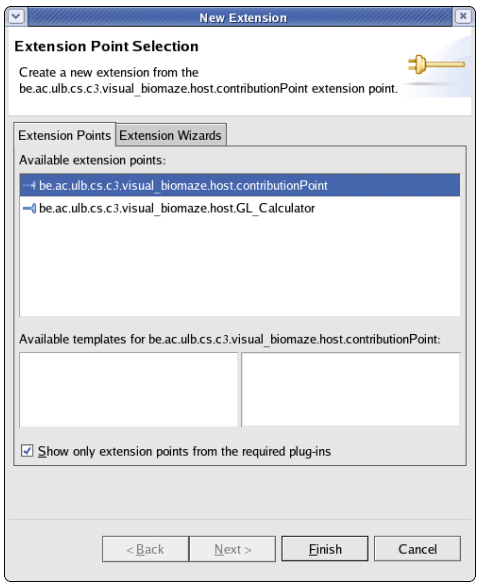



Figure 9.17: Select the extension point that you want extend.

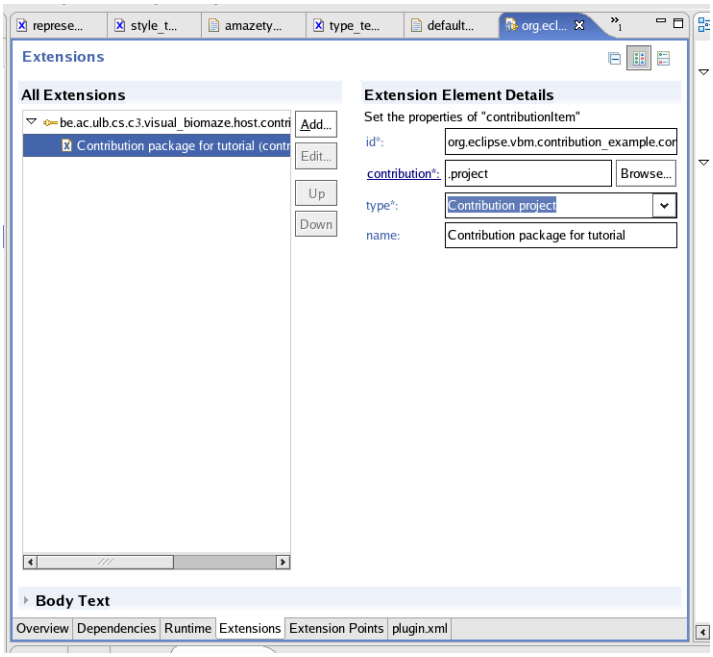



Figure 9.18: Set the project as the extension point.

## 9.8 Creating new arc-types and node-types

As exposed in the previous chapter, constraint patterns use the arc-types and nodes-type. These types correspond to types present in the graph to visualize. These types are restricted by an XML schema pointed by the XML schema that validates constraint patterns. If the graph we want displaying is having new other types, we have to define compliant graphical constraints and then, we have to define new allowed-type. The Visual BioMaze framework allows user to define its own types of node and arc.

### 9.8.1 Creating a new type file (Figure 9.19)

1. Select **File→New→Other**
2. Select **New contribution** under the **Visual BioMaze Contribution** section
3. Select the contribution project, in our example `org.eclipse.contribution_example`
4. Set the **Contribution file name** to `type_test.xsd`, note that the extension in **xsd**
5. Select the **New Types file** as type of contribution
6. Press **Finish**

Then, Visual BioMaze has created the new type file in `org.eclipse.contributio-n_example/contrib/types/type_test.xsd`. The creeated-file describe all the possible type, for the moment, only the type *Any* and *Undef* are authorized for nodes, and only *Any* for arcs. Now, add your new type in the file, just add a line `<xs:enumeration value="'nodetype1"'>` below the tag `ElementType` for a new type node and below the tag `ArcType` for new type of arc. The code below shows how adding the type `nodetype1`, `nodetype2` and `arctype1`, `arctype2`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:simpleType name="ElementType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="nodetype1"/>
    <xs:enumeration value="nodetype2"/>
    <xs:enumeration value="Any"/>
    <xs:enumeration value="Undef"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ArcType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="arctype1"/>
    <xs:enumeration value="arctype2"/>
    <xs:enumeration value="Any"/>
  </xs:restriction>
  </xs:simpleType>
```

```
</xs:schema>
<!--Created by Visual BioMaze, ULB Department of Computer and
                    Network Engineering - Faculty of Engineering 2003-2006-->
```

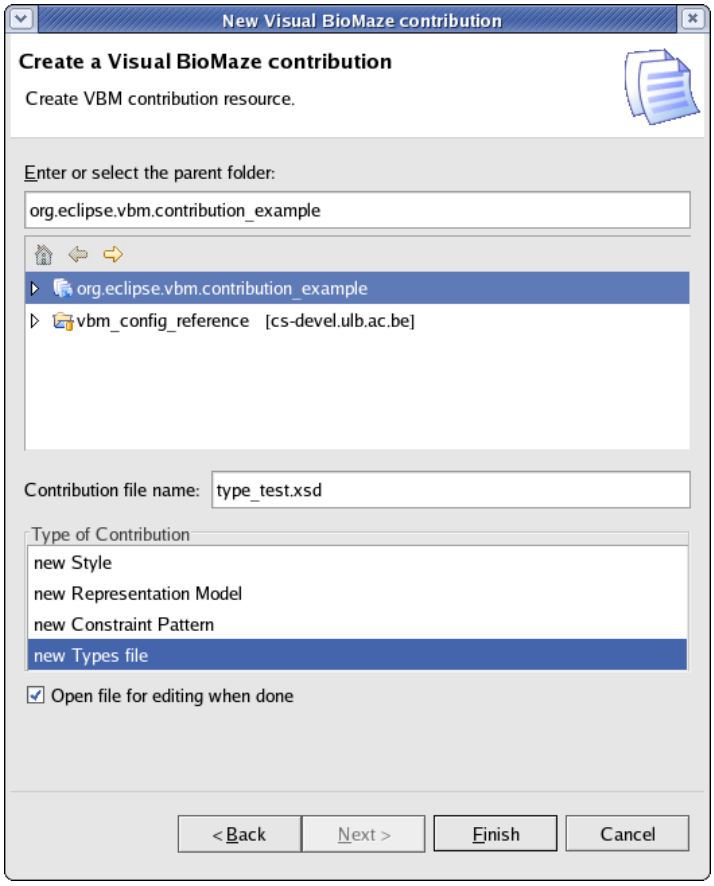



Figure 9.19: First, create a new type file schema.

Now we have a set of allowed-types. Let's define these types in our representation model.

### 9.8.2 Import the default schema model

The reference to the allowed-types is accomplished by a reference in the XML schema that validates the constraint pattern. In order to update this reference we have to import this schema in the contribution project next to our constraints.

1. Select the contribution project, in our example `org.eclipse.contribution_example`
2. Select the directory `org.eclipse.contribution_example/contrib/cp`
3. Select **File→Import**
4. Select **File system**, press **Next**
5. Browse trough your file system in order to fin the directoy `vbm_config_reference/model`, press **Next**

6. Check the file `defaulcp.xsd`

The default constraint pattern schema is then imported in your contribution project. Now we can update the references. First our constraint pattern has to reference the imported schema:

1. Open the file `constraint8dir_test.xml` the constraint pattern we have just create in previous section
2. Change the schema location tag as following: replace

```
xsi:schemaLocation="'http://cs.ulb.ac.be/research/biomaze
                        ../../../vbm_config_reference/model/defaultcp.xsd
```

   by :

```
xsi:schemaLocation="'http://cs.ulb.ac.be/research/biomaze defaultcp.xsd
```

Now we have to update the reference to the allowed-types in the schema

1. Open the imported schema in the contribution project, i.e., `org.eclipse.contributio-n_example/contrib/cp/defaultcp.xsd`
2. Change the include schema location tag as following: replace

```
<xs:include schemaLocation="../types/amazetype.xsd">
```

   by :

```
<xs:include schemaLocation="../types/type_test.xsd">
```

That's all ! Now your model references the local XML schema and the schema references the new type we have just created.

*Note:* When the contribution project is loaded, the contribution types files are loaded with their name space in order to avoid name collisions. The XML schemas embedded in contribution project that reference theses types are then invalid (the reference is wrong). These schemas are edited by the Visual BioMaze framework at loading time and the references are automatically updated with the right name space. Then, users have not to modify the references to the type file with name space before loading the contribution project.

## 9.9 Testing the contribution

In order to test the right behaviour of your contribution plug-in, the user can launch a instance of eclipse that will load plug-ins of the workbench.

1. Open the `MANIFEST.MF` file of the contribution project, and on the **Overview** subpage click on **Launch en Eclipse application**

2. Check if the style `style_test.xml` is present in the style preference page and check if the content is valid.

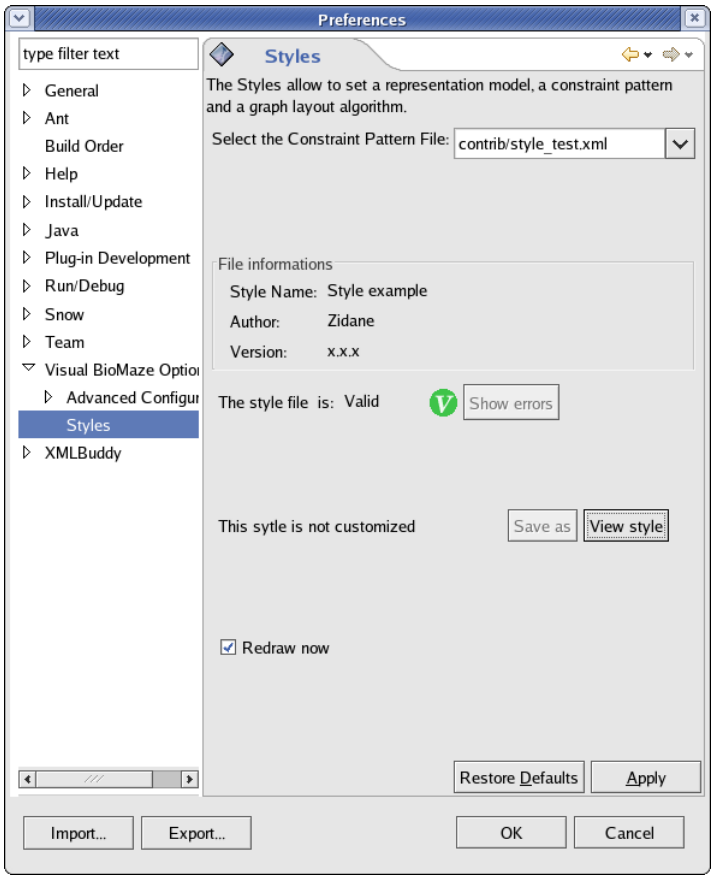



Figure 9.20: The contribution is style is well present and valid.

*Note:* When the contribution is under development, the user can also directly load the contribution without launching the Run-time workbench. Just select the contribution project, right click and select **load contributions**. This will load the entire contribution project in the configuration directory of Visual BioMaze.

## 9.10 Packaging the contribution

1. Select the contribution project
2. Click on right button of the mouse and select **Export**
3. Select **Archive file** and press **Next**
4. Check the contribution project `org.eclipse.vbm.contribution.example`
5. Browse the file system in order to choose where the archive will be created
6. Press **Finish**

Eclipse has then created the archive file in the file system. The contribution plug-in is exported! The only thing to do is now to unzip the archive file in the plug-in directory of Eclipse. Restart Eclipse, and now the contribution plug-in is installed. If the user wants to distribute the contribution plug-in, he can provide the archive file and it is all !

# Chapter 10

# Implemented algorithms

This section describes the different algorithms implemented by the Laboratory of Computer and Network Engineering. Such algorithms receive a graph $G = (V, E)$, where $V$ denotes the set of nodes and $E$ the set of edges composed of pairs of nodes $(u, v)$ such that $u, v \in V$. The goal is to set the position of each node in the drawing area.

## 10.1 The Layered algorithm or Sugiyama algorithm [21, 24]

The idea is to place nodes on parallel layers, and edges are drawn as straight lines between two consecutive layers. The algorithm is split into three phases.

1. Layer assignment: compute a layering of the graph via topological properties od the nodes, i.e., assign all nodes to disjoint subsets $L_1$, $L_2$, ..., $L_m$ calling layers.

2. Minimize crossing: for each layer $L$, determine permutations of the nodes in $L$ with the aim of obtaining less crossing. The problem is here reduced to the double-layer edge crossing minimization.

3. Coordinate assignment: Transform the topological layout of the second phase into geometric layout by assigning to each $v \in V$ its $y$ and $x$ coordinates.

### 10.1.1 Phase 1: Layer assignment

#### Goal

Assign a layer $L_i$ to each node $v \in V$, in a graph $G = (V, E)$.
We need: for each node $v \in V$

- $indeg(v) = |\{e = (u, v) \in E \mid u, v \in V\}|$: the number ingoing edges in $v$
- $outstar(v) = \{e = (v, u) \in E \mid u, v \in V\}$: outgoing edges

#### Assignment algorithm

1. Select the set $S = \{v_i \in V \mid indeg(v_i) = 0 \ And \ v_i \in V\}$ representing the nodes not having any ingoing edges.

2. Assignment of the same layer $L_i$ for all nodes of $S$. The nodes entering in the layer $L_i$ by a FIFO policy. For each node $v_i \in S$, delete the $outstar(v_i)$

$$L(v_i) = L_j \tag{10.1}$$
$$delete(outstar(v_i)). \tag{10.2}$$

3. $j = j + 1$, and return to (1) while $\exists\ v_i \in V \mid indeg(v_i) = 0$

**Moving nodes**

In biochemical graphs, it is often the case that one node is only connected to another one, i.e. $outdeg(v) = 1, indeg(v) = 0$. This is typically the case for genes or compounds. The problem with such nodes is that they stay in first layer (because $indeg(v) = 0$). We have to transfer these nodes near to their next nodes.
$\forall (u, v) \in E, if L(v) > L(u) + 1 \Rightarrow L(u) = L(v) - 1$
Where $L(v)$ is the layer assigned to the node v.

### 10.1.2 Phase 2: Crossing minimization

**Goal**

Minimizing the edge crossing, but this problem is NP-hard, even between two consecutives layers. We apply a *layer by layer sweep* algorithm. Starting from initial configuration of the nodes on each layer, some heuristics consider pairs of layers $(L_{fixed}, L_{free}) = (L_1, L_2),\ (L_2, L_3),\ (L_4, L_5),\ ...,\ (L_{n-1}, L_n)$ and try to determine the permutation to apply to each free layer for reducing the number of edge crossing with its previous layer. Thus such methods reduce the problem to the *2-layer crossing minimization problem* [9]. We need for each node $v_i \in L_i$:

- $indeg(v_i)$ and $pos(v_i)$ in $L_i$

**The 2-layer crossing minimization**

We consider a subgraph $G_{NS}$ of two consecutive layers, the northern and the southern. We can write $G_{NS} = (V, E)$ with $V = N \cup S$ and such that $\forall e = (u, v) \in E|\ u \in N\ and\ v \in S$.

Given a permutation $\langle n_1, n_2, n_3, ..., n_p \rangle$ of all $n_i \in N$, we search a pemutation $\langle s_1, s_2, s_3, ..., s_k \rangle$ of all $s_i \in S$ in according to some aesthetic criteria like here the less number of edge crossing. For finding this permutation some heuristic methods exist and provide good results. In our implementation we used the barycenter heuristic

$$barycenter(v) = \frac{1}{indeg(v)} \sum_{n_i \in instar(v)} i. \tag{10.3}$$

This method consists in arranging permutations according to the barycenter order. If we have a permutation $\langle s_1, s_2, s_3 \rangle$ and we found the barycenter values $\langle 3, 1, 2 \rangle$, the final permutation will be $\langle s_2, s_3, s_1 \rangle$

**Algorithm**

1. $\forall n_i \in N$:
    - Assign a label representing the node relative position in the northern layer
2. $\forall s_i \in S$:
    - Compute $barycenter(s_i)$
3. Sort by insertion the diffrent barycenters
4. Order the South permutation according to the barycenter sort

### 10.1.3 Phase 3: Coordinate assignment

**Goal**

Once the topologic order is fixed for all nodes, we can assign the position in a coordinate system. The $y$ coordinate is fixed for each layer, while the $x$ coordinate depends of the size of the node to draw. This size is defined by the glyph associated to the node.

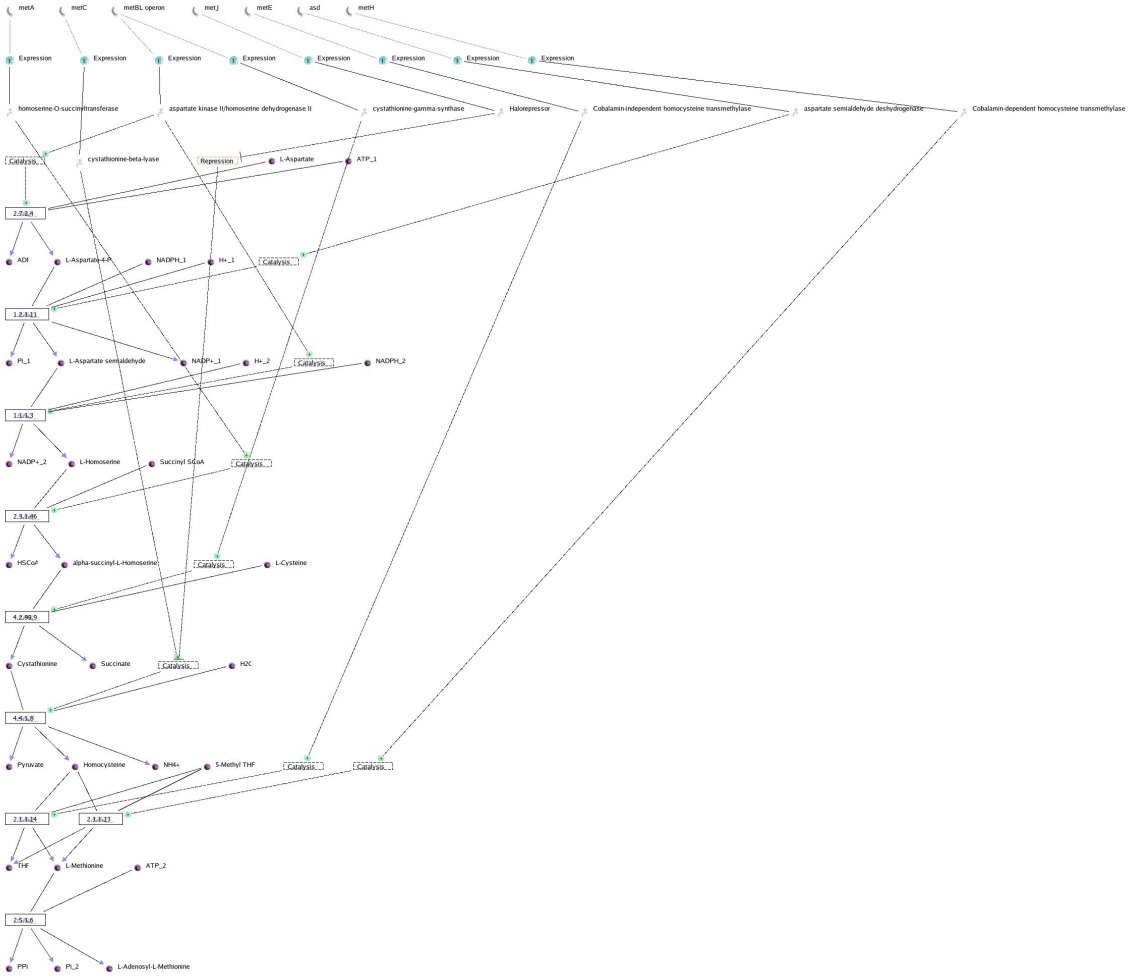



Figure 10.1: The result with the Sugiyama graph layout algorithm.

## 10.2 Force-directed algorithm: Variant of Simon *et al.*

The force-directed algorithms is a family of methods that proposes to consider graphs as a physical system. These algorithms compute the total system energy and try to minimize it by moving nodes in an optimal position. But the global optimization of the energy is an NP-complete problem. The most popular heuristic is the *spring embedder* algorithm. First introduced by [3] and [16], this algorithm simulates a mechanical system in which graph's edges are replaced by springs and nodes are replaced by rings connecting edges incident on a node. From the initial configuration (initial position of each node), the system oscillates until it stabilizes at a minimum-energy configuration.

### 10.2.1 Equations of the problem

The force-directed methods [9] try to compute nodes positions $p_v$ for which the physical system reaches an equilibrium state such that $f(p_v) = 0, \forall v \in V$.Such a state is approximated in practice by an iterative algorithm that, starting from initial (often random) positions $p_v$ for each node $v \in V$, computes $f(p_v) \forall v \in V$ and then updates the position $p_v = p_v + \mu \cdot f(p_v)$, where the step length $\mu$ is a number smaller than one. The expression of the spring is given by

$$f(p_v) = \sum_{e=(u,v) \in star(v)} K_{u,v}(\|p_u - p_v\| - l_{u,v}) 1_{p_u \vec{p}_v}, \tag{10.4}$$

where $K_{u,v}$ is the stiffness and $l_{u,v}$ is the natural length of the spring between $u$ and $v$ such that the spring force between them is proportional to the difference between the $distance(u, v)$ and the natural length of the spring.

Kamada *et al.* describe an interesting variant in [10]. These authors associate a potential energy to each node pair, that are not adjacent, . The consequence is that we consider a graph in which all nodes are connected by springs, the important parameter to consider, is now $K_{u,v}$ and $l_{u,v}$. Kamada al, propose these expression

$$l_{u,v} = L \cdot d_{u,v} \tag{10.5}$$

$$K_{u,v} = \frac{K}{d_{u,v}^2}, \tag{10.6}$$

where $d_{u,v}$ is the shortest path between $u$ and $v$ in the graph, $L$ and $K$ are two constants that need to be adjusted. The Kamada algorithm begins in a initial configuration with a defined potential energy

$$W_{sys} = \sum_{(u,v) \in V} \frac{K_{u,v}}{2} \cdot (\|p_u - p_v\| - l_{u,v})^2. \tag{10.7}$$

We can decompose this expression in

$$W_{sys} = \sum_{(u,v) \in V} \frac{K_{u,v}}{2} \cdot (\sqrt{(x_u - x_v)^2 + (y_u - y_v)^2} - l_{u,v})^2. \tag{10.8}$$

The goal [18] of the spring embedding system concept is to minimize this quantity. Usually it cannot be assumed that there exists a configuration of nodes position that leads to $W_{sys} = 0$.

Instead, Kamada is looking for a local minima with respect to interdependent $x_u, y_u$. The property of a local minimum can be described by

$$\frac{\partial W_{sys}}{\partial x_u} = 0 \tag{10.9}$$

$$\frac{\partial W_{sys}}{\partial y_u} = 0, \tag{10.10}$$

for all $u \in V \mid p_u = (x_u, y_u)$. Since it is to difficult to compute the value of each position for minimizing the energy for the entire system, Kamada tries to decrease the energy by moving nodes one by one until the energy reaches the minimum. Then, in each step, we have to resolve these partial derivates for one node in order to find its optimal displacement. According to the given expression of the potential energy, we can write

$$\frac{\partial W_{sys}}{\partial x_m} = \sum_{j=1}^{n} K_{m,j} \cdot ((x_m - x_j) - \frac{l_{m,j}(x_m - x_j)}{\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2}}) \tag{10.11}$$

$$\frac{\partial W_{sys}}{\partial y_m} = \sum_{j=1}^{n} K_{m,j} \cdot ((y_m - y_j) - \frac{l_{m,j}(y_m - y_j)}{\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2}}), \tag{10.12}$$

for all $m \in V \mid p_m = (x_m, y_m)$ Simon comments this equation in [18] by explaining:

- The change of energy when moving one point has only to consider springs that are connected to it because all other springs remain constant. Therefore, one summation index can be omited.
- The summation of the summand where $m = j$ does not matter because it is 0.

For all points $m \in V$ it is possible to calculate the potential power $\nabla$ with respect to the complete energy $W_{sys}$:

$$\nabla_m = \sqrt{\frac{\partial W_{sys}}{\partial x}^2 + \frac{\partial W_{sys}}{\partial y}^2} = \left| grad(\vec{W}_{sys}) \right| \tag{10.13}$$

If $\nabla_m$ is minimized for all nodes m, we found a local minimum of $W_{sys}$. The iterative algorithm of Simon starts by moving the node that presents the highest $\nabla$.

### 10.2.2 Solving the equations

The most common way to find the zero of a function is the well-known *Newton-Raphson algorithm*. For a node $m$ of coordonate $p_m = (x_m, y_m)$, we are interested by:

1. How far should the point moved in x-direction = $\delta x$ such that $\frac{\delta W}{\delta x} = 0$ ?
2. How far should the point moved in y-direction = $\delta y$ such that $\frac{\delta W}{\delta y} = 0$ ?

Simon shows in [18] that using the first-order Taylor progression, the Newton-Raphson algorithm can be reduced to $f(x + \delta x) = f(x) + f'(x) \cdot \delta x$, where $\delta x$ describes the 2-D vector for which the node x has to be moved with respect to one direction. Because of the assumption

$f(x+\delta x)=0$, it follows that: $-f(x)=f'(x)\cdot\delta x$. The resulting complete move can be calculated by the summation of partial moves, given by

$$-f(x,y)=\frac{\partial f}{\partial x}\delta x+\frac{\partial f}{\partial y}\delta y. \tag{10.14}$$

Once the $\delta x$ and the $\delta y$ are known, we move m such that $x_m=x_m+\delta x$ and $y_m=y_m+\delta y$. If $f(x,y)=\frac{\partial W_{sys}}{\partial x_m}$ and $f(x,y)=\frac{\partial W_{sys}}{\partial y_m}$ we get the following equations

$$-\frac{\partial W}{\partial x_m}=\frac{\partial^2 W}{\partial^2 x_m}\cdot\delta x+\frac{\partial^2 W}{\partial x_m\partial y_m}\cdot\delta y \tag{10.15}$$

$$-\frac{\partial W}{\partial x_m}=\frac{\partial^2 W}{\partial y_m\partial x_m}\cdot\delta x+\frac{\partial^2 W}{\partial^2 y_m}\cdot\delta y. \tag{10.16}$$

The partial derivatives can be writren as follows

$$\frac{\partial^2 W}{\partial^2 x_m}=\sum_j 1-\frac{l_{m,j}\ (y_m-y_j)^2}{((x_m-x_j)^2+(x_m-x_j)^2)^{3/2}} \tag{10.17}$$

$$\frac{\partial^2 W}{\partial x_m\partial y_m}=\sum_j \frac{l_{m,j}\ (y_m-y_j)\cdot(x_m-x_j)}{((x_m-x_j)^2+(x_m-x_j)^2)^{3/2}} \tag{10.18}$$

$$\frac{\partial^2 W}{\partial^2 y_m}=\sum_j 1-\frac{l_{m,j}\ (x_m-x_j)^2}{((x_m-x_j)^2+(x_m-x_j)^2)^{3/2}} \tag{10.19}$$

$$\frac{\partial^2 W}{\partial y_m\partial x_m}=\sum_j \frac{l_{m,j}\ (y_m-y_j)\cdot(x_m-x_j)}{((x_m-x_j)^2+(x_m-x_j)^2)^{3/2}}. \tag{10.20}$$

If we assume $A=\frac{\partial^2 W}{\partial^2 x_m}$; $B=\frac{\partial^2 W}{\partial x_m\partial y_m}$; $C=\frac{\partial^2 W}{\partial y_m\partial x_m}$; and $D=\frac{\partial^2 W}{\partial^2 y_m}$ we have the solutions

$$\delta x=\frac{1}{A}\cdot\frac{\partial W}{\partial x_m}-\frac{\partial W}{\partial y_m}\cdot\frac{B}{AD-CB}+\frac{\partial W}{\partial x_m}\cdot\frac{BC}{A(AD-CB)} \tag{10.21}$$

$$\delta y=\frac{A\cdot\frac{\partial W}{\partial y_m}-C\cdot\frac{\partial W}{\partial x_m}}{D-CB}. \tag{10.22}$$

### 10.2.3 The Simon *et al.* contribution [18]

The major contribution of Simon's team was the introduction of *peripheric rules* on the spring embedder algorithm. Simon explains in [18] that the necessity of these rules comes from the properties of the Newton-Raphson algorithm and the properties of the spring embedder algorithm.

#### Properties of the NRA

The NRA does necessarily converge. Simon gives an example with the function $f(x)=-x^4+6x^2+11$, The algorithm oscillates between -1 and 1, and never identify the two zero points +2.7335, -2.7335. We have to protect our algorithm to a non-terminating program. Simon proposes to use the *simulated annealing* [12](SA). It uses an analogy with thermodynamics

and most particularly the process of metals cooling and annealing. The principle is quite simple, at high temperature the molecules of liquid metal move freely with respect to one another. If metal cools down the mobility of molecule decreases, and if the cooling process is slow enough the metal molecules are able to order themselves and form a pure crystal with a minimum energy. The variation of energy in function of temperature is carried out by the Boltzman distribution.

Simon transfers the metal's molecules behaviour for controlling the termination of the NRA algorithm. At beginning the energy computing by the NRA is allowed to vary significantly, but with progressive iterations, the annealing asks that the energy decreases and that the range of allowed variations decreases. If the algorithm converges in the prescribed time, the SA method does not matter for calculation, otherwise the SA stop the NRA.

**Properties of the spring embedder**

Another property is inherent to the concept of spring embedder: The algorithm calculates only local minima, i.e., it tries always to minimise the energy from a starting point, therefore, a lower energy that requires increasing the complete energy is not reached. In addition if a node representing the most potential power is moved such that the complete energy is down of the local minima valley, each move will increase the energy so, the node will be replaced such that the energy is down of the valley. We have a deadlock. Simon propose to *shake* randomly this node in this situation, in hope that could be better.

### 10.2.4 Our contribution

Our contribution is limited to two important aspects. We start from the observation that the algorithm is highly sensible to the initial configuration, and in the Simon algorithm the initial configuration is randomly defined. This sensibility can be explained by the fact that if a bad initial configuration is given (bad refers to a configuration that needs a lot of node displacements) the algorithm has no time to converge. But even it converges to physical-acceptable minimum energy, the visual flow of the process visualized is not outlined, and we cannot read easily such graphs. Our solution is to apply a layered Sugiyama-style algorithm for setting the initial configuration. Hence the layered algorithm sorts the different consecutive stages of the biochemical process on layers, and after the spring embedder with SA arrange the different nodes for minimizing energy. The main drawback of the layered algorithm then disappears. In such algorithm the nodes with no incident edge are drawn in the first layer, and the nodes like genes stay in this layer and decrease the visibility. The spring embedder SA algorithm provides a way to arrange the position of such nodes near of involved nodes. We have compared the graph of the Methionine with different initial position policy. We have taken a special representation model where all nodes have the same glyph, and we have considered the methionine biosynthesis pathway as graph test. We have observed the variation of the system energy:

- Random policy: from 3E7 to 5E6 in 300 iterations
- Circular policy : from 2E7 to 5E6 in 300 iterations
- Sugiyama policy : from 2E7 to 2E6 in 180 iterations

We have observed that for graphs with a depth smaller than 3, the Sugiyama-style prepositions is not adapted. For this observation we have taken a set of graphs with a depth between one and three and we have compared the different energy level with the different initial positions. As conclusion, we provide a Sugiyama-style initial positions for graphs having a depth greater then three, and for the others we provide a circular initial position around a circle with a radius equals to the mean edge length.

Our second contribution coming from the shake function. Instead of moving randomly the nodes all over the graph, we move the node randomly around its position. This modification may seem no efficient, but the result is that the node is shaking about four times to attempt a minimal power. This behaviour can be explained by the fact that, once the node is shaked around, it stays nearly of the minimal energy but with a sufficient shake move, it can go out of the deadlock exposed in previous section.

Finally, we implemented a relocate function that finds the two nodes with respectively the minimum x-coordinate and the minimum y-coordinate, and apply a translation of all nodes as follows

$\forall u \in V$

$$x_u = x_u - \delta x \tag{10.23}$$
$$y_u = y_u - \delta y, \tag{10.24}$$

where $\delta x = x_{min} - mean_{width\ glyph}$ and $\delta y = y_{min} - mean_{height\ glyph}$.

### 10.2.5 Results

We first apply our version of the Simon algorithm on the methionine pathway showed by Figure 10.1. Figure 10.2 shows the potential energy decreasing with iterations, while Figure 10.3 shows the maximum power $\nabla$ as a function of iterations.

The main drawback of force-directed method is the choice of constants (K and L) that play key role in the algorithm efficiency.

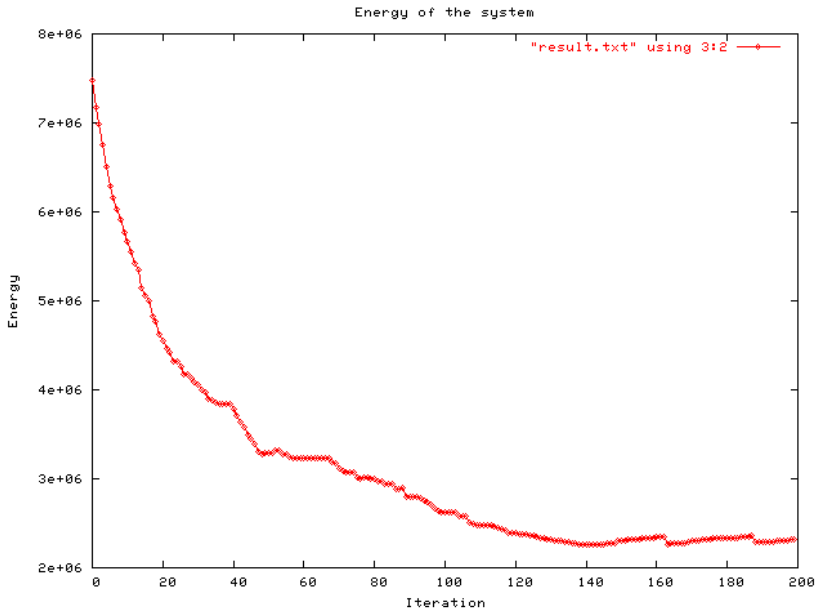



Figure 10.2: The potential energy of the system $W_{sys}$ decreasing with the iterations

### 10.2.6 The modified Spring-embedder controled by simulated annealing algorithm

**Algorithm 10.2.1:** SIMON($IGraph\ graph$)

$compute\ initial\ position\ (Simon:\ random,\ Skhiri:\ Sugiyama)$
$compute\ shortestWay\ d_{u,v}\ \forall\ u,v \in V$
$compute\ parameters\ l_{u,v}\ K_{u,v}$
$compute\ W_{sys}$
$compute\ \nabla u\ \forall\ u \in v$
$p_i = find\ Maximum\ Power(graph)$
$do$
  {
  $while(\ power(p_i) > epsilon\ AND\ annealing(p_i))Do$
  {
    $compute\ \delta x,\ \delta y$
    $move\ node(p_i)$
  }
  $Restore\ p_i\ |power(p_i) = min\ of\ the\ computed\ p_i\ in\ the\ inner\ loop$
  $p_j = find\ Maximum\ Power(graph)$
  $//stoped\ by\ the\ annealer\ and\ p_i\ stay\ with\ the\ max\ power$
  $if((p_j == p_i)\ AND\ (power(p_i) > epsilon)\ shake(_i)$
$}while(annealing(W_{sys})\ AND\ power(p_i) > epsilon)$

The $annealing(W_{sys})$ controls the energy variation as function of temperature. This

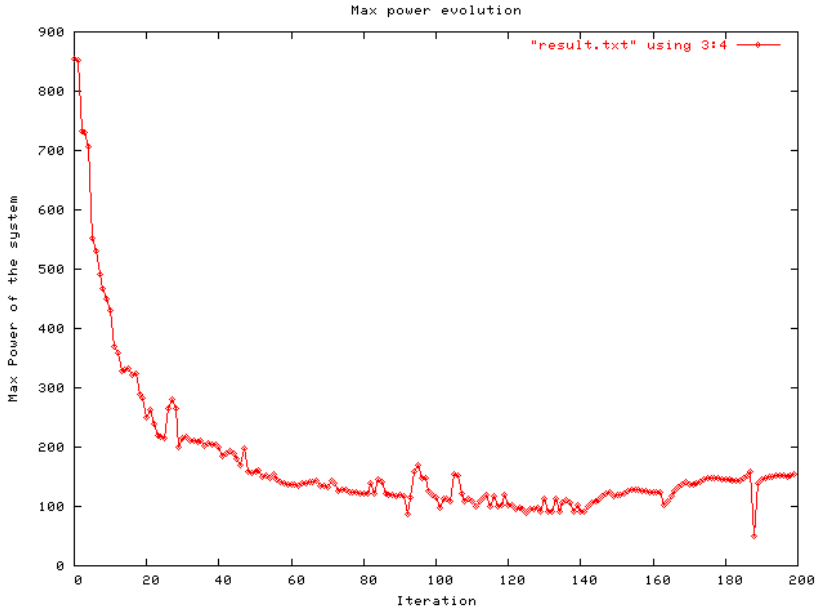



Figure 10.3: The maximimum $\nabla$ function of iterations

variation is not allowed to come over $\Delta W_t$

$$\Delta W < \Delta W_t = \frac{W_0}{2 \cdot t_0} \cdot t, \tag{10.25}$$

where $W_0$ is the initial energy, and $t_0$ is the initial temperature, 100 K in our implementation. A $t = t_0$, th energy is allowed to vary about its half, and $\Delta W$ decreases with $t$.

### 10.2.7 Shortest path finding

As already said, we need to find the for each pair $(u, v)$ $inE$ the shortest way $d_{u,v}$. This research take a lot of time and no really efficient algorithm exists. We have implemented the *Dijkstra's algorithm* [19]. The problem can be enounced as follows: We research the shortest way between:

- a given start node and a given target node
- one node and all others
- $n$ start nodes and $m$ target nodes

The Dijkstra's algorithm is used to calulate the shortest path in the well-known routing algorithm OSPF( Open Shortest Path Finding).

**The algorithm**

**Algorithm 10.2.2:** DIJKSTRA($INode\ v$)

$Initialization \quad cost(v) = 0$
$cost(x) = \infty\ \forall x! = v$
$\Pi \leftarrow \oslash$
$while(size(\Pi)! = size(V))$
{
$x = find\ max\ cost\ node \mid x \notin \Pi$
$\Pi \leftarrow \Pi \cup \{x\}$
$Y = instar(x) \cup outStar(x)$
$For\ y \in Y$
$If(cost(x) + 1 < cost(y))$
$cost(y) = cost(x) + 1$
}

We start from one node $v \in V$, we initialize $cost(v) = 0\ AND\ cost(u) = \infty\ \forall u \in V_{/\{v\}}$, $\Pi$ is the set of shortest path already found. The algorihtm turn until this set is equal to the initial set of node $V$. When the algorithm stops, we know all shortest paths from v to all other nodes. We have to run this algorithm for all nodes of $V$.

## 10.3 Force-directed algorithm: Spring with repulsion forces

Some authors [6, 8] add a repulsion force between non-adjacent nodes, compute the resultant on each node and then update the position $p_v = p_v + \mu \cdot f(p_v)$. We have not found papers where authors compute the total energy of the system taking into account the repulsion force. In this section we will compute total energy of the system and the partial derivatives of the first and second order. With these expressions we are able to use the Variant of Simon *et al.* for assigning the nodes position.

### 10.3.1 Equation of the repulsion force and energy

We consider a repulsion force between all nodes, even not adjacent nodes. The force is given by

$$f(p_v) = \sum_{u,v \in V} \frac{K_{rep\{u,v\}}}{\|p_u - p_v\|} \cdot -1_{p_u \vec{p}_v}. \tag{10.26}$$

The work of this force is given by the integral of the force on the displacement. The total energy is given by this expression

$$W_{rep} = \sum_{(u,v) \in V} \frac{K_{rep\{u,v\}}}{\|p_u - p_v\|} \Leftrightarrow W_{rep} = \sum_{(u,v) \in V} \frac{K_{rep\{u,v\}}}{\sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}}. \tag{10.27}$$

Now we have to compute the partial derivatives of the first-order $\frac{\partial W_{rep}}{\partial x}$ and $\frac{\partial W_{rep}}{\partial y}$

$$\frac{\partial W_{rep}}{\partial x_m} = \sum_{j=1} \frac{K_{rep\{m,j\}} \cdot (x_m - x_j)}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^3} \tag{10.28}$$

$$\frac{\partial W_{rep}}{\partial y_m} = \sum_{j=1} \frac{K_{rep\{m,j\}} \cdot (y_m - y_j)}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^3}, \tag{10.29}$$

and the expressions of the partial derivative of the second order

$$\frac{\partial^2 W_{rep}}{\partial x_m^2} = \sum_{j=1} \frac{-K_{rep\{m,j\}}}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})} + \frac{3 \cdot K_{rep\{m,j\}} \cdot (x_m - x_j)^2}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.30}$$

$$\frac{\partial^2 W_{rep}}{\partial y_m^2} = \sum_{j=1} \frac{-K_{rep\{m,j\}}}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})} + \frac{3 \cdot K_{rep\{m,j\}} \cdot (y_m - y_j)^2}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.31}$$

$$\frac{\partial^2 W_{rep}}{\partial x_m \partial y_m} = \sum_{j=1} \frac{3 \cdot K_{rep\{m,j\}} \cdot (x_m - x_j) \cdot (y_m - y_j)}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.32}$$

$$\frac{\partial^2 W_{rep}}{\partial y_m \partial x_m} = \sum_{j=1} \frac{3 \cdot K_{rep\{m,j\}} \cdot (y_m - y_j) \cdot (x_m - x_j)}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}}. \tag{10.33}$$

### 10.3.2 Total energy of the system

The expressions of the $\delta x$ and $\delta y$ move are identical, the expression of the energy and then $\nabla W$ changes

$$\frac{\partial^2 W_{rep}}{\partial x_m^2} = \sum_{j=1} 1 - \frac{l_{m,j}\ (y_m - y_j)^2}{((x_m - x_j)^2 + (x_m - x_j)^2)^{3/2}} + \frac{-K_{rep\{m,j\}}}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})} \tag{10.34}$$

$$+ \frac{3 \cdot K_{rep\{m,j\}} \cdot (x_m - x_j)^2}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.35}$$

$$\frac{\partial^2 W_{rep}}{\partial y_m^2} = \sum_{j=1} 1 - \frac{l_{m,j}\ (x_m - x_j)^2}{((x_m - x_j)^2 + (x_m - x_j)^2)^{3/2}} + \frac{-K_{rep\{m,j\}}}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})} \tag{10.36}$$

$$+ \frac{3 \cdot K_{rep\{m,j\}} \cdot (y_m - y_j)^2}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.37}$$

$$\frac{\partial^2 W_{rep}}{\partial x_m \partial y_m} = \sum_{j=1} \frac{l_{m,j}\ (y_m - y_j) \cdot (x_m - x_j)}{((x_m - x_j)^2 + (x_m - x_j)^2)^{3/2}} + \frac{3 \cdot K_{rep\{m,j\}} \cdot (x_m - x_j) \cdot (y_m - y_j)}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.38}$$

$$\frac{\partial^2 W_{rep}}{\partial y_m \partial x_m} = \sum_{j=1} \frac{l_{m,j}\ (y_m - y_j) \cdot (x_m - x_j)}{((x_m - x_j)^2 + (x_m - x_j)^2)^{3/2}} + \frac{3 \cdot K_{rep\{m,j\}} \cdot (y_m - y_j) \cdot (x_m - x_j)}{(\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2})^{5/2}} \tag{10.39}$$

### 10.3.3 Results

We have chosen a value for $K_{rep\{m,j\}} = K_{m,j} \cdot C$, where $C$ is a constant whose value is 1 for any non-adjacent nodes and 40 for two adjacent nodes. This disposition provides a graph

where the connected elements are spread each other, and outline a main direction in pathway. The fact that the two constant forces are proportional is also important, it allows to forces to distribute it equally.

The results behaves fairly good (Figure 10.4: in average after 150 iterations, the energy stabilizes and the algorithm provides nice graphs. As previously said, the initial position policy depends on the graph's depth and the most difficult problem remains the evaluation of constants.

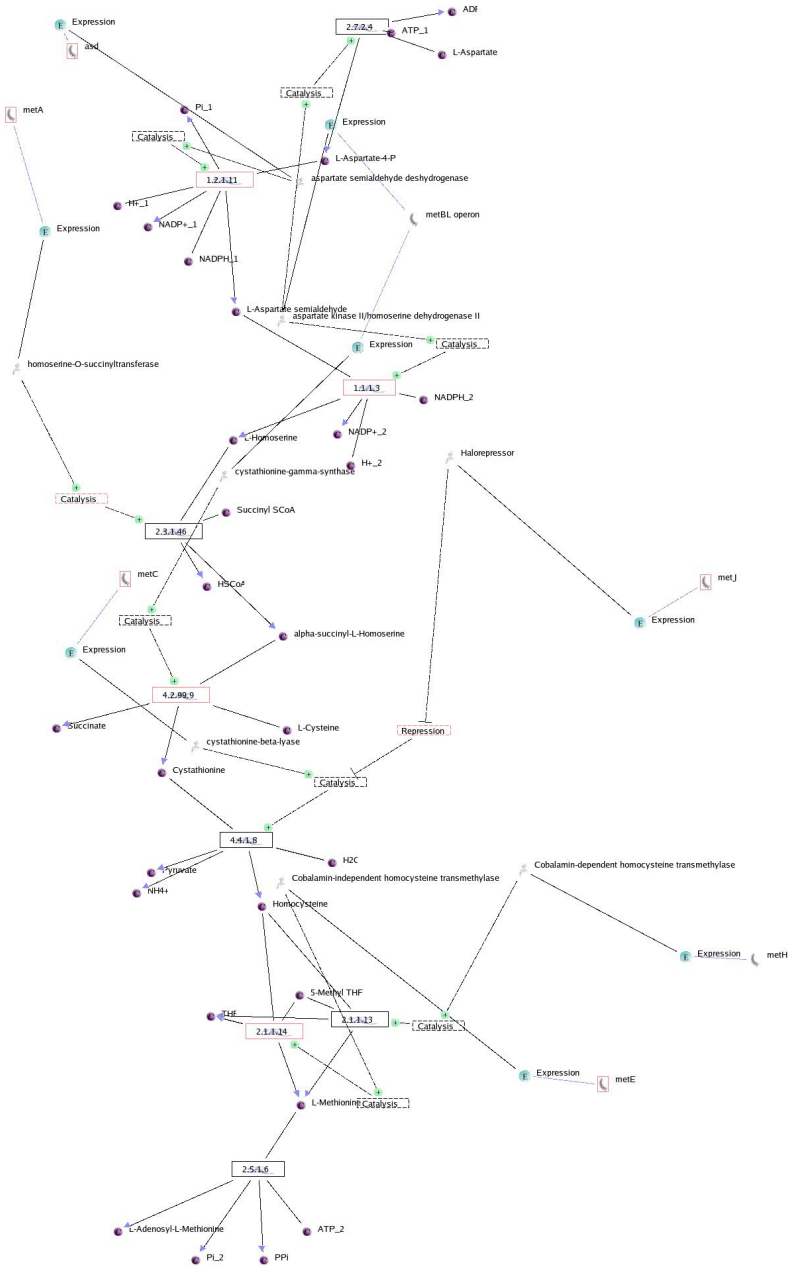



Figure 10.4: The result of the modified force directed algorithm algorithm on the regulated metabolic pathway of the Methionine. The red elements are shaked nodes.

## 10.4 Constrained Simple Compound Graph Layout (CSCGL)

### 10.4.1 Introduction

Graph drawing has emerged in recent years as an important area in computer science. Graphs provide a way to structure the information presented to users, by showing objects and relations between them. Automatic graph drawing tools allow to generate drawings on demand. More often than not, the output of this kind of tool can be improved manually. The reason is that graphs usually have application-specific semantics known to the people working with the graph but not known to the graph drawing algorithms. Most of the graph layout algorithms are designed to produce drawings in accordance with æsthetics criteria, not in accordance with semantics. Therefore no graph layout algorithm can be (semi-)automatically adapted to a new application domain. How can we express the drawing knowledge of a particular domain? In which way the graph layout algorithm must consider it? These are the questions that we have addressed in our research.

As the definition of a universal algorithm that is able to adapt itself to any application domain is quite complex, we propose a generic algorithm suited for bioinformatics and more particularly for the representation of biochemical networks. Although biochemical networks are simple di-graphs, they convey a rich semantics. Since biologists (people having the knowledge about graph semantics) have drawn by hand biochemical graphs [26] for a long time, the graph layout algorithm must take into account this knowledge. In addition, new databases such as the aMAZE project [14] provide tools allowing to extract multiple kinds of biochemical networks in a single graph.

In this section we describe the *Constrained Simple Compound Graph Layout* (CSCGL) algorithm, a graph layout algorithm that is generic, i.e., it can be adapted to any type of biochemical networks (signal transduction, metabolic pathways, regulation networks, gene-protein interactions graphs, etc.). Since the knowledge of how to draw graphs is expressed by users as graphical constraints, the CSCGL algorithm is able to produce drawings according to the specific domain semantics. This algorithm was implemented in the Visual BioMaze framework, a generic viewer of graphs developed under the Eclipse platform [1].

The structure of this chapter is as follows. In Section 2 we describes the different steps composing the CSCGL algorithm, while Section 3 shows experimental results of our algorithm. Finally, Section 4 describes our conclusions and future perspectives.

### 10.4.2 The CSCGL Algorithm

Although it is generic, the CSCGL algorithm was adapted for biochemical networks. As described in [28], the visualization of such networks involves several issues. The most important one is that the representation depends of the type of biochemical network. For example, a metabolic pathway follows a main vertical direction outlining the process. Also, in [27] it is suggested to represent biochemical pathways vertically and its regulation horizontally. From a graph layout point of view this raises the issue of representing a graph with two (or more) different directions according to the subgraph type.

In [23] is proposed a force-directed method that applies different magnetic fields in the desired direction of the graphs. Arcs are magnetic-proof or not depending on its type. Although this method provides a way to apply a preferred direction to a set of arcs, it does not lead to the best results for outlining a main flow direction and requires an evaluation of the

constants involved in algorithm. In addition the computing time of such method increases significantly with the number of nodes.

Our approach is to allow the user to define graphical constraints stating which kind of subgraphs must be placed along a specific direction. Our algorithm is based on *compound graphs*. Such graphs are represented by an inclusion-directed graph and an adjacency-directed graph. Thus, a compound graph is defined by a triple $G = (V, F, A)$ where $V$ is the set of nodes, $F$ is the set of inclusion arcs, and $A$ is the set of adjacency arcs. An arc $(u, v) \in F$ means that $u$ is included in $v$, while $(u, v) \in A$ means that the next node in the graph after $u$ is $v$. The set of nodes is partitioned into $B$, the base nodes, i.e., the leafs of the inclusion tree, and $S$ the compound nodes.

Our algorithm uses *simple compound graphs*, i.e., graphs verifying the following conditions:

- The inclusion graph defined by $F$ has a depth of 1.
- $\nexists\, (v, w) \in F$ such that $v \in B$ and $w \in S$
- $\nexists\, (v, w) \in F$ such that $v \in S$ and $w \in B$

This algorithm is also called *constrained* since the graph is dynamically built according to graphical constraints. Given a node $u$, we denote by $indeg(u)$ the number of arcs arriving to the node $u$. Similarly, $outdeg(u)$ denotes the number of arcs starting from node $u$. A node $u$ where $outdeg(u) = 0$ is called a *leaf*; it is called a *root* if $indeg(u) = 0$.

### Algorithm Description

Usual algorithms for drawing compound graphs are composed of four steps, similar to those for general directed graphs (e.g., [20] and [22]). The CSCGL algorithm is composed of 7 phases:

- Phase 1: Cycle management
- Phase 2: Compound graph construction
- Phase 3: Global layer assignment
- Phase 4: Compound layer assignment
- Phase 5: Edge-crossing minimization
- Phase 6: Expansion of compound nodes
- Phase 7: Coordinate assignment

The phases 3, 5, and 7 are similar to those used for compound graphs. The other phases are original.

### Phase 1: Cycle Management

Unlike traditional methods using by layered algorithms such as *backward edge search* [7], we do not need to obtain an acyclic graph. Instead, we enumerate the nodes composing each cycle in order to draw them according to graphical constraints. For example the graph given in Figure 10.5 (a) has 3 cycles. Such graph represents a typical biochemical network. The

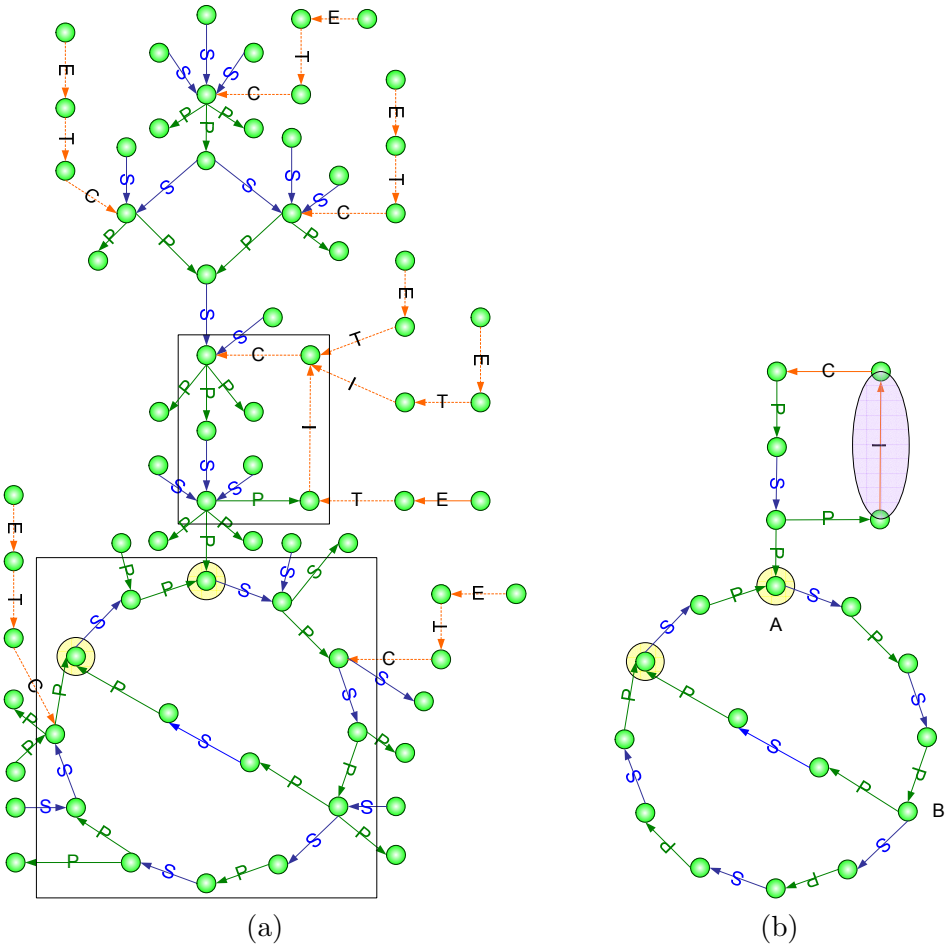



Figure 10.5: a) Initial graph. The graph contains 3 cycles. The one inside the top rectangle is a simple cycle, while the two cycles inside the bottom rectangle compose a topological cycle. (b) Graph obtained after the step of cycle detection. The junction nodes are marked with a circle. The arc inside the ellipse will be inverted.

arcs are typed as follows: E(xpression), T(ranslation), C(atalysis), P(roduct), S(ubstrat), and I(nhibition). Although the nodes are also typed, this is not shown in the figures to simplify the presentation.

In order to enumerate all possible cycles, we developed a new cycle management algorithm adapted from the well-known *Greedy-Cycle-Removal* algorithm [4] that consists in two steps described next.

*Cycle Detection*

We remove iteratively all leafs of the graph, i.e., at each iteration we remove all nodes $u \in V$ such that $outdeg(u) = 0$. Similarly, we remove iteratively all roots of the graph, i.e., all nodes $u \in V$ such that $indeg(u) = 0$. If the resulting set $V'$ is empty the graph has no cycle. Figure 10.5 (b) shows the result of this step.

*Cycle Isolation*

The aim of this step is to provide a set of node lists, each one corresponding to a cycle. In order to find all cycles, including the inner ones (the topological cycle in Figure 10.5 has an inner

cycle), the algorithm finds the list of *junction nodes*, defined as $\{u \mid u \in V,\ indeg(u) > 1\}$. Figure 10.5 (b) shows the junction nodes marked with a circle.

The cycle isolation algorithm starts by finding the cycles that can be reached from all junction nodes. It uses backtracking to obtain all cycles that can be reached from a particular junction node. For example, in Figure 10.5 (b), starting from the junction node A, this method first finds the inner cycle, and then backtracks to node B to find the outer cycle.

The cycles found are then removed from the graph. In our example of Figure 10.5(b) the remaining graph is composed of the upper cycle. As can be seen, there may still exist simple cycles in which there are no junction nodes. The cycle isolation algorithm then continues by finding such simple cycles using a depth-first search.

### Phase 2: Compound Graph Construction

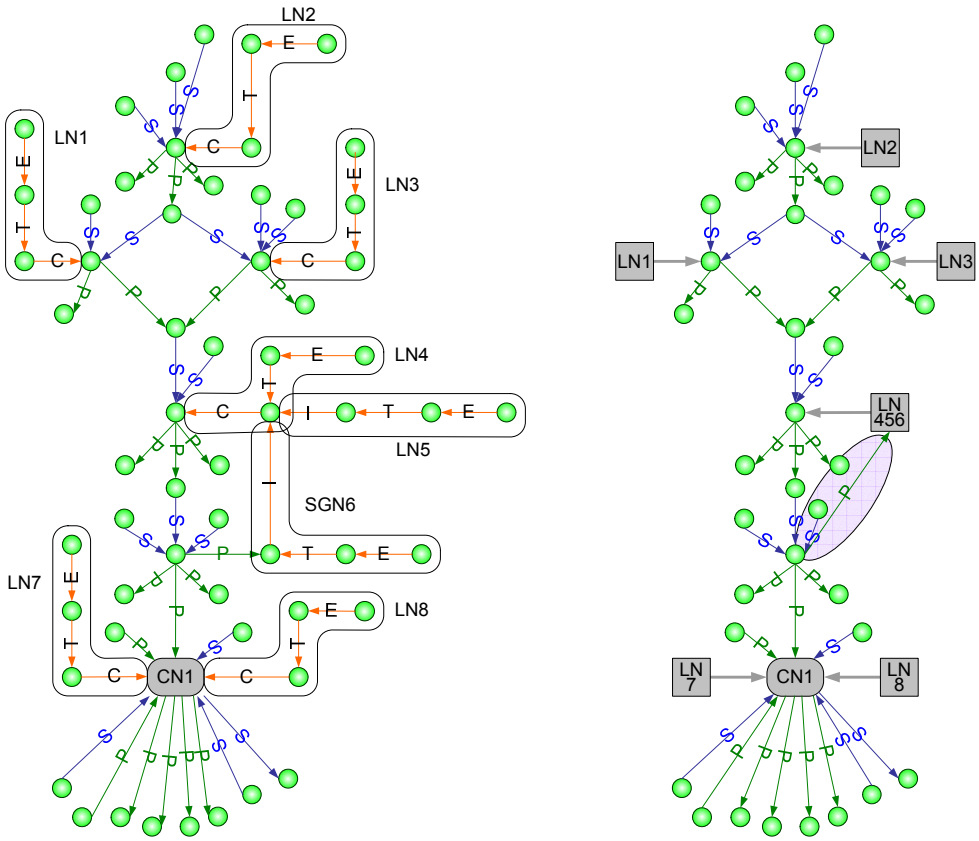



Figure 10.6: Phase 2: (a) Initial graph where subgraphs corresponding to the linear constraints are identified. b) Graph obtained by replacing these subgraphs with l-nodes.

This phase takes into account the graphical constraints defined by the user. In our algorithm we have currently implemented two kinds of constraints: circular constraints and linear constraints. These are explained next.

Recall that the result of Phase 1 is a set of lists, each one containing the ordered list of nodes composing a cycle. We distinguish two kinds of cycles: *topological cycles* that must be

visualized in a circular way, and *simple cycles* that may be represented as usual nodes. For example, as shown in Figure 10.5 (a), in biochemical networks metabolic cycles are visualized as circles (bottom rectangle), while cycles in genetic regulation are not represented in this way (top rectangle).

*Circular constraints* determine whether a cycle is topological or simple. They define the set of allowed arc types for topological cycles. These cycles are replaced by a compound node called *circular node* (or c-node). Figure 10.6 (a) is obtained by replacing in Figure 10.5 (a) the topological cycle by a compound node CN1. For simple cycles a depth-first search algorithm [25] can be used to find the minimal backward edges, i.e., a minimal set of edges to be reversed that makes the cycle disappear, e.g., the edge shown inside an ellipse in Figure 10.5 (b). Since for æsthetics reasons not all edges can be inversed, another type of graphical constraint specifies the allowed backward edges.

*Linear constraints* define a list of arc types that must be drawn in a linear way according to a particular direction. In our example of Figure 10.5 (a) there are two of such constraints: ⟨E, T, C⟩ and ⟨E, T, I⟩, referring to arc types Expression, Translation, Catalysis, and Inhibition. In biochemical networks such subgraphs are usually represented horizontally. The algorithm replaces all subgraphs matching with a linear constraint (excepted the last edge) by a compound node called *linear node* (or l-node). This is shown in Figure 10.6 (b). Notice that when several l-nodes intersect, as is the case of LN3, LN4, and LN5, they may be merged into one l-node, called LN345 in the figure.

Notice also that when replacing a subgraph by a compound node, some cycles will be introduced. In the example of Figure 10.6 (b) the node LN456 is involved in a cycle. In this case we proceed as for simple cycles and find the minimal backward edges allowing to remove such cycles. In the figure it is the arc shown inside the ellipse.

**Phase 3: Global Layer Assignment**

This phase is strictly the same as the layer assignment of the Sugiyama algorithm described in [21, 24]. The only difference is that the set of nodes for which we apply the layer assignment are the base nodes and the compound nodes. Figure 10.7 shows the result for our example.

**Phase 4: Compound Layer Assignment**

In this phase we assign (local) layers to compound nodes. Recall that our compound nodes are l-nodes corresponding to linear constraints and c-nodes corresponding to circular constraints.

Layer assignment for l-nodes is done as in the previous phase, excepted that we start from the leafs of the graph, while in the previous phase we started from the roots of the graph. The reason for this is that such subgraphs must follow a direction (e.g., horizontal) which is different from the one used for the whole graph (typically vertical).

The compound layer assignment for c-nodes uses a traditional cyclic inner layout. Each node receives a relative layout with respect to its position in the cycle. Figure 10.8(b) shows such a layer assignation.

As result of this step, to each compound node is assigned an internal topology. Therefore, each node will have two important parameters: the depth and the breadth. These are used in Phase 6 when expanding compound nodes.

As already said, in our current implementation we only allow linear and circular graphical constraints. Other types of graphical constraints (e.g., hierarchical) may be considered. Each

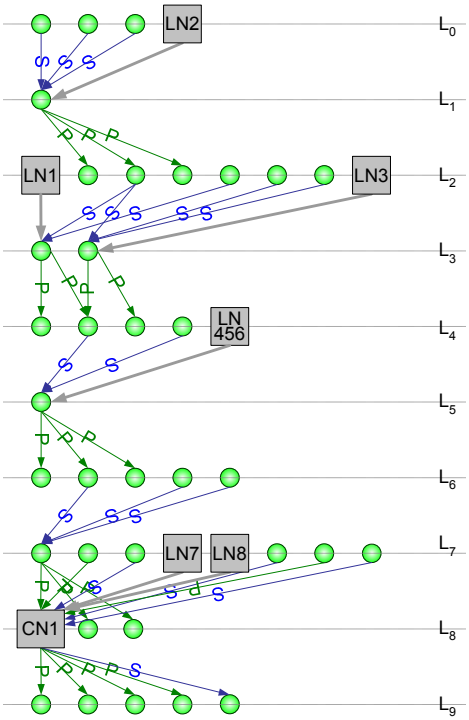



Figure 10.7: Phase 3: Assignment of global layers to the graph.

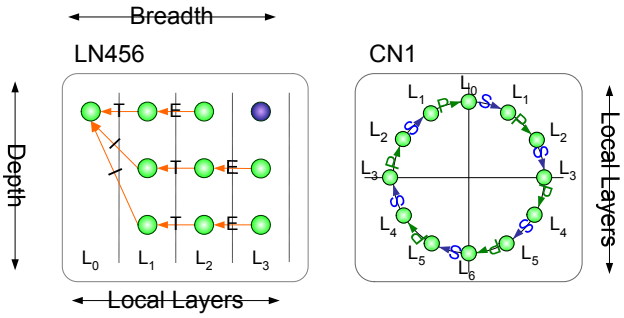



Figure 10.8: Phase 4: Layer assignment for compound nodes (a) l-nodes, (b) c-nodes.

graphical constraint will have a particular compound layer assignation in order to draw them with its specific semantics. This problem is addressed in our future work.

### Phase 5: Edge-Crossing Minimization

The edge-crossing minimization follows the same process in general layers as in compound layers. The edge-crossing minimization is based on the barycentre [24] and median [5] heuristics. Thus we use a 2-layer edge-crossing minimization.

### Phase 6: Expansion of Compound Nodes

In this phase the compound nodes are expanded into the general graph.

L-nodes are moved down one level since recall from Step 2 that compound nodes do not include the last arc, and that such nodes must be aligned horizontally. Then, l-nodes are moved to the right of their level in order to compartmentalize the information they convey with respect to unconstrained nodes. For example, in biochemical networks l-nodes correspond to regulation pathways, which all must be represented at one side of the main pathway. Finally, for æstethic reasons, it is ensured that at most one l-node is located in each layer by moving down the other l-nodes.

Then, the depth of each compound node is taken into account and the nodes in next level are moved down as many levels as needed to leave enough place to compound nodes. As shown in Figure 10.9 (a), this is done by including fake nodes (represented in dark blue). Finally, the compound nodes are expanded as shown in Figures 10.9 (b) and 10.10 (b).

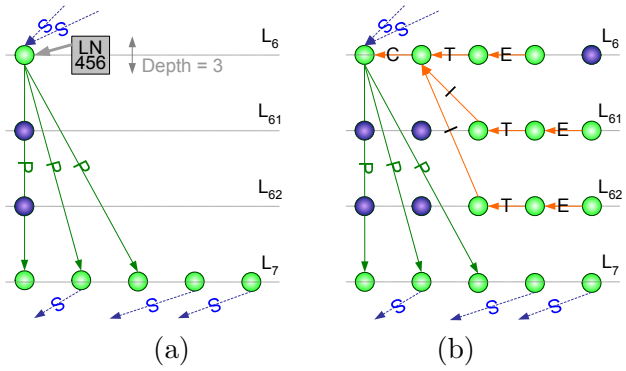



Figure 10.9: Expansion of l-nodes located in a layer $L_i$. (a) The nodes of the layer $L_{i+1}$ are moved down as much layers as the depth of the l-node, by including fake nodes (in dark blue). (b) The node is expanded.

Figure 10.10 shows an example of expansion of c-nodes. Notice that the expansion is followed by a rearrangement of nodes in layers similar to that applied in Step 3. Notice also that the layers generated by the c-nodes are different from global layers, in particular since their height varies to ensure a circular rendering.

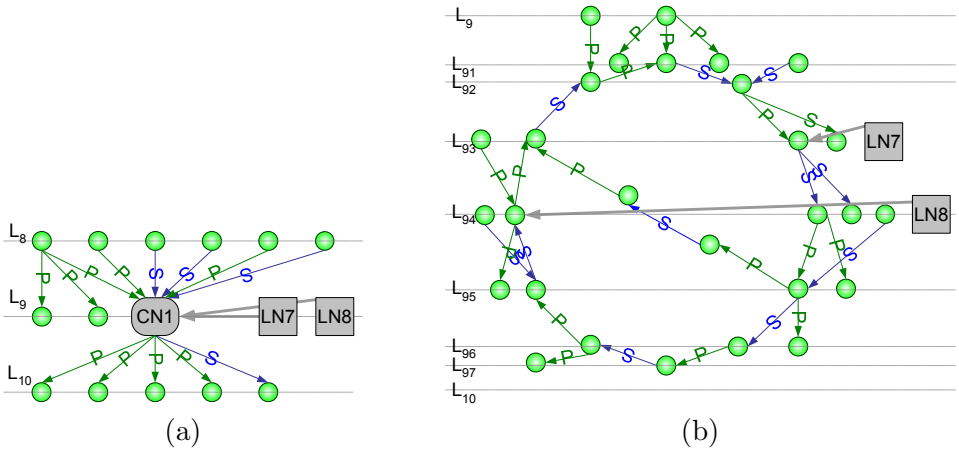
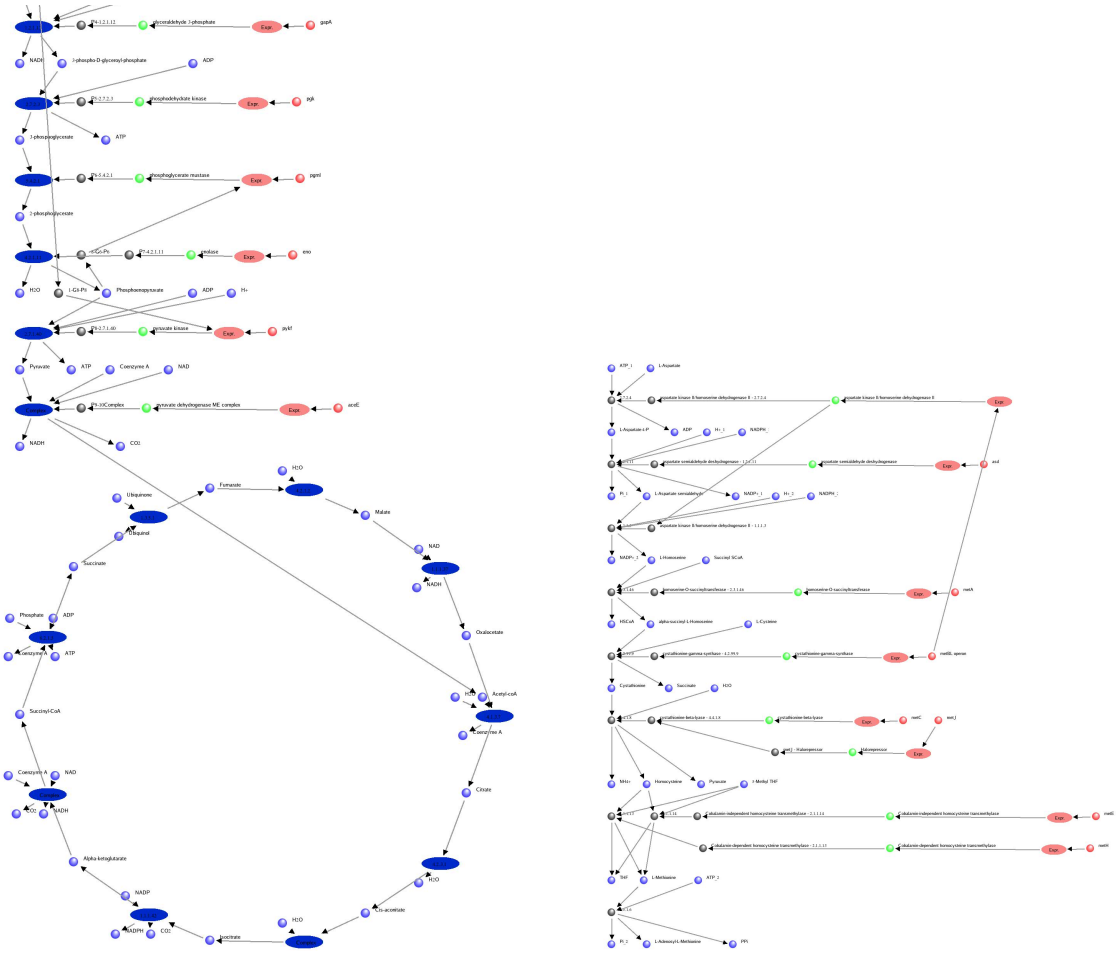



Figure 10.10: (a) Initial situation. (b) Expansion of c-nodes.

**Phase 7: Coordinate Assignment**

In this phase the algorithm transforms the topological layout obtained after Phase 6 into a geometric layout by assigning two-dimensional coordinates to each node. All nodes of a layer $L_i$ will have the same $y_i$ coordinate. Therefore, it suffices to assign an $x$-coordinate to these nodes according to the topological order of $L_i$. We also assign a geometrical position to fake nodes, but we remove them after this phase.

### 10.4.3 Results



(a) (b)

Figure 10.11: The result of the CSCGL algorithm on (a) the glycolysis, a cyclic-regulated pathway, and (b) the methionine, a linear pathway.

Figure 10.11 shows the results of applying the CSCGL algorithm on the glycolysis and the methionine pathways. Three graphical constraints express (1) that genetic regulation has to be horizontal, (2) the allowed arc types in metabolic cycles, which are shown in a circular way, and (3) the allowed type for backward arcs, which is inhibition. If users do not define graphical constraints, the biochemical network will be drawn as in traditional layered algorithms.

In order to show the efficiency of our algorithm, we have randomly generated 25 graphs from 10 to 7000 nodes. We have calculated that in biochemical networks, the average number of arcs is around 120% of the number of nodes. Thus, the random generator provides graphs of $n$ nodes and $n \times 1.2$ arcs. To these randomly-generated graphs we applied the CSCGL algorithm, as well as our implementation of the Sugiyama algorithm [24] and a particular force-directed algorithm, the variant of Simon [18] which is based on spring-embedder, repulsion force, and simulated annealing in order to ensure that the algorithm converges (the maximum iterations allowed were fixed to 80).

The result of this experiment is shown in Figure 10.12. These experiments were carried out on a Pentium P4 HT with 768 M of RAM. We have to take into account the fact that graphs are generated randomly and then cycles too. Thus, some graphs contains more than 500 cycles (as the graph of 2000 nodes), the progression of both, the Sugiyama and the CSCGL are then not strictly positive. Simon's algorithm needs more than 20 minutes for drawing graphs of almost 600 nodes while the CSCGL algorithm follows mainly the same behavior as Sugiyama's for graphs having less than 3000 nodes. For bigger graphs the CSCGL diverges from the Sugiyama, for instance a graph of 7000 nodes needs 24 minutes while the Sugiyama only needs 2 minutes. However, the implementation of CSCGL without the minimization phase needs only 4 minutes for the same graph. A deep optimization of this phase will be addressed in future works.

In conclusion, although our algorithm has higher computing time than that of Sugiyama's algorithm, it is still pretty small for graphs having less than 3000 nodes and even for bigger graphs the computing time remains satisfactory. It is worth noting that the average size of metabolic pathways hardly exceeds 300 nodes.

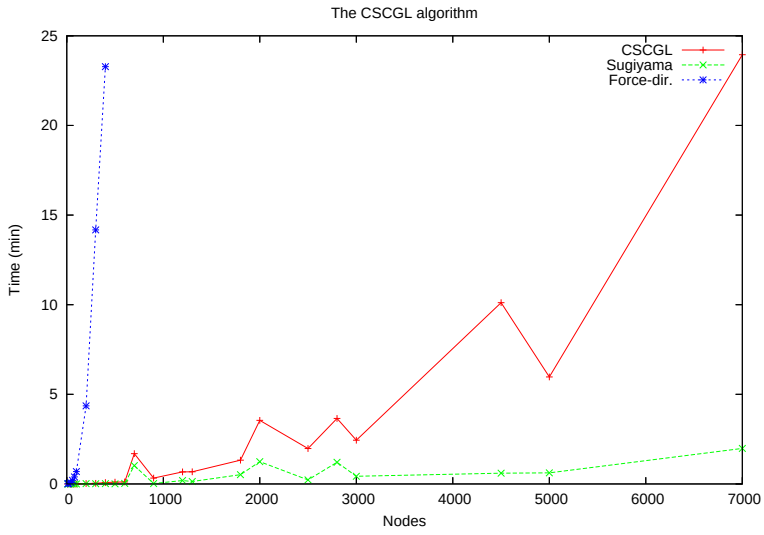



Figure 10.12: Experimental results of our algorithm.

### 10.4.4 Conclusion

In this chapter we have described the CSCGL algorithm, a generic algorithm suited for drawing biochemical networks. The key concept of our algorithm is that it is generic, meaning that it can draw any kind of biochemical network, where each of them is drawn according to

its specific semantics, and this semantics is defined by users as graphical constraints.

Using these constraints our algorithm builds compound nodes containing sets of simple nodes to which a graphical constraint must be applied (Phases 1 and 2). Next, the CSCGL algorithm computes the general position of both simple nodes and compound nodes, where the internal structure of compound nodes is not considered (Phase 3) and then determines the relative position of the internal nodes of the compound nodes (Phase 4). The algorithm then continues by minimizing edge crossings (Phase 5) and then expands the compound nodes in the resulting graph (Phase 6). Finally, the algorithm translates the topological structure of the graph into geometric positions (Phase 7).

Our contribution is to provide a new kind of compound graph layout algorithm than can be adapted to the semantics of any domain related to biochemical networks. The CSCGL algorithm can cope with any kind of biochemical graphs and users can customize the representation of each type of such graphs according to particular representation rules of a domain. Further, for graphs containing less than 3000 nodes the computing time is close to that of traditional Sugiyama-like algorithms and significantly smaller than force-directed layout algorithms. In addition, we developed an efficient cycle enumeration algorithm providing all cycles of a graph, including the inner cycles.

Future works address the problem of enriching the set of graphical constraints in order to allow users to express specific behavior in particular compound nodes. In addition, we have to address the processing of topological cycles. The current version does not adequately process the inner cycles, which are often present in metabolic pathways. Finally, we will optimize Phase 5 devoted to edge-crossing minimization in order to improve the computing time of the algorithm with the aim to arrive to a performance similar to that of Sugiyama's algorithm.

### 10.4.5 Constraint Pattern editor

This editor will be available from version 1.1.X of Visual BioMaze.

The constraint patterns taking into account by the CSCGL algorithm are the graphical constraints provided by the Visual BioMaze Framework. But users can define their own constraint patterns and load them in the Framework. In order to build efficiently such a file, the Visual BioMaze Framework provides the *the Constraint Patterns editor* as shown by Figure 10.13. This editor allows editing the XML file as a tree, to remove or to add pattern or arc-type, but also validate your modified file. Indeed if the file is modified, the editor will automatically validate the new entry as shown by Figure 10.14. The `Valid model` command in the popup menu of Figure 10.15 will validate the current edition file but also shows validator comments.

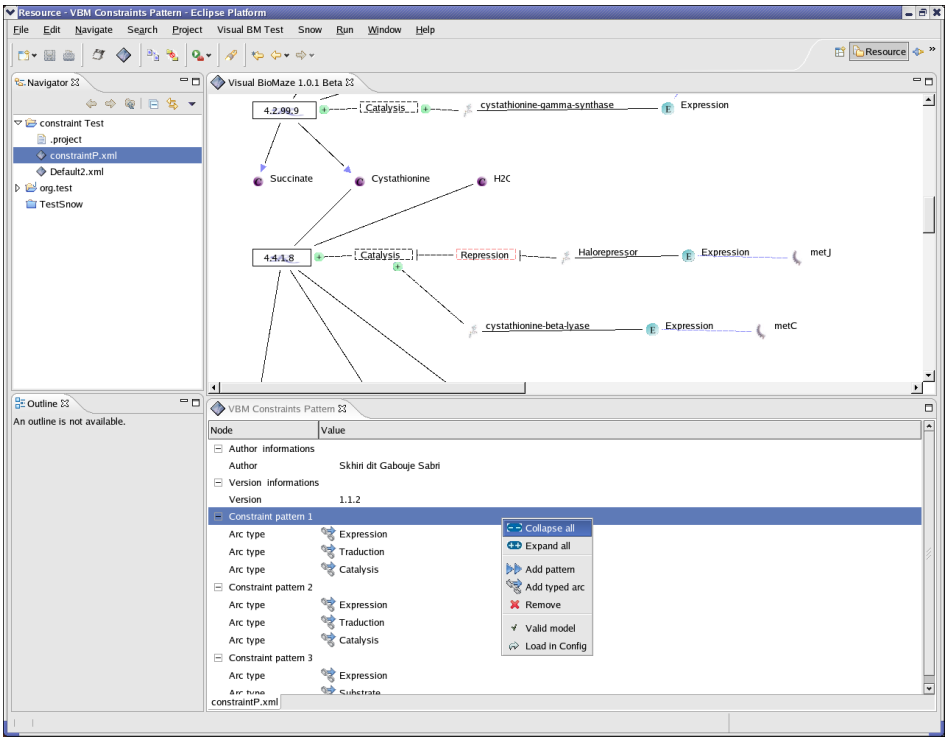



Figure 10.13: The Visual BioMaze contraint patterns editor provides an easy way in order to build valid constraint pattern files.

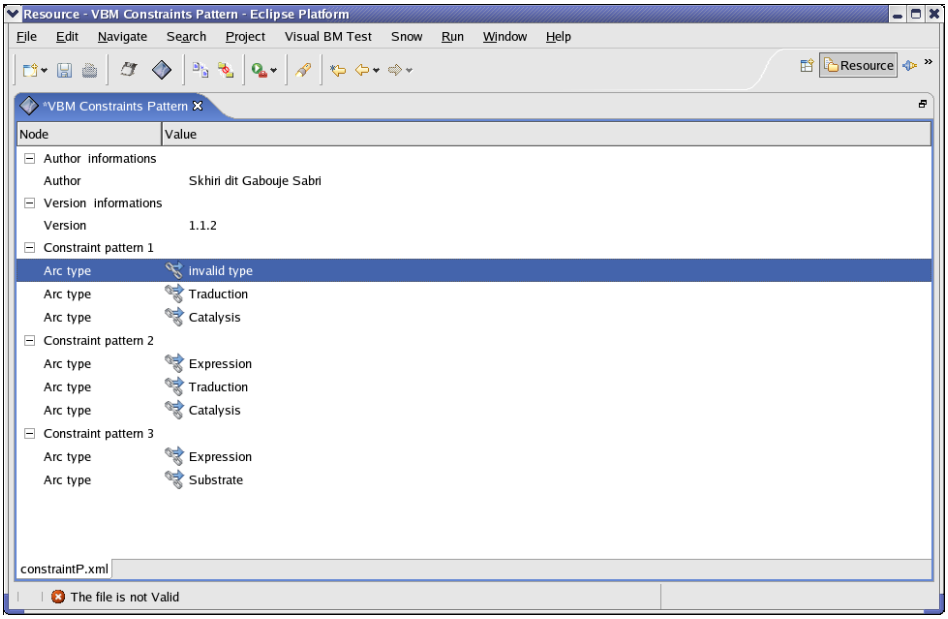



Figure 10.14: The Visual BioMaze constraint patterns editor validator watches any modifications of the file and valid it.

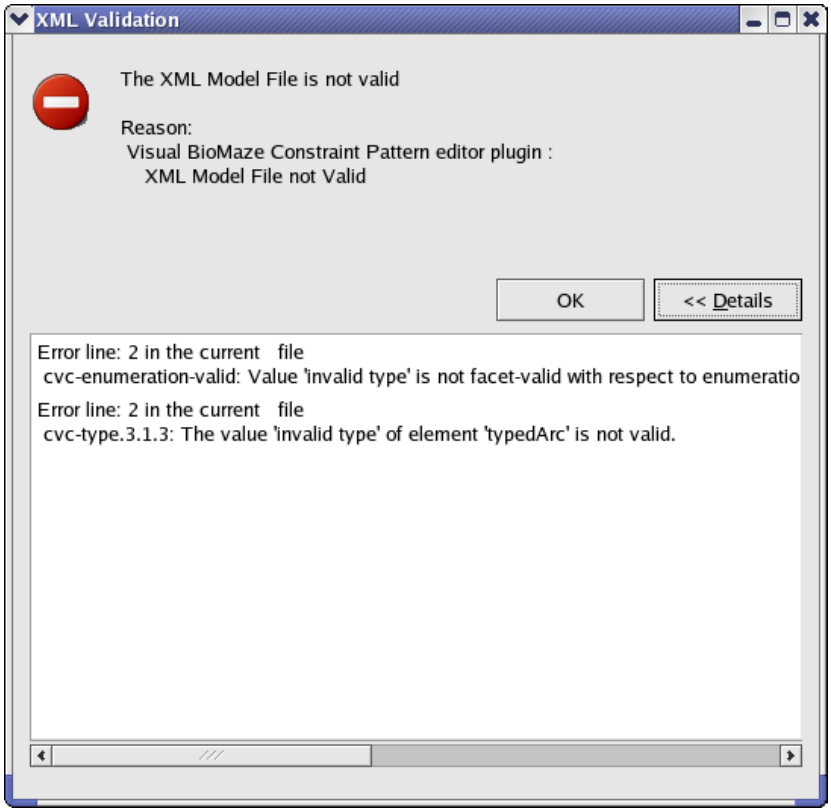



Figure 10.15: The `Valid model` command in the popup menu allows to show the validation errors according to the XML schema definition for the constraint pattern file.

The Load in config command in the editor popup menu will export the current constraint pattern file in the reserved configuration directory. Then users can load this file as constraint patterns reference in the Visual BioMaze preference page (Figure 10.16). Then all graph layout algorithms taking into account the constraint patterns will receive the constraint patterns represented by the loaded file.

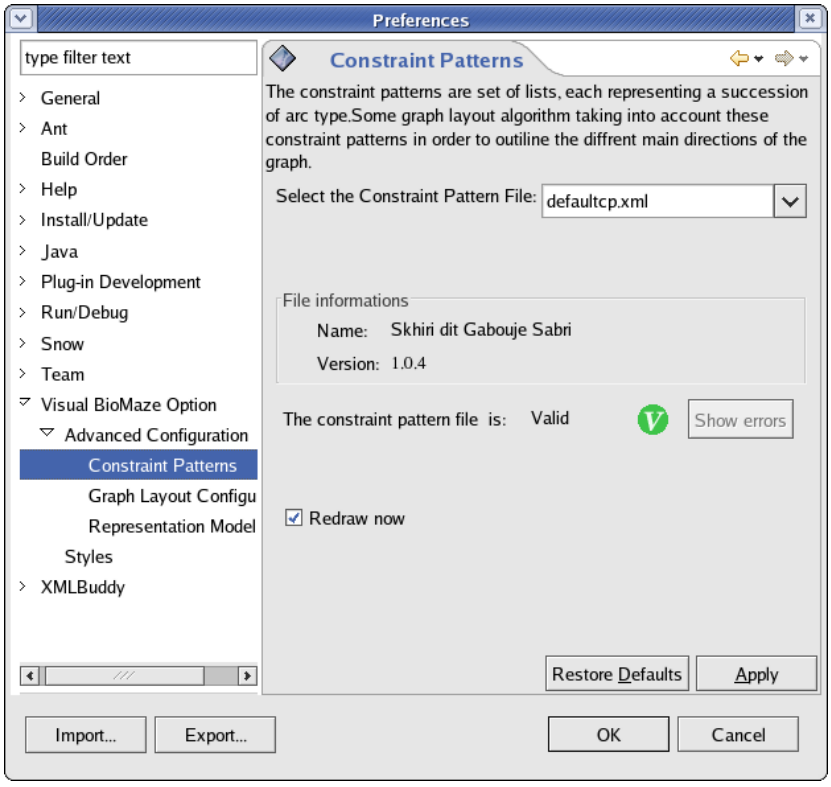



Figure 10.16: Once the constraint patterns file is exported, users can choose it and apply as reference constraint patterns.

### 10.4.6 Future work

The major adaptation comes from the introduction of the keyword *any* in constraint patterns. We will also modify the algorithm to differentiate the main compound from the simple compound, where the principal compound is the set of biochemical entities that form the backbone of the pathway. Finally, we will introduce visual improvements like balance in both general and compound layers. One other main adaptation comes from handling inner cycles. For the moment we detect inner topological cycles but we does not treat them, later we will draw them inside the cycle which contains them.

# Bibliography


[1] The Eclipse project. `http://www.eclipse.org`.

[2] D.L. Cook, J.F. Farley, and S.J. Tapscott. A basis for visual language for describing, archiving and analyzing functional models of complex biochemical systems. *Genome Biology*, 2:research0012.1–OO12.10, 2001.

[3] P. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 2001.

[4] P. Eades, X. Lin, and W.F. Smyth. A fast effective heuristic for feedback arc set problem. *Information Processing*, 13:427–437, 1993.

[5] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

[6] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[7] M.R. Garey and D.S. Johnson. Crossing number is np-complete. *SIAM J. Discrete Math.*, 4:312–316, 1986.

[8] Mao Lin Huang, Peter Eades, and Junhu Wang. Online animated graph drawing using a modified spring algorithm. *Australian Computer Science Comm.: Proc. 21st Australasian Computer Science Conf., ACSC*, 20(1):17–28, 4–6 1998.

[9] M. Junger and P. Mutzel. *Graph Drawing Software*. Springer, 2003.

[10] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31:7–15, 1989.

[11] P.D. Karp, S. Paley, and P. Romero. The Pathway Tools software. *Bioinformatics*, 18(90001):s225–s232, 2002.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[13] K.W. Kohn. Molecular interaction map of the mamalian cell cycle control and DNA systems repair. *Molecular Biology of the Cell*, 10:2703–2374, 1999.

[14] C. Lemer, E. Antezana, F. Couche, F. Fays, Santolaria X., R. Janky, Y. Deville, J. Richelle, and S. Wodak. The aMaze lightBench: a web interface to a relational database of cellular processes. *Nucleic Acids Research*, 32:443–448, 2003.

[15] R. Maimon and S. Browning. Diagrammatic notation and computational structure of gene networks. In *Proc. of the 2nd Int. Conf. on Systems Biology*, pages 311–317, 2001.

[16] N.R. Quinn and M.A. Breuer. A force directed component placement procedure for printed circuit boards. In *IEEE Transaction on circuits and system*, volume 26, pages 377–388, 1979.

[17] S. Shavor, J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, and P. McCarthy. *The Java Developer's Guide to Eclipse*. Addison-Wesley, 2003.

[18] F. Simon, F. Steinbruckner, and C. Lewerentz. 3d-spring embedder for complete graphs. Technical report, Technical Unieversity Cottobus, 2000.

[19] E. Sopena. *Element de théorie des graphes*, 2004.

[20] K. Sugiyama. *Graph drawing and applications: For sofware and knowledge engineers*. World Scientific, 2002.

[21] K. Sugiyama and P. Eades. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.

[22] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–893, 1991.

[23] K. Sugiyama and K. Misue. A simple and unified method for drawing graphs in magnetic-spring algorithm in lecture notes in computer science. In *Graph Drawing 1994*, volume 894. Springer-Verlag, 1995.

[24] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:109–125, 1981.

[25] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1:146–149, 1982.

[26] J. van Helden, A. Naim, C. Lemer, R Mancuso, M. Eldridge, and S. Wodak. From molecular activities and processes to biological function. *Briefings in Bioinformatics*, 2(1):81–93, 2001.

[27] J. van Helden, A. Naim, C. Lemer, R. Mancuso, M. Eldridge, S. Wodak, and D. Gilbert. Representing and analyzing molecular and cellular function in the computer. *Biological Chemistry*, 381(9–10):921–935, 2001.

[28] E. Zimanyi and S. Skhiri dit Gabouje. Semantic visualization of biochemical databases. In *Semantic for GRID Databases: proc. of the int. conf. on Semantics for a networked world ICSNW04*, 2004.