A New Constraint-Based Compound Graph Layout Algorithm for Drawing Biochemical Networks

Sabri Skhiri dit Gabouje
Alcatel Bell, Mobile Solution Division (MSD), Belgium
Email: sabri.skhiri@alcatel.be
Esteban Zimányi
Department of Computer and Decision Engineering (CoDE)
Université Libre de Bruxelles, Belgium
Email: ezimanyi@ulb.ac.be

Abstract
Due to the huge amount of information available in biochemical databases, biologists need sophisticated tools to accurately extract the information from such databases and to interpret it correctly. Those tools must be able to dynamically generate any kind of biochemical subgraph (i.e., metabolic pathways, genetic regulation, signal transduction, etc.) in a single graph. The visualization tools must be able to cope with such graphs and to take into account the particular semantics of all kinds of biochemical subgraphs. Therefore, such tools need generic graph layout algorithms that adapt their behavior to the data semantics. In this paper we present the Constrained Compound Graph Layout (C^2GL) algorithm designed for the generic representation of biochemical graphs and in which users can represent knowledge about how to draw graphs in accordance with the biochemical semantics. We show how we implemented the C^2GL algorithm in the Visual BioMaze framework, the visualization tool of the BioMaze project[1].

Keywords: Metabolic pathways, graph layout, visualization of biochemical networks, compound graphs.

1 Introduction

The advent of a new generation of biochemical databases, e.g., aMAZE (Lemer, 2004), asks for more efficient and flexible visualization tools. Biochemical databases typically contain information about biochemical entities such as compounds, genes, and polypeptides, as well as the interactions between them. There are two categories of interactions: (1) *transformations*, such as reactions (between compounds), expressions (of a gene that lead to synthesis of polypeptides), assembly-disassembly (between biochemical entities) and (2) *control*, like catalysis (a polypeptide catalyzes or inhibits one or more biochemical reactions), or activation-deactivation (turning on or off the biochemical function of a polypeptide). The term *biochemical pathway* or *biochemical network* regroups different families of networks. *Metabolic pathways* are networks of biochemical reactions catalyzed by polypeptides resulting from the expression of a gene. This expression is controlled by a set of parameters like transcription factors, activation, inhibition, etc. For this reason metabolic pathways are said to be genetically regulated. These regulatory actions are represented in *metabolic regulation networks*. Finally, *signal transduction networks* describe the

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

information transfer from a cellular location, typically the extra-cellular medium, to another one, typically the cell nucleus.

Biochemical pathways are modeled in the aMAZE database as complex networks of interactions. A powerful extraction tool allows users to extract and generate on-demand a metabolic pathway as well as its genetic regulation, transduction signal information, or any other biological information. Such networks are called *heterogeneous biochemical graphs*, where nodes are biological elements such as compound, gene, polypeptide, and where interactions are arcs such as substrate, product, catalyze, inhibition, expression, etc. An example of such graphs arrives when a liver cell receives the signal that the sugar rate is increasing (signal transduction) and its response in which the insulin acts (metabolic pathway).

For many years each subpart of such a graph has been drawn separately by hand in biochemistry books and have thus graphical conventions that graph layout algorithms must respect. Then, visualization tools coping with such graphs must be able to draw each biochemical subpart according to its particular semantics and representation rules in the same graph. This problem can be generalized as "how to draw a graph according to the particular semantics of its subgraphs?"

2. Background

Graphs provide a way to structure the information presented to users by showing the relevant objects and the relations between them. Such a representation is used in a significant number of areas: software design, business processes, semantic web, electrical design, bioinformatics, etc. Graph drawing has emerged in recent years as an important area in computer science. The aim of graph drawing algorithms is to compute the position of each node of a graph while optimizing several æsthetics and efficiency criteria, such as edge-crossing minimization, bend minimization, area minimization, symmetries, angle maximization, distance between nodes, etc. (Sugiyama & Eades, 1990; Sugiyama & Misue, 1995).

Usually the output of graph drawing algorithms can be improved manually. The reason is that most of graph drawing algorithms do not take into account domain semantics known by the persons working with the graph. For instance, in a graph representing a typical class diagram one could prefer to see the generalization or "is-a" relationship vertically forming tree-like hierarchies while the aggregation or "part-of" relationship shown horizontally. This constraint is impossible to express in layered graph layout algorithms. On the other hand, force-directed graph layout algorithms try to optimize a mathematical function, named *energy*, which satisfies æsthetic constraints such as in Simon *et al.* 2000, Eades 1984 or Davidson & Harel 1996. This means that the semantics of the graph to draw must be expressed as a mathematical function and must be optimized against æsthetic criteria. One drawback of such algorithms is that each application domain should define its own mathematical criteria. In addition, the computation time may become important. In Kaufmann & Wagner 2001, the authors note that these æsthetics and efficiency criteria contrast with more intuitive criteria concerning the semantics and the intended meaning of graphs. Then, we should be able to separate the semantics of information conveyed by the graph and the graph layout algorithm. The questions we try to answer are then

"how can we express the drawing knowledge of a particular application domain and how can the graph layout algorithm consider it?"

Drawing a biochemical pathway is such a typical graph drawing problem (Van Helden *et al.*, 2001). The traditional æsthetics and efficiency criteria are obviously important but do not take into account the semantics of biological pathways defined by biologists for many years. In addition, as already said, such graph layout algorithms must be able to cope with heterogeneous graphs and thus must take into account the semantics of each kind of subgraph involved. Most of the bioinformatics tools aiming to represent biochemical pathways on-demand, such as BioPath (Schreiber, 2002), PathDB (Mendes *et al.*, 2000), ISYS (Siepel *et al.*, 2001), and BioCYc (Karp & Paley, 1994) are not able to draw heterogeneous graphs having in the same drawing a biochemical subpart, a transduction subpart, and a regulation subpart. The reason is that either they use force-directed algorithms or because the graph layout algorithm used has been customized for a specific type of subgraph, and thus cannot represent a heterogeneous graph according to the different semantics. As graphs to lay out are generated on demand as answers to queries addressed to the database, the processing time becomes a significant constraint. Thus, we must provide an efficient method that limits the computation time as well.

The CSCGL algorithm (Skhiri dit Gabouje & Zimányi, 2005) provides a first attempt to solve this problem. However, the algorithm suffers from the following drawbacks: (1) it can only place a subgraph horizontally and thus, it is unable to draw products and substrates in one side of biological reactions, (2) the layout of the genetic regulation of cycles is not in accordance with usual drawing conventions, (3) the available graphical constraints are too limited, and (4) the computing time is too high for graphs having more than 2000 nodes.

In this paper we present a generic algorithm that answers more efficiency this problem and in which users can introduce the knowledge about how to draw a graph more accurately and in accordance with the particular semantics of biochemical pathways. This knowledge is introduced in the algorithm via graphical constraints. We have implemented our algorithm in the Visual BioMaze framework (Zimányi, 2004; Skhiri dit Gabouje, 2005), a generic viewer of graphs developed in the BioMaze project (Dooms *et al.*, 2004a; Dooms *et al.*, 2004b; Zampeli *et al.*, 2004).

2 The BioMaze Project

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 available and its continuous growth, the need to integrate such information, as well as the necessity of sophisticated software tools for manipulating it represent true challenges for 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, metadata, 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.

BioMaze is implemented in the Eclipse platform. It uses the database of the aMAZE project (Lemer *et al.*, 2004).

Visual BioMaze (VBM) (Zimányi & Skhiri dit Gabouje 2004), is the visualization framework of the BioMaze project. Although it was developed in the context of BioMaze, the visualization framework can be used independently. 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 graph, with any kind of graphical representation and with any kind of graph layout algorithm.

When a graph has to be shown, the visualization framework uses 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 a graph, i.e., the graphics associated to each node type and each arc type. On the other hand, a *graph layout algorithm* computes the position of each node of the graph. A generic graph viewer must answer to two fundamental requirements: it must be able to associate to each node type a specific graphical representation and it must be able to dynamically load any graph layout algorithm. The Visual BioMaze framework fulfills these two requirements thanks to a modular architecture based on plugins.

As a result, the tool can cope with any kind of graph independently of its semantics and users can choose both a suited representation model and a suited graph layout algorithm. Concerning the BioMaze project, it means that VBM can cope with any kind of biochemical graph 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 that retrieves such a graph, the VBM framework is able to represent these two kinds of biochemical networks in the same drawing. Such a result is not yet available in other biochemical visualization software.

Therefore, a generic visualization framework requires algorithms that adapt themselves to the specific semantics of a domain, and that can draw graphs accordingly. The C^2GL algorithm is one of such generic algorithms.

3 The Algorithm

3.1 Graphs

The Constrained Compound Graph Layout (C^2GL) algorithm takes as input general typed directed graphs $G(V, E, T_V, T_E)$, where V is the set of typed nodes, E is the set of typed arcs, and T_V (respectively, T_E) is the set of node (respectively, arc) types. However, the algorithm is suited for graphs outlining a particular information flow as in the case of metabolic pathways. Those pathways are a collection of interconnected biological reactions in which the main substrates and products constitute the backbone of the pathway. Figures 1 and 2 show, respectively, a graph and its backbone.

Our algorithm starts by extracting the backbone constituting the main direction of the graph. Then, it builds the associated compound digraph based on the definition of graphical constraints. These constraints, expressed in XML files, describe how the algorithm must place a set of nodes. Eventually, the resulting compound graph is processed in order to compute the node positions.

Compound graphs were introduced in Sugiyama & Misue (1991). Such a graph $C = (G,T)$ is defined as a directed graph $G = (V, E_G)$ and a rooted tree $T = (V,E_T)$ that share the same vertex V. In this case we have $G = (V,E_G,E_T,T_V,T_E)$. An example is given in Figure 8.

Definition (Kaufmann & Wagner 2001) An inclusion digraph is a pair $T = (V, E_T)$ where E_T is finite set of inclusion edges such that $(u, v) \in E_T$ means that u includes v. T is required to be a rooted tree. An adjacency digraph is a pair $G = (V, E_G)$ where F is a finite set of adjacency edges such that $(u, v) \in E_G$ means that u is adjacent to v. A compound digraph is defined as a triple $D = (V, E_T, E_G)$ obtained by composing these two graphs.

Definition The depth of a node $v \in V$ is the number of nodes on the path between v and the root of T and is denoted by depth(v), with depth(root) = 1.

Definition Given a graph $G = (V, E)$ and a vertex $v \in V$, we denote the set of incoming arcs of v by instar(v) = $\{ (u, v) \in E \mid u \in V \}$, and the set of outgoing arcs from v by outstar(v) = $\{ (v, u) \mid u \in V \}$.

3.2 Container types

The C^2GL algorithm uses three different container types.

- *8-direction container*, which defines a central reference node and 8 other nodes placed around the reference node (Figure 3).
- *Unidirectional container*, which aligns all its inner nodes in one direction which is either horizontal or vertical (Figure 4).
- *Circular container*, which aligns all its inner nodes along a circle (Figure 5).

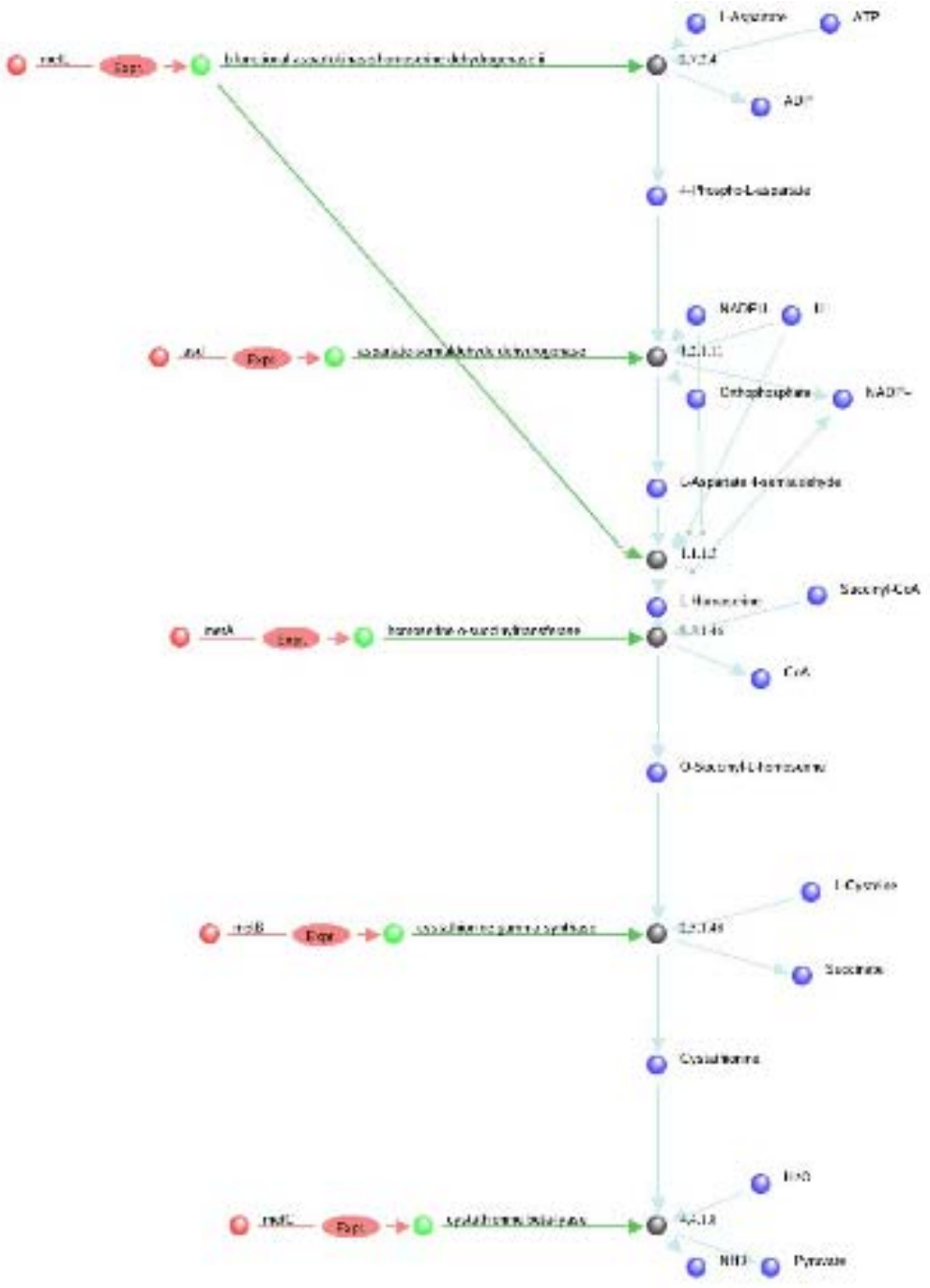

Figure 1. A part of the methionine biosynthesis pathway and its genetic regulation. The main direction (vertical) shows the sequence of biochemical reactions of the pathway. Each of them is regulated by a polypeptide resulting from the expression of a gene. This graph was automatically generated by the C^2GL algorithm.

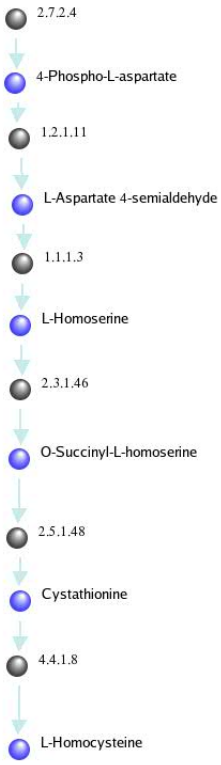

Figure 2. The backbone of the graph of Figure 1. The backbone is a subgraph representing the main direction of the information flow.

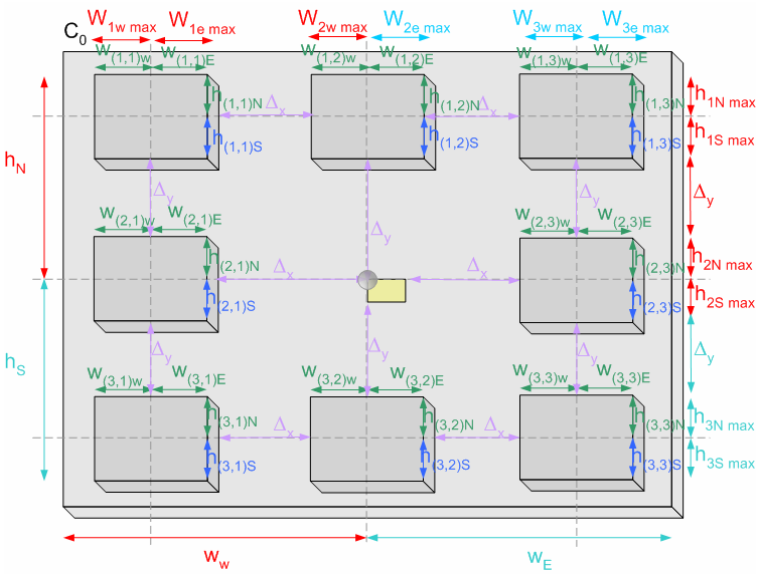

Figure 3. An 8-direction container defines a reference node and 8 other nodes around it. The other nodes can be simple nodes or other containers.

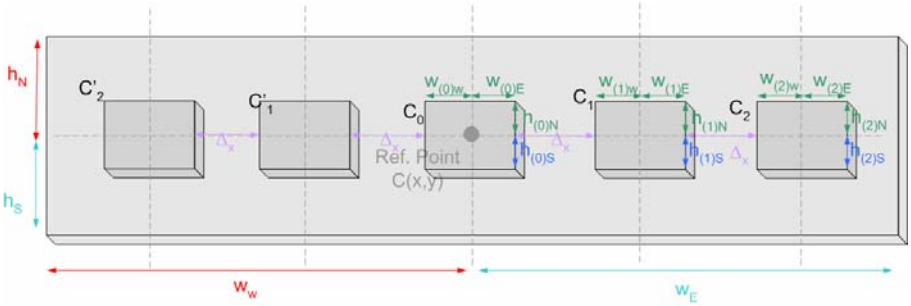

Figure 4. An unidirectional container.

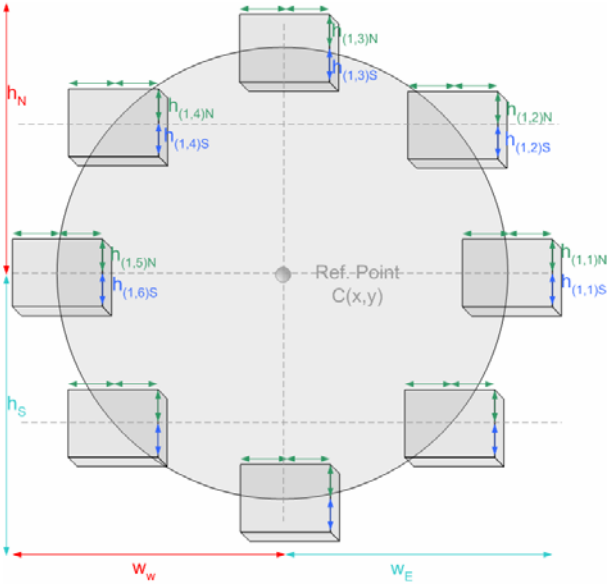

Figure 5. A circular container.

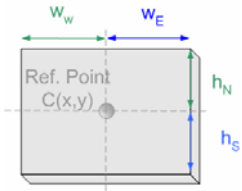

Figure 6. All containers, independently of their type, are seen as nodes having a certain width and height.

These containers provide sufficient expressive power for giving any particular layout to the nodes connected to the main backbone. Each container is a compound node, i.e., a graph C = (*G*, *T*) in which inner nodes, that can be either simple nodes or other containers, compose the set of vertex *V* shared by *G* and *T*.

From the outside, each container is seen as a node having a width and height (Figure 6). Although the size of each container depends on its type, each of them has four parameters: h_N (the North height), h_S (the South height), w_E (the East width), and w_W (the West width). These parameters are used in the coordinate assignment phase.

3.3 Graphical constraints

We use graphical constraints to express the information about how to draw any information flow diagram according to specific domain semantics. These graphical

constraints are expressed as a set of *local constraints*, *circular constraints*, and *backbone constraints* which are encoded in XML. These constraints are described next.

3.3.1 Local constraints

A local constraint is defined by a set of tables as those shown by Tables 1 and 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). 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 1) we have to define one such a table.

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

Table 1. Local constraints around the reference type node.

S	**N**	**E**	**W**	**NE**	**NW**	**SE**	**SW**
Ver.	Horiz.	Ver.	Ver.	Ver.	Ver.	Horiz.	Ver.

Table 2. Alignment policies.

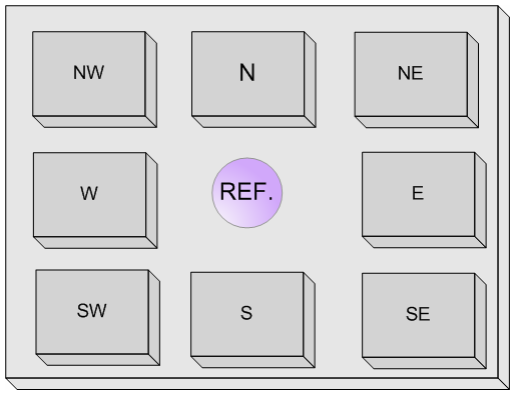

Figure 7. Local constraints define the disposition of nodes types around a node type reference.

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 4.4. The local constraints define also the kind of behavior to adopt if more than one node is matched in one constraint container. For instance, Table 2 states that if two nodes must be placed in NW of the reference node, those will be aligned vertically.

3.3.2 Circular constraints

Circular constraints define whether a cycle must be drawn along a circle or must be drawn by inversing the minimal set of backward edges (Eades *et al.*, 1993). 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.

3.3.3 Backbone constraints

These constraints define the set of nodes and arcs that constitute the main direction flow of the graph (Table 3).

From	To	Arc type
Compound	BiologicalReaction	Substrate
BiologicalReaction	Compound	Product

Table 3. The set of nodes and arcs constituting the backbone of metabolic pathways.

4 Algorithm description

The C^2GL algorithm is composed of five phases:

- Phase 1: Cycle management
- Phase 2: Backbone extraction
- Phase 3: Root container building
- Phase 4: Container building
- Phase 5: Coordinate assignment

We describe next these phases in more detail.

4.1 Cycle management

This phase isolates and enumerates all cycles. We first find the strongly-connected components (Tarjan, 1972) of the graph. Then, we apply a modified version of the depth-first search that uses backtracking in order to enumerate cycles. If more than one cycle is found in a strongly-connected component, the outer cycle is called the *main cycle* and the others are called *inner cycles*. Each component is replaced by a *cycle node*, which is a special node representing a cycle and its inner cycles. Algorithm 1 returns all cycles of a strongly-connected component.

```
procedure cycleSearch
        /* start is an arbitrary node of the component, cycles is the list of all cycles of
the     component initialized as empty, and currentCycles is a temporary list.*/
        Node current = start;
        Bool stop = false;
        while stop ≠ true
                if current is not marked
                        mark(current)
                        currentCycle.add(current)
                        if current.outStar >1
                                for i ← |current.outStar|
                                        next_i = current.outStar[i]
                                        int index = size(currentCycle)
                                        List cycle_i = copy(currentCycle)
```

```
                        cycleSearch(next_i, cycles, cycle_i)
                        demark(cycle_i, index)
                    end for
                else
                    current = current.outStar[0]
                end if
            else
                stop = true
                if current = currentCycle[0]
                    cycles.add(currentCycle)
                end if
            end if
        end while
end procedure
```

Algorithm 1. Cycle search.

This algorithm marks recursively the nodes of the strongly-connected components, if one marked node is found, we add the cycle to the list of cycles (variable cycles in the pseudo-code). If a node has more than one outgoing arc, we store the size of the current cycle, we copy its list of nodes and we launch recursively the algorithm. When the algorithm returns, all cycles that can be reached from this node have already been discovered. Finally, we demark the nodes of the current cycle from index, because they can be involved in other cycles.

4.2 Backbone extraction

This phase extracts all nodes composing the backbone of the graph. These nodes represent the main direction from which the graph has to be drawn. Given a set of backbone constraints *C*, this phase is composed of four steps:

- Step 1: Removing all nodes *u* such that $u \notin C$
- Step 2: Removing all nodes *u* such that instar(*u*) = 0
- Step 3: Removing all nodes *u* such that outstar(*u*) = 0
- Step 4: Backbone layers assignment

Step 1 consists in removing nodes and arcs that are not involved in the backbone. Steps 2 and 3 process the backbone, and finally Step 4 assigns a layer to each node. If Step 4 finds a cycle node, it will investigate its content (i.e., the list of nodes composing the main cycle and the lists composing the inner cycles) in order to know if the cycle must be included in the backbone. In Step 4 layers cannot be assigned as in a traditional layered algorithm, since in this case the branches of a pathway will not appear correctly. Figure 8 (a) shows several branch nodes, i.e., nodes having more than one successor. Our approach is as follows. For each successor node *u* of a branch node: (1) we launch the algorithm recursively with a new list of layers (newLayers) and with *u* as root, and (2) we build a container layer with the newLayers as content. Figure 8 (b) shows such a result, the container layers are recursively built from a branch. If a node already belongs to another layer container, a bend point is taken instead of the node itself.

As a result we obtain a set of layers constituting the backbone, where layers contain either simple nodes or other container layers.

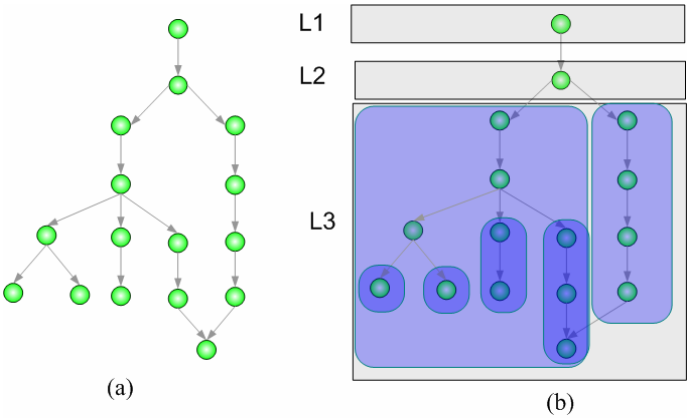

Figure 8. (a) Example of layer assignment in a branch case. (b) The algorithm builds recursively the container layers that will be transformed later in unidirectional containers.

4.3 Root container building

This phase builds an unique container from the backbone extracted in the previous phase. The procedure is simple. The backbone is composed by layers. Each of them will be transformed into an unidirectional container as shown by Figure 9, and all these containers will be included in a root container. If a layer contains a container layer instead of a simple node, we build recursively a unidirectional container corresponding to this container layer. As a result we obtain a root container which is the root of $T = (V, E_T)$, the inclusion graph of the general compound graph. After this phase we must delete all arcs involved in the backbone, since they cannot participate into the next phase.

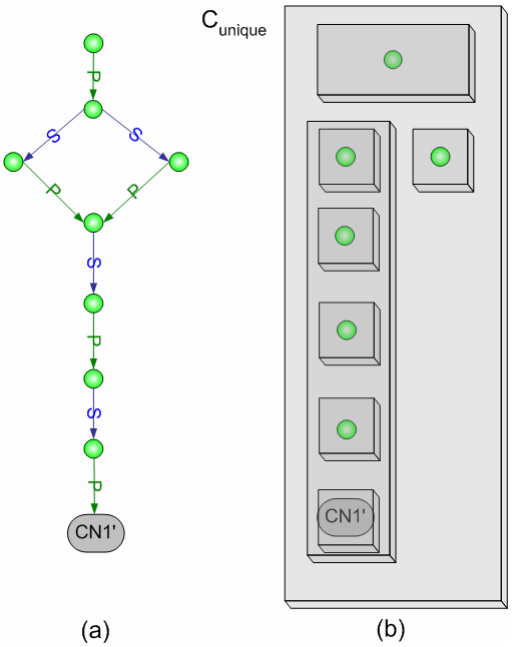

Figure 9. (a) The extracted backbone. (b) The unique container that is the root of the new created compound graph built from the backbone.

4.4 Container building

This phase parses the backbone and for each node builds the associated compound node from local constraints. The input of this phase is the root container and the set of local constraints, and as output we obtain the root container enriched by new containers in which inner nodes are either simple or compound nodes. We denote by C_L the set of local constraints, where each $c_i \in C_L$ has all attributes defined by Table 1. Also C_B denotes the set of backbone nodes.

This phase parses recursively the *constrained nodes* of the root container, i.e., the set of nodes $\{ u \mid u \in V, u \in C_B \}$ such that there is a constraint $c_i \in C_L$ involving u.type as reference. Given a reference node u for a constraint c_i:

1. For each position (E, W, NW, etc.) of the reference node we build a new container container$_i$ and we add the constrained nodes found for this position (Figure 10).
2. We replace the reference node u by the container$_i$.
3. We apply recursively the same procedure to the constrained nodes.

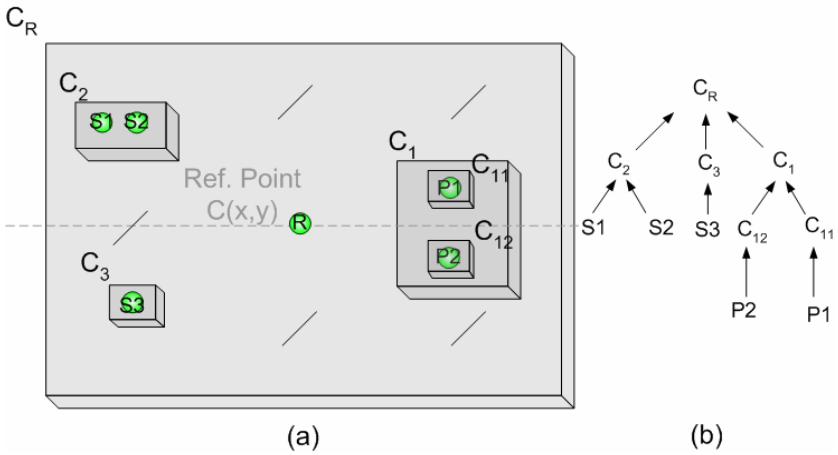

Figure 10. (a) Building of constrained containers in a container C_R. For the reference node R there are 2 nodes in NW, one node in SW, and 2 nodes in E. Further, in position E, there is a local constraint on the constrained nodes. We build 2 new containers, we add the nodes inside and we replace the nodes by the container parent. We continue by applying recursively this phase. (b) The corresponding inclusion graph T before investigating nodes P_1 and P_2.

4.5 Coordinate assignment

This phase computes the position of each inner node in all containers. First, the size of the root container will be computed, this includes the recursive computation of the size of all inner containers. Then, the position of each base node and container can be computed. We assign an initial position (0,0) to the reference position of the root container, and this position will be propagated to inner nodes and inner containers.

4.5.1 Unidirectional container

Consider a unidirectional container with n inner nodes aligned vertically. The position of each node n_i in the North part of the container is as follows, where $i = 1, \ldots, n/2$:

$$x_i = x_{ref}$$
$$y_i = y_{ref} - \sum_{k=b}^{i-1} n_k.h_N + n_{k+1}.h_S + \delta y$$

where $n_i.h_N$ (respectively, $n_i.h_S$) is the North (respectively, South) height of node n_i If n mod 2 = 0, b = 1, otherwise b = 0. For the South part of the container, the – sign in the second equation above is replaced by a + sign.

4.5.2 8-direction container

The position of each node is computed from the size of its neighbors. 8-direction containers can be seen as a 3 × 3 matrix, each node being identified by $n_{j,k}$ where j is the row and k the column. For example, $n_{1,2}$ is the node in the North position (N). We define next the position of the last line of the container, the other lines are computed similarly. For node $n_{3,1}$:

$$x_{3,1} = x_{ref} - (n_{3,2}.w_W + \delta x + n_{3,1}.w_E)$$
$$y_{3,1} = y_{ref} + n_{2,1}.h_S + \delta y + n_{3,1}.h_N$$

where δx (respectively, δy) is the minimal distance between nodes along the y (respectively x) direction. For the node $n_{3,2}$:

$$x_{3,2} = x_{ref}$$
$$y_{3,2} = y_{ref} + n_{2,2}.h_S + \delta y + n_{3,2}.h_N$$

And finally for the node $n_{3,3}$:

$$x_{3,3} = x_{ref} + n_{3,2}.w_E + \delta x + n_{3,3}.w_W$$
$$y_{3,3} = y_{ref} + n_{2,3}.h_S + \delta y + n_{3,3}.h_N$$

4.5.3 Circular container

The size of the circle is computed from the size of the inner containers. Since the radius R is already computed, the position of each contained node is as follows:

$$x_i = x_{ref} + R.\cos(2i\pi/n)$$
$$y_i = y_{ref} + R.\sin(2i\pi/n)$$

for i = 1, … , n, where n is the number of inner nodes. Concerning the inner cycles, we first isolate the *anchor nodes*, i.e., the first and last nodes shared between the inner cycle and the main cycle. We compute the equation of the line passing by those points, and then, we compute the position of each node of the inner cycle on this line.

4.6 Results

The main characteristics of the C^2GL algorithm are as follows:

1. it emphasizes the hierarchical flow of biochemical networks due to the backbone extraction (Figure 11);
2. it can cope with any kind of biochemical network;

3. users can customize the representation of subgraph types according to the graphical representation rules; and
4. metabolic pathways are visualized as is usually done in biochemistry books.

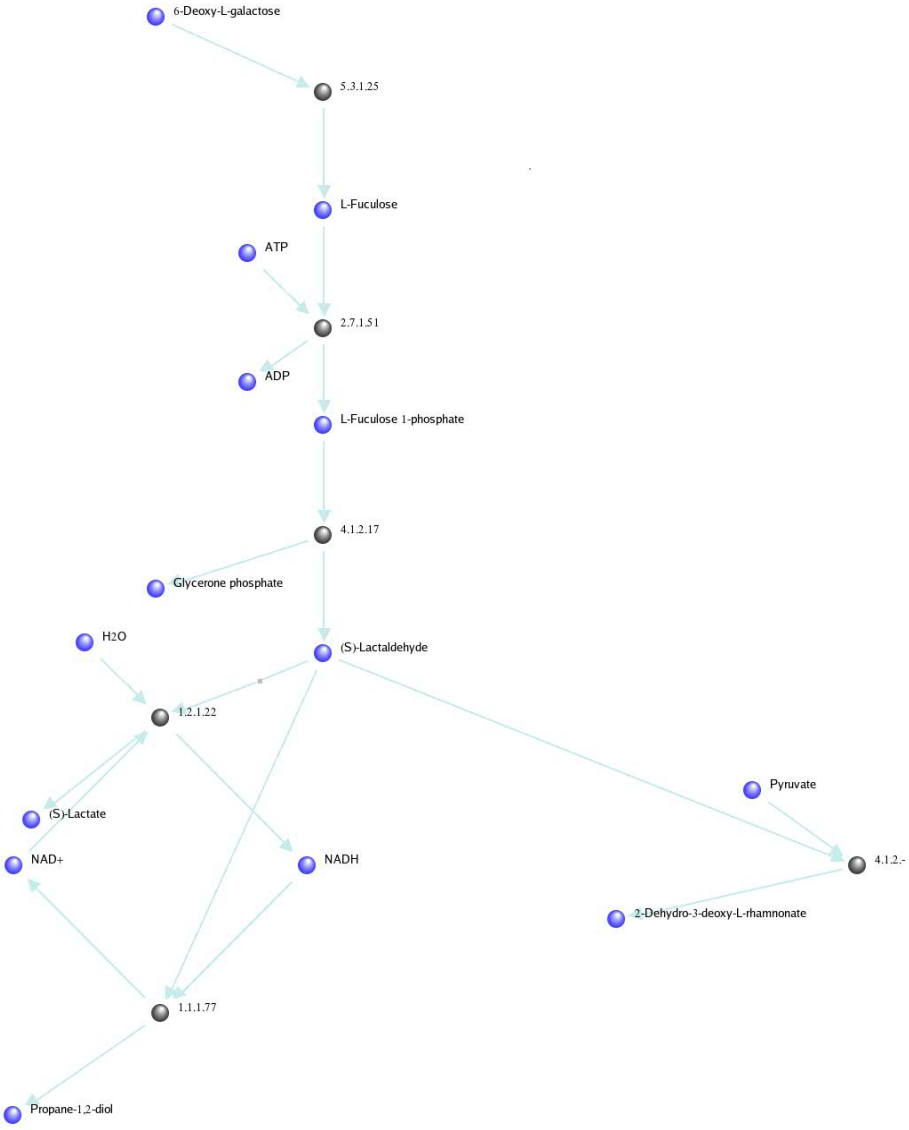

Figure 11. The fuculose catabolism pathway. The drawing shows the backbone of the pathway given by our extraction algorithm. Notice that the branches are correctly drawn, in particular the left branch which is a cycle.

Figure 12 shows the results of our algorithm on the proline biosynthesis pathway. We used graphical constraints stating the positions of substrate, product, and polypeptide for biological reactions as well as gene and gene expression for polypeptides.

Figure 13 shows the results on a part of the glycolisys and the TCA cycle. In addition to the graphical constraints defined for the proline, cycle constraints allowed product and substrate in biochemical cycles. As can be seen, our algorithm allows to visualize metabolic pathways including cycles or branches, as well as other biological information such as genetic regulation, signal transduction, etc. Further, users can customize the representation of each kind of biological interaction.

Figure 12. The proline pathway and its genetic regulation.

The computing time of graph layout algorithms constitutes often a bottleneck, since users do not want to wait several minutes for visualizing the regulated metabolic pathway they have just extracted from a database. In order to test the robustness of our algorithm, we randomly generated 30 graphs from 5 to 10,000 nodes, and we successively applied the C^2GL algorithm, the circular layout algorithm, Sugiyama's algorithm (Sugiyama *et al.*, 1981), and a particular force-directed algorithm, Simon's variant (Simon *et al.*, 2000) which is based on spring-embedder, repulsion force, and simulated annealing in order to ensure the convergence of the algorithm (the maximum allowed iterations were fixed to 150). All these algorithms are available in the Visual BioMaze framework. The circular layout algorithm is often used as a reference algorithm for measuring complexity since it has a linear complexity in O(n).

Figure 14 shows the results of the experiments. As can be seen, the C^2GL algorithm is very efficient, the computing time is similar to that obtained by the circular layout algorithm, i.e., our algorithm behaves as having linear complexity. Further, the computing time is below that of Sugiyama's algorithm. For instance our algorithm needs 40 seconds to draw a graph with 10,000 nodes while Sugiyama's algorithm takes 150 seconds.

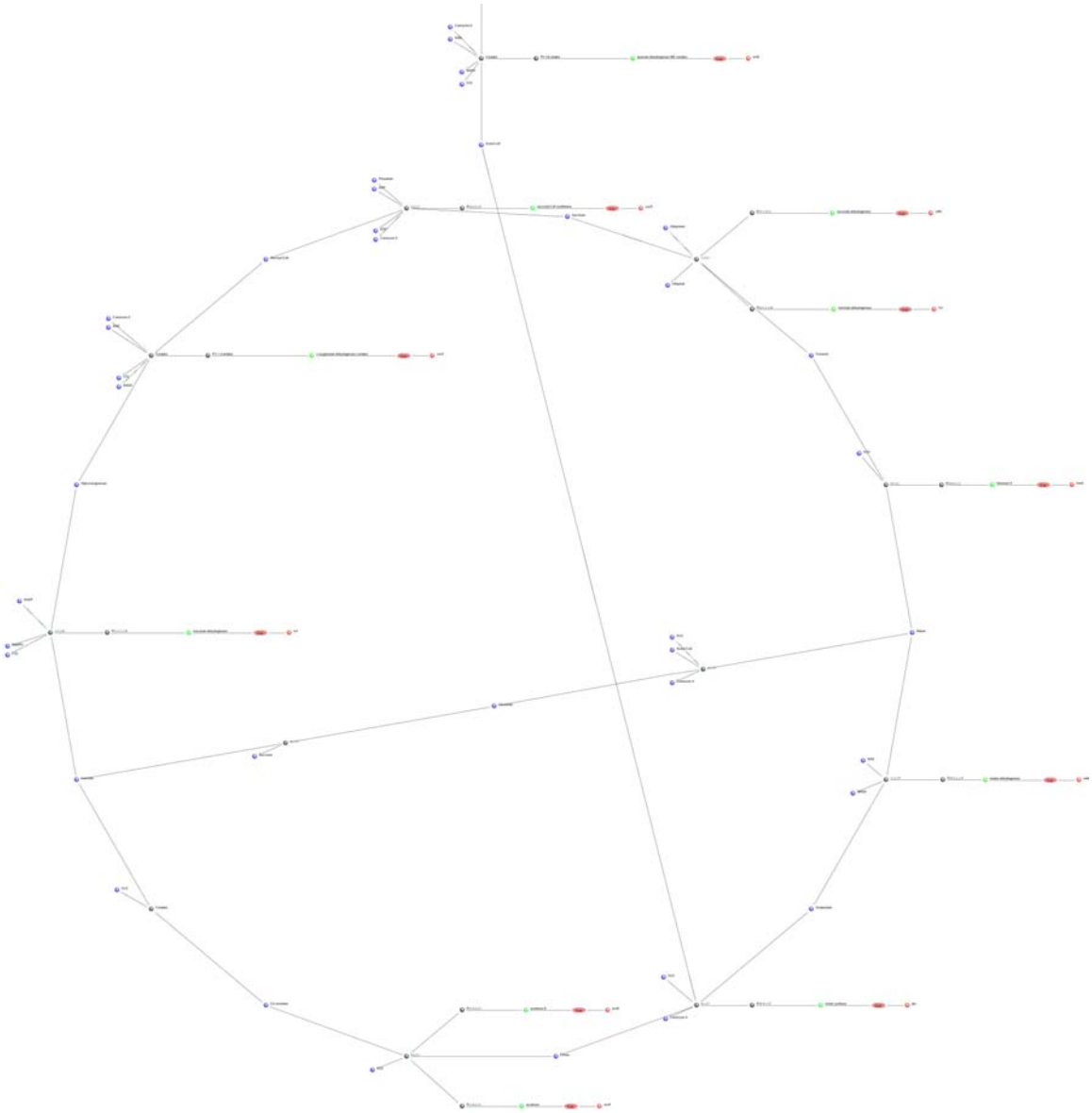

Figure 13. The TCA cycle and its genetic regulation. The figure shows the topological arrangement of a biochemical cycle.

5 Future work

The current version of our algorithm applies a simple layout for the inner cycles of a cycle that has to be drawn along a circle. As already said in Section 4.1, a line is computed between the nodes belonging to the main cycle and the nodes of the inner cycle are placed along this line. However, this algorithm is too simple, for instance it cannot avoid the collision of nodes when two inner cycles cross over each other. This problem is not trivial. The next release will improve the layout of the interior of main cycles.

We will also introduce æsthetic criteria such as minimizing drawing area and minimizing edge crossing. In order to minimize the drawing area, we can apply a method proposed in Castello *et al.*, (2000), which consists in applying a technique similar to the one used for minimizing VLSI chip area.

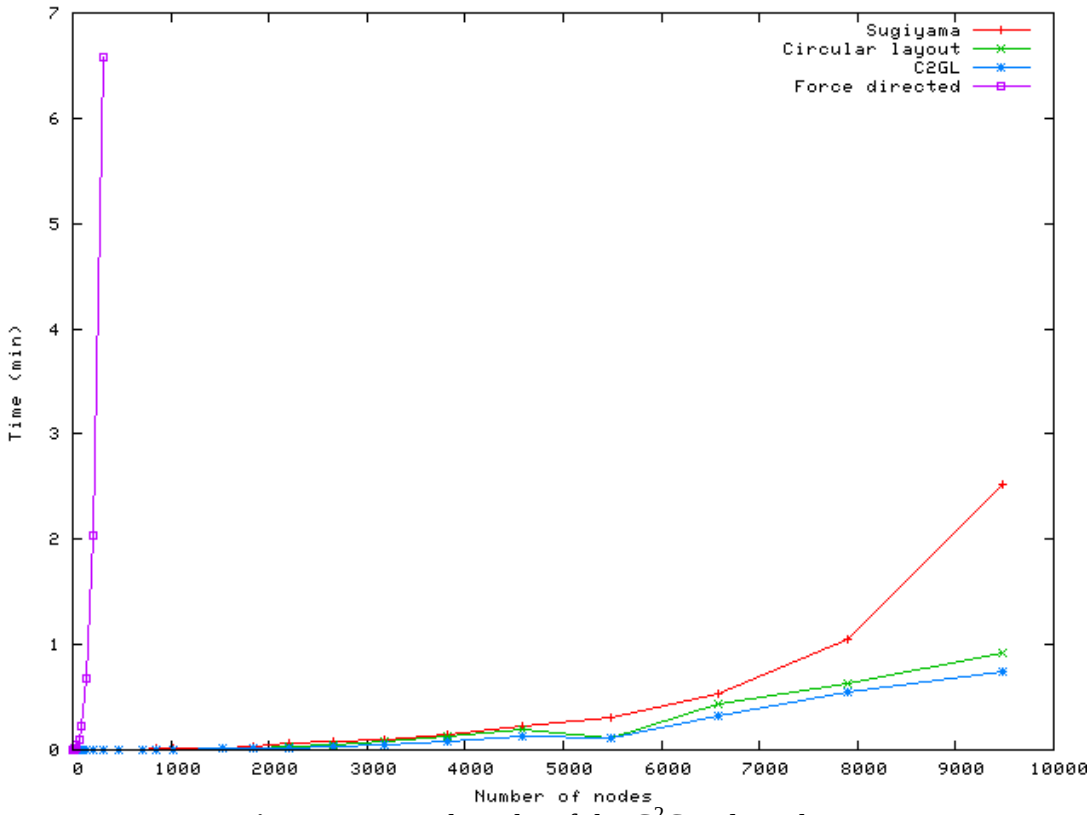

Figure 14. Benchmarks of the C^{2}GL algorithm.

6 Conclusion

Most graph layout algorithms cannot draw a graph according to the specific semantics of an application domain, they only try to increase general æsthetic criteria. For this reason, traditional graph layout algorithms are unable to represent biochemical networks, which are composed by different subgraph types such as signal transduction, metabolic pathway, and genetic regulation. The reason is that each subgraph type has its own semantics and thus, its own drawing rules.

In this paper we described the C^{2}GL algorithm, a constraint-based graph layout algorithm. The algorithm uses graphical constraints to recursively build compound nodes. This approach enables to draw any kind of biochemical network, even in the same graph. In addition, users can customize the representation of each type of biochemical subgraph according to their particular representation rules. Further, our algorithm can cope with branches and cycles in order to produce drawings that conform with usual biochemistry graphical representation rules.

We showed the results of applying our algorithm on genetically-regulated metabolic pathways, such pathways being extracted typically from a database. In addition we evaluated the robustness of the C^{2}GL on 30 random graphs from 5 to 10,000 nodes and we obtained computing times similar to those of an O(n) algorithm.

7 References

R. Castello, R. Mili, & I.G. Tollis (2000). An algorithmic framework for visualizing statecharts. In *Proceedings of the 8th International Symposium on Graph Drawing, GD 2000*, pages 139–149.

R. Davidson & D. Harel (1996). Drawing graphs nicely using simulated annealing. *ACM Transaction on Graphics*, 15(4): 301–331.

G. Dooms, Y. Deville, & P. Dupont (2004a). Constrained path finding in biochemical networks. In *Proceedings of the 5th Open Days in Biology, Computer Science and Mathematics, JOBIM 2004*, pages J0–40.

G. Dooms, Y. Deville, & P. Dupont (2004b). A Mozart implementation of CP(BioNet). In *Proceedings of the 2nd International Conference on Multiparadigm Programming in Mozart/Oz, MOZ 2004*, pages 237–250. LNCS 3389, Springer.

P. Eades (1984). A Heuristic for Graph Drawing. *Congressus Numerantium* 42, pp. 149–160.

P. Eades, X. Lin, & W.F. Smyth (1993). A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323.

P.D. Karp & S.M. Paley (1994). Automated drawing of metabolic pathways. In *Proceedings of the 3rd International Conference on Bioinformatics and Genome Research*.

M. Kaufmann & D. Wagner (2001). *Graph Drawing: Methods and Models*. Springer.

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

P. Mendes, D.L. Bulmore, A.D. Farmer, P.A. Steadman, M.E Waught, & S.T. Wlodek (2000). PathDB: a second generation metabolic database. In *Animating the Cellular Map, Proceedings of the 9th International BioThermoKinetics Meeting*, pages 207–212. Stellenbosch University Press.

F. Schreiber (2002). High quality visualization of biochemical pathways in BioPath. In *Silico Biology*, 2(0006).

A. Siepel, A. Farmer, A. Tolopko, M. Zhuang, P. Mendes, W. Beavis, & B. Sobral (2001). ISYS: A decentralized, component-based approach to the integration of heterogeneous bioinformatics resources. *Bioinformatics*, 17(1):83–94.

F. Simon, F. Steinbruckner, & C. Lewerentz (2000). *3D-Spring embedder for complete graphs*. Technical report, Technical University Cottbus, Germany.

S. Skhiri dit Gabouje (2005). *The Visual BioMaze Tutorial*. http://cs.ulb.ac.be/research/ biomaze.

S. Skhiri dit Gabouje & E. Zimányi (2005). Generic visualization of biochemical networks: A new compound graph layout algorithm. In *Poster Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms, WEA 05*.

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

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

K. Sugiyama & K. Misue (1995). A simple and unified method for drawing graphs: Magnetic-spring algorithm. In *Proceedings of the DIMACS International Workshop on Graph Drawing, GD'94*, pages 364-375. LNCS 894, Springer.

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

R. Tarjan (1972). Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160.

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

S. Zampelli, Y. Deville, & P. Dupont (2004). Finding patterns in biochemical networks: A constraint programming approach. In *Proceedings of the 5th Open Days in Biology, Computer Science and Mathematics, JOBIM 2004*, pages J0–85.

E. Zimányi & S. Skhiri dit Gabouje (2004). Semantic visualization of biochemical databases. In Semantic for GRID Databases: Proceedings of the International Conference on Semantics for a Networked World, ICSNW04, pages 199–214. LNCS 3226, Springer.