Dynamic visualization of metabolic pathways

combining mental maps and static metabolic pathway rendering techniques

Direks, G.L.F.

Award date:
2014

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Author
Gijs Direks,
e-mail: g.l.f.direks@student.tue.nl

Supervisors
Michel Westenberg
Kasper Dinkla

Date
December 20, 2013

Dynamic Visualization of Metabolic Pathways

Combining Mental Maps and Static Metabolic Pathway Rendering Techniques

Abstract

In this thesis, we identify the requirements of creating static metabolic pathway layouts and maintaining the mental map when adding additional pathways to an existing visualization in an interactive setting. We describe methods which accomplish a good static layout, as well as methods which maintain the mental map. We combine a static layout algorithm by Karp and Paley with a mental map preserving algorithm by Misue et al. with good results. We also detail how to combine other algorithms to achieve a similar effect.

We take special care when trying to find a good balance between maintaining the mental map and creating a clear layout. This results in an algorithm which is good at maintaining the mental map when combining multiple metabolic pathways.

The prototype application which implements this algorithm generates these layouts quickly, without any noticeable delay. While the prototype is algorithmically complete, it can use some aesthetic improvements.

Technische Universiteit
Eindhoven
University of Technology

Table of contents

Title
Dynamic Visualization of Metabolic Pathways

Where innovation starts

Table of contents

Title
Dynamic Visualization of Metabolic Pathways

1 Introduction

The processes that occur in a living cell are numerous and complex. In the field of biology, processes that describe metabolism in an organism are modelled by metabolic networks. Typically biologists are not interested in the entire metabolic network, but rather in small predefined parts called metabolic pathways. Biologists often use visualizations of these pathways to better understand the processes. Several databases such as MetaCyc and its Pathway Tools siblings [4], Reactome [21] and KEGG [8] provide static visualization of these pathways.

The focus of automatic pathway rendering not only lies on creating a clear representation, but to also mimic textbook examples of these pathways. Biologists are already used to interpreting networks drawn in this style. There has been significant work towards automated metabolic pathway rendering. Given a pathway, it is possible to create (near-)textbook results for the layouts of the pathway. For example, MetaCyc uses an algorithm by Karp and Paley [9] to draw all its pathways. On the other hand, KEGG uses hand-drawn pathways. This results in cleaner layouts than similar pathways in MetaCyc (especially when the pathways are large), but it takes much time to produce such drawings for all pathways.

Merely presenting the user with small pieces of the metabolic network without any interaction is not good enough. The databases mentioned all allow the user to explore these pathways more dynamically: clicking on enzymes and reactions reveals more information, and exposes which pathways are connected to the current one. This allows the user to explore the pathway by clicking through to adjacent information. However, the user cannot combine multiple pathways together to form a new, larger pathway.

In this thesis, we explore the possibilities of combining pathways interactively.

This last feature is not a trivial consequence of being able to draw single pathways nicely. When combining two (or more) pathways, we could just create an entirely new layout for the new, larger, pathway. However, when the user has already studied one of these pathways, recreating the layout may destroy the understanding the user already had of this pathway.

This is were the concept of a mental map comes into play. The mental map is the way the user has sorted the information in the pathway, for example by placement, topology or clustering. This mental map allows the user to quickly find and identify different aspects of the pathway. If we can preserve the mental map when combining pathways, the user might be better able to quickly grasp the larger, more complex pathway.

Preserving the mental map is by definition a somewhat vague concept. Like defining a layout, we can merely define what we think helps preserve the mental map and then create algorithms which conform to those ideas. For example, Misue et al. [15] give multiple properties to preserve, from which they choose one: the preservation of horizontal and vertical ordering of nodes. They then present their force scan algorithm which accomplishes this.

We will further examine the concepts of metabolic pathways and mental maps in Chapter 2. In Chapter 3 we will discuss static metabolic pathway layout, stating requirements and identifying the elements we wish to layout. We explain algorithms which create these layouts in Chapter 4. Chapter 5 describes some example algorithms for mental map preservation. In Chapter 6 we combine static layout techniques with mental map preserving algorithms, giving the user the opportunity to interactively add pathways to the currently shown one.

2 Background

2.1 Metabolic pathways

A full-fledged metabolic pathway contains many pieces of information: the metabolic process converts compounds to other compounds via chemical reactions. A reaction can convert on or more reactant compounds (or *substrates*) into one or more product compounds. These reactions might be catalysed by enzymes. Pathways describe reactions which typically occur in multiple organisms, though different organisms may utilize different enzymes to catalyse the same reaction.

Reactions can be connected to other reactions: the product of one reaction can be a substrate in the next. All compounds in a reaction (enzymes, reactants and products) have a chemical structure and might have one or more common names.

We can see an example of a metabolic pathway in Figure 2.1. Here we can see the citric acid cycle, which clearly identifies different aspects of the reaction. In a glance, the user can see that this pathway is a cyclic process (due to the circular form), he can identify enzymes (due to the blue color and placement of the enzyme names) and he can see the difference between main compounds and side compounds (see Section 2.1.1 for more details). In this pathway we can also see that some reactions can convert compounds both from left to right and from right to left, i.e. they are reversible.

Databases such as MetaCyc and its Pathway Tools siblings [4], Reactome [21] and KEGG [8] provide this information, along with a lot more data: which genes are responsible for the creation of certain compounds, or in which organisms a given pathway exists.

Reactions take place in certain parts of the cell: certain reactions might only take place in the cell nucleus while others take place in the mitochondria. Certain reactions might only transport a compound from one area to another.

2.1.1 Main Compounds and Side Compounds

Metabolic pathways consist of main compounds and side compounds. The main compounds are the most important chemicals in the pathway. For example, compounds which are created by one reaction and then used by another are main compounds. Side compounds are those compounds which only contribute some energy or a few atoms to the process. For example, ATP is frequently used as an energy source in a reaction, which results in a by-product of ADP. In most pathways, we can consider ATP and ADP to be side compounds.

Figure 2.1 shows for an example image of the citric acid cycle. We can see a distinct difference between main compounds and side compounds: black indicates main compounds, red indicates side

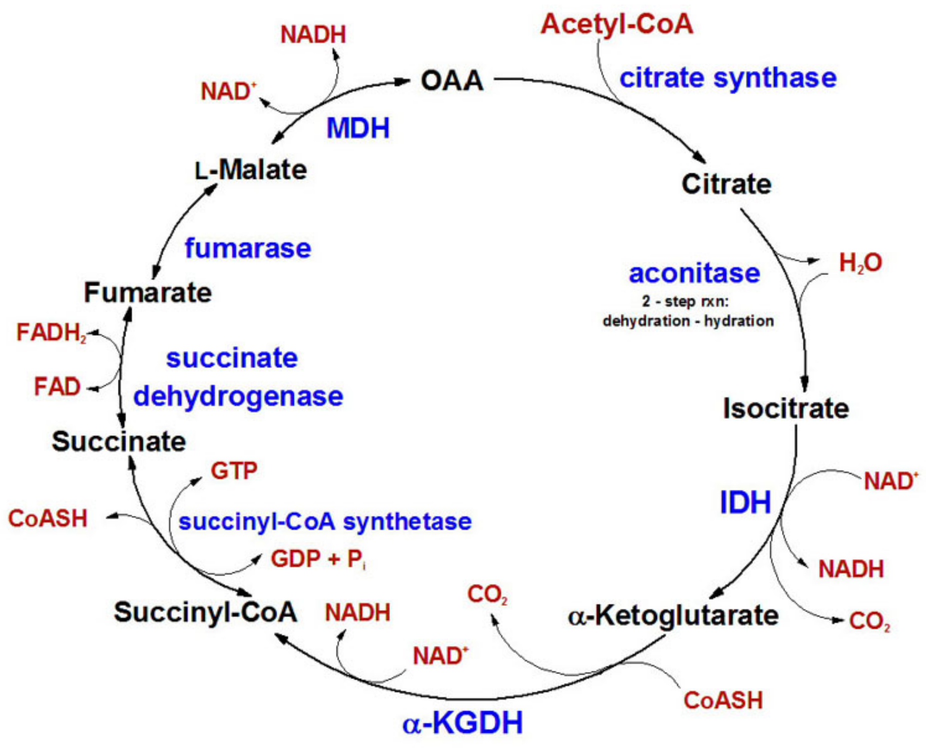


Figure 2.1: Citric acid cycle. In this image, we can identify compounds (black text), enzymes (blue text) and side compounds (red text). Every hyperedge represents a reaction, with arrows indicating the reaction direction (left to right, right to left, or both).
Image from `themedicalbiochemistrypage.org/tca-cycle.php`*, November 2013.*

compounds. Where needed, the red side compounds are duplicated - there are multiple instances of the NADH and NAD^+ molecules.

When we have a pathway, we can assume that compounds which are both substrate and product are main compounds: these are the compounds which link the reactions together. Removing them would have significant impact on the network structure. Compounds which are only substrate or only product might be side compounds. However, a product (or substrate) is only a side compound if it is parallel to another product (or substrate).

In Figure 2.4a and 2.4b we can see the difference between a pathway drawn with and without side compounds. Removing the side compounds does not alter the linear structure of the pathway.

KEGG [8], Reactome [21] and MetaCyc [4] do not show the difference between main compounds and side compounds explicitly. MetaCyc is the only one with a visual clue: it connects side compounds with curved edges instead of straight ones.

2.1.2 Pathway Exploration

In this section, we will discuss some forms of pathway exploration already available to users. We will start by comparing Reactome [21] and MetaCyc [4] and KEGG [8], the databases we have already mentioned, and give some other examples of pathway exploration tools. We will limit ourselves to the utilities provided for pathway exploration. We will not address other properties of the databases, like gene exploration or quality control of the content.

KEGG

The KEGG web interface provides the user with hand-drawn images. As a consequence, pathway exploration is very static: the user can inspect one pathway, then click on certain elements to open a new picture of that pathway. No other interaction is provided.

Figure 2.2 shows the glycolysis pathway in KEGG. Elements (such as compounds, enzymes, reactions or pathways) are clickable and open a new page with more information about that element.

The user is able to navigate through pathways in multiple ways: he can click on elements in the current pathway, he can search for pathways or scroll through a list. KEGG also provides some overview pathways, such as carbon metabolism or the entire metabolic network. Figure 2.3 shows a part of the carbon metabolism overview. On the left hand side we see the pathway groups and individual pathways contained in the overview. The user is able to select one or more of these. On the right hand side, we can see the layout of the combined pathways. The selected pathways are highlighted in color, allowing the user to examine specific pathways more easily, while still maintaining the context of the larger network. Again, the user is able to click through to see individual pathways.

MetaCyc

Much like KEGG, MetaCyc provides the user with a static image. However, the user is able to select multiple levels of detail for each image, allowing a more compact or more detailed view to be selected. The user is also able to obtain more information about elements by clicking them. Some of this information, like the chemical structure of compounds, can also be obtained by selecting a more detailed view, allowing the user to follow the evolution of a chemical through the pathway.

Figure 2.4 shows how MetaCyc [4] presents the user with different detail levels to aid in understanding. The lowest detail setting allows the user to see only the most important parts of a pathway: the names of the main compounds as well as reaction directions. When increasing the level of detail, the user is presented with more information of the pathway, but potentially loses the ability to comprehend larger pathways quickly. When we compare Figure 2.4a to Figure 2.4d, the former is much clearer in representing a linear pathway while the latter presents more information.

Reactome

Reactome works much the same way as KEGG and MetaCyc, providing the user with a pathway which can be clicked through. Like KEGG, Reactome allows the user to select parts of the pathway, which will be highlighted. Reactome produces pathway layouts which include locational information: the user can see in which parts of the cell certain reactions take place. In Figure 2.5 we can see how Reactome represents locational information.

Reactome provides the user with a hierarchical view of all pathways, of which the user then can select smaller pathways. These are then highlighted. Like KEGG, this allows the user to examine pathways in relation to the surrounding pathways.

Complete Metabolic Network

The famous Biochemical Pathways poster created by Michal [14] shows the entire metabolic network. All layout was done by hand. This poster in itself is not a very useful tool to explore pathways, but it was the basis for ExPASy [1]. ExPASy incorporated scanned parts of the Biochemical Pathways poster, allowing the user to explore links between pathways electronically.

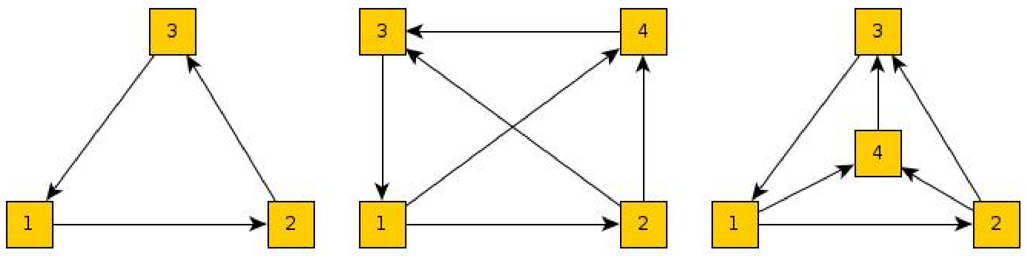
Lambert et al. [12] propose a method to create a layout for the entire metabolic network, while keeping individual pathways recognisable. They achieve this by allowing nodes to be duplicated, reducing the complexity of the layout significantly.

2.2 Mental Maps

The focus of most layout algorithms is on creating the best looking image given a certain input graph. However, when a user interacts with a graph (by adding or removing nodes or edges), the image can drastically change - even with only small changes to the input graph.

When a user examines a graph, he makes a mental map of which pieces are important and what structures are present within a graph. When the graph is changed slightly, the user should be able to recognize these pieces and place any newly inserted nodes into his mental map.

A small example can be seen in Figure 2.6. Here, the user may have studied the small triangle and formed a mental map of its structure. Then, when node 4 is added, the layout is adjusted and the mental map the user had of the triangle is destroyed. We could have preserved the mental map by keeping the triangular layout of the initial graph and adding node 4 in the center.



(a) A small graph. *(b) Mental map is broken.* *(c) Mental map is preserved.*

Figure 2.6: Mental map problem: when adding a fourth node to the triangle in (a), we may destroy the mental map if we naively create an entirely new layout. This can happen when we produce (b). The mental map is better preserved in (c).

2.2.1 Criteria

There are several aspects as to what constitutes the mental map. Eades et al. [7] provide some concrete aspects of a model of the mental map. These models are used in their paper on the Force Scan method

[15] (see also Section 5.1). Bridgeman et al. [3] also provide criteria to maintain the mental map. In the following, some of these aspects and criteria are described in more detail.

One very simple way to preserve most aspects mentioned below, is to just place new nodes arbitrarily. This has the distinct drawback that new nodes may overlap with other nodes, edges or labels. A simple scaling algorithm could be applied overcome this. However, this either destroys the readability of the labels (when we shrink nodes, or zoom out); or it destroys the users ability to see the entire graph at once (when we zoom in, or place nodes further apart).

As such, any layout adjustment algorithm must not adjust node sizes, while striving to maintain a similar layout size. This size can be determined by, for example, screen size or paper size (when printing graphs).

Orthogonal Ordering and Shape

The most simple aspect of a mental map is the orthogonal ordering of the nodes. An adjustment to the layout should therefore strive to keep this ordering intact. In other words, if a node is to the left of another node, it must be to the left of that node after layout adjustment.

Both Eades and Bridgeman consider this aspect. Bridgeman et al. expand on this concept by taking a node's *rank* and the layout's *shape* into account. The rank of a node is determined by the amount of nodes to the right and above it. Thus, maintaining orthogonal ordering also maintains the rank of every node and vice versa.

The shape of a layout pertains to the relative positions of connected nodes. For each edge, we record the direction as one of the eight cardinal and intermediate directions (N, E, S, W, NE, SE, NW, SW). Then, the number of edges that maintain their direction during layout adjustment is an indicator for the preservation of the mental map.

Displacement of Nodes

Ideally, we would like nodes to be displaced as little as possible.

Eades et al. measure this by creating several graphs which model proximity between nodes. Examples include Gabriel graphs, minimum spanning trees, relative neighbourhood graphs, and Delaunay triangulations. Preservation of the mental map during layout adjustment can then be measured in terms of (one of) these graphs. We can try to maintain the proximity graph, or choose to keep changes to these graphs to a minimum.

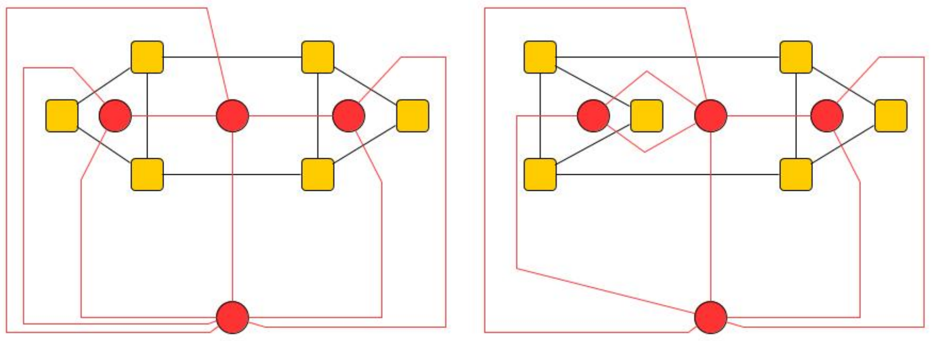
Bridgeman et al. define some criteria which model clustering and distribution of nodes. The *average relative distance* measures how nodes are distributed, it is the average of the distances between pairs of nodes (connected or unconnected).

They also use a *nearest neighbour* criterion: if a node v is the nearest neighbour of u before layout adjustment, then we also want this to be the case after layout adjustment. This criterion captures relative displacement. If we count the number of nodes which are nearer to u than v is, we obtain a cost function for which we can optimize.

Another nearest neighbour criterion captures the fact that we want the user to be able to look at a spot on the screen and quickly find the node he expects to be there. For each node v, we have an original location p_v. After layout adjustment, we count the number of nodes which are closer to p_v than v is.

Topology

Eades et al. define topology in terms of dual graphs: A face of a graph is a region enclosed between edges in the graph. For a graph G we can construct its dual graph D. Every face in G corresponds to a node in D, there is an edge between a pair of nodes in D if and only if their corresponding faces in G are adjacent.



(a) Two dual graphs. *(b) Changes in layout affect the dual graph.*

Figure 2.7: Dual graphs. In (a) the red and yellow graphs are duals of eachother. In (b) we change the topology of the yellow graph, resulting in a different dual graph. In (a) the center node is connected to four edges, in (b) it is connected to five.

When a layout adjustment of G does not affect its dual graph, we say that the adjustment preserves topology. This also aids in preservation of the mental map.

Bridgeman et al. look at every triplet of nodes u,v,w. If u,v,w is ordered clockwise (or counter-clockwise) in the original layout, we want this to also be the case after adjustment. We can then try to preserve the mental map by minimizing the number of triplets which change orientation.

2.2.2 Effectiveness

Several studies have been performed as to whether the preservation of the mental map actually aids the user's understanding.

Purchase et al. [16] have performed a user study on twenty people indicating that mental maps do aid in the understanding of a graph. Participants were asked four questions about information contained in dynamic graphs. In two of the four questions, preservation of the mental map increased performance in terms of response time. In the other two questions, no difference was measured.

In another user study, Purchase et al. [17] provide insight in the usage of mental map preserving techniques: they have found that trying to mix good layout and preserving the mental map adversely affects the user's performance. They state that the best course of action is to take an extreme approach: either focus completely on preserving the mental map, possibly creating a very poorly laid out graph;

or focus on presenting the user with a clear picture, completely ignoring any mental map the user might have had of a previous result. Trying to mix these two requirements by making a somewhat good layout and preserving parts of the mental map results in worse performance than taking either extreme.

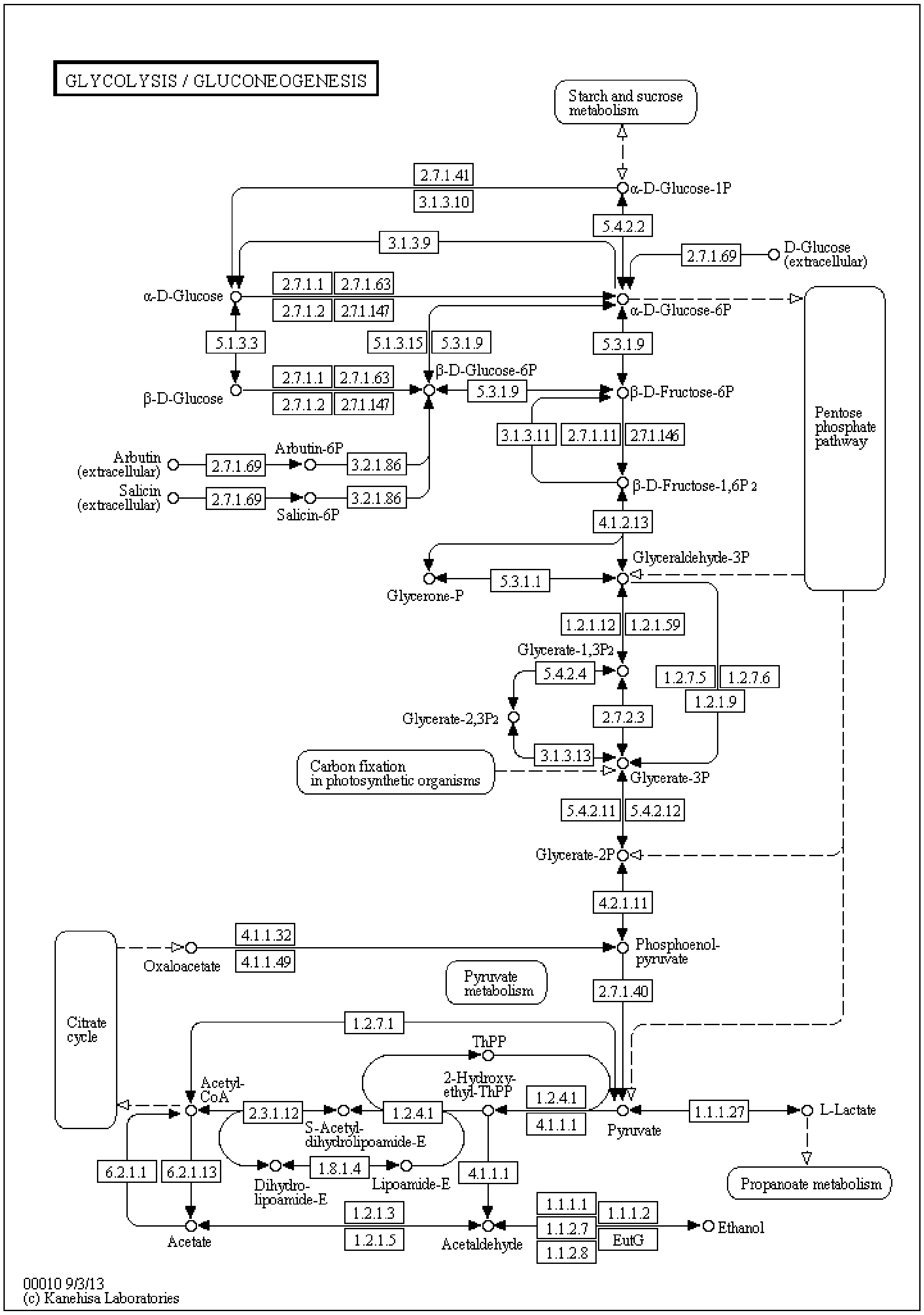


Figure 2.2: Glycolysis as shown by KEGG [8]. The user is able to click on each element to obtain information, but no additional information can be accessed directly.

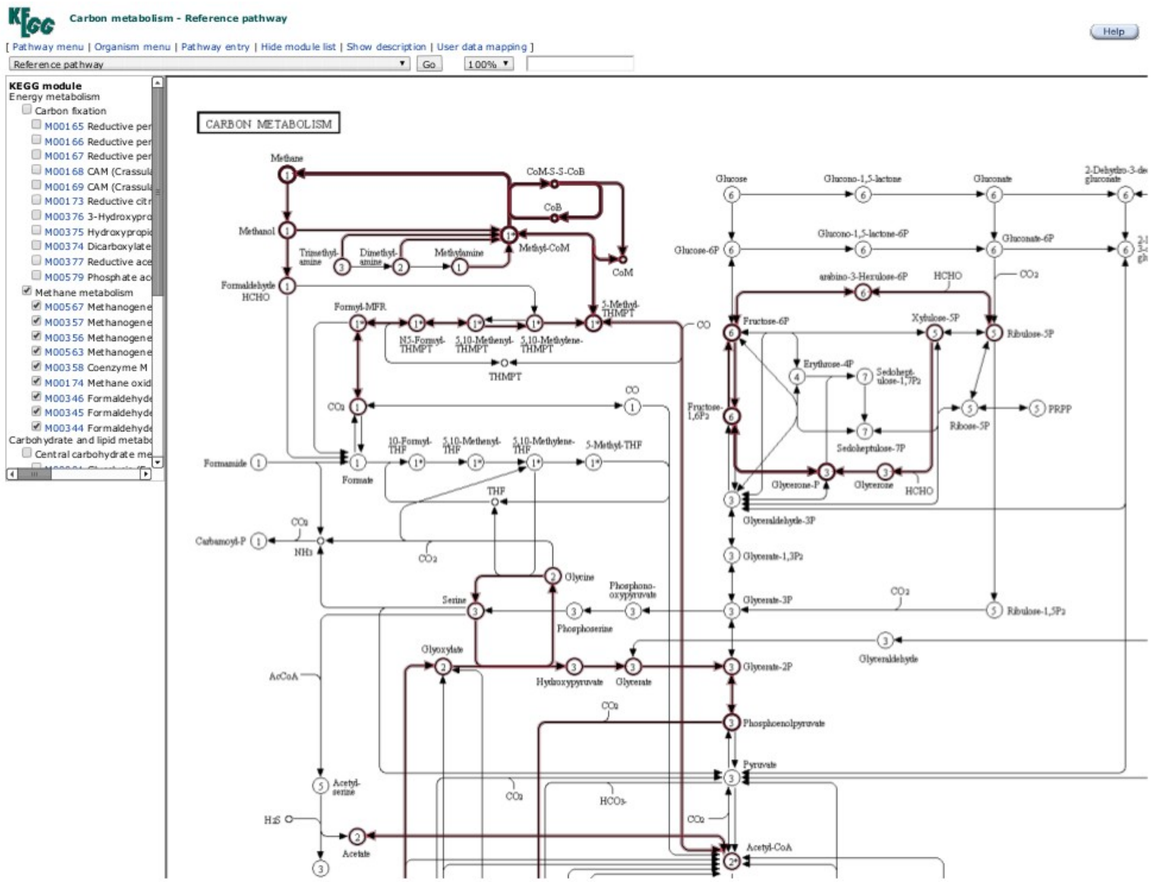
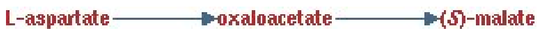
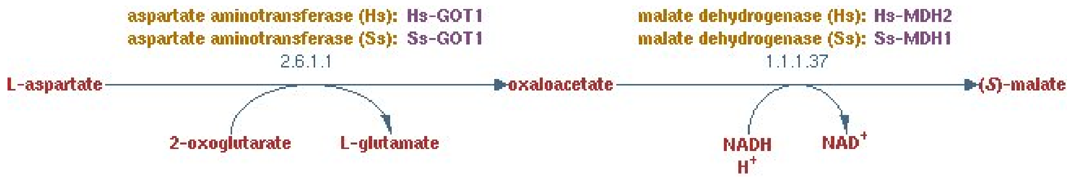


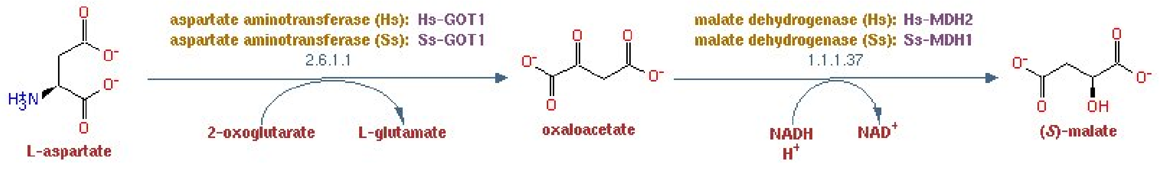
Figure 2.3: Carbon metabolism as shown by KEGG [8]. The user is able to select one or more pathways or pathway groups on the left, which will be highlighted. As in normal pathway exploration, the user is still able to click on elements to obtain more information.



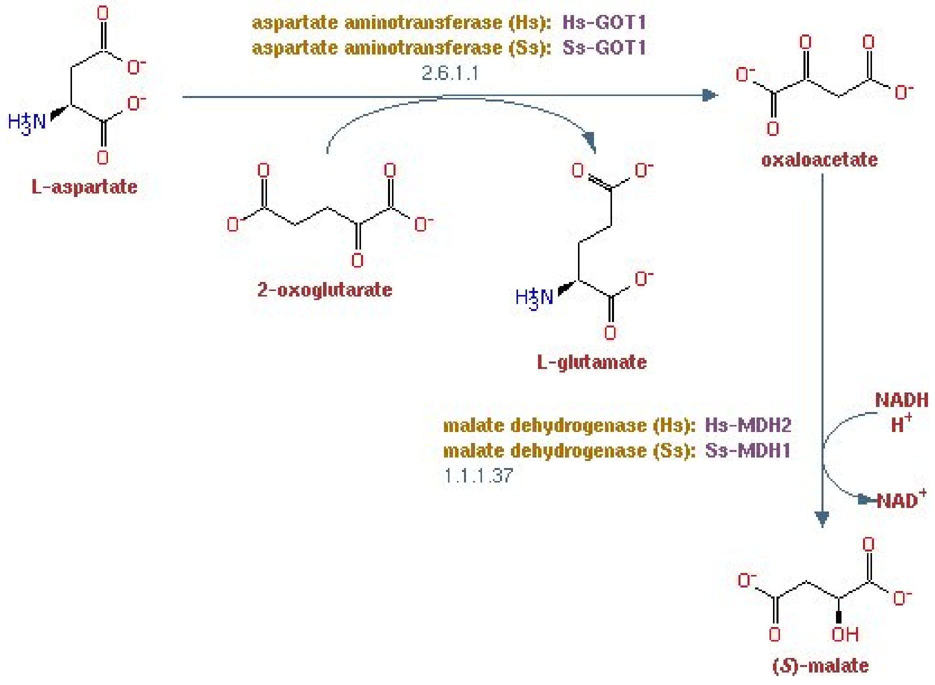
(a) Lowest detail. Only the names of the compounds and reaction directions are shown.


aspartate aminotransferase (Hs): Hs-GOT1
aspartate aminotransferase (Ss): Ss-GOT1
2.6.1.1
malate dehydrogenase (Hs): Hs-MDH2
malate dehydrogenase (Ss): Ss-MDH1
1.1.1.37
L-aspartate
oxaloacetate
(S)-malate
2-oxoglutarate
L-glutamate
NADH
H+
NAD+

(b) Medium detail. Here we also see the names of side compounds and the enzymes which catalyse the reactions.


aspartate aminotransferase (Hs): Hs-GOT1
aspartate aminotransferase (Ss): Ss-GOT1
2.6.1.1
malate dehydrogenase (Hs): Hs-MDH2
malate dehydrogenase (Ss): Ss-MDH1
1.1.1.37
L-aspartate
oxaloacetate
(S)-malate
2-oxoglutarate
L-glutamate
NADH
H+
NAD+

(c) High detail. The chemical structure of the main compounds is included.


aspartate aminotransferase (Hs): Hs-GOT1
aspartate aminotransferase (Ss): Ss-GOT1
2.6.1.1
L-aspartate
oxaloacetate
2-oxoglutarate
L-glutamate
malate dehydrogenase (Hs): Hs-MDH2
malate dehydrogenase (Ss): Ss-MDH1
1.1.1.37
NADH
H+
NAD+
(S)-malate

(d) Highest detail. This also shows the chemical structure of the side compounds.

Figure 2.4: Aspartate degradation with different detail levels, as drawn in MetaCyc.[4] Increasing detail levels add enzymes, side compounds and chemical structures.

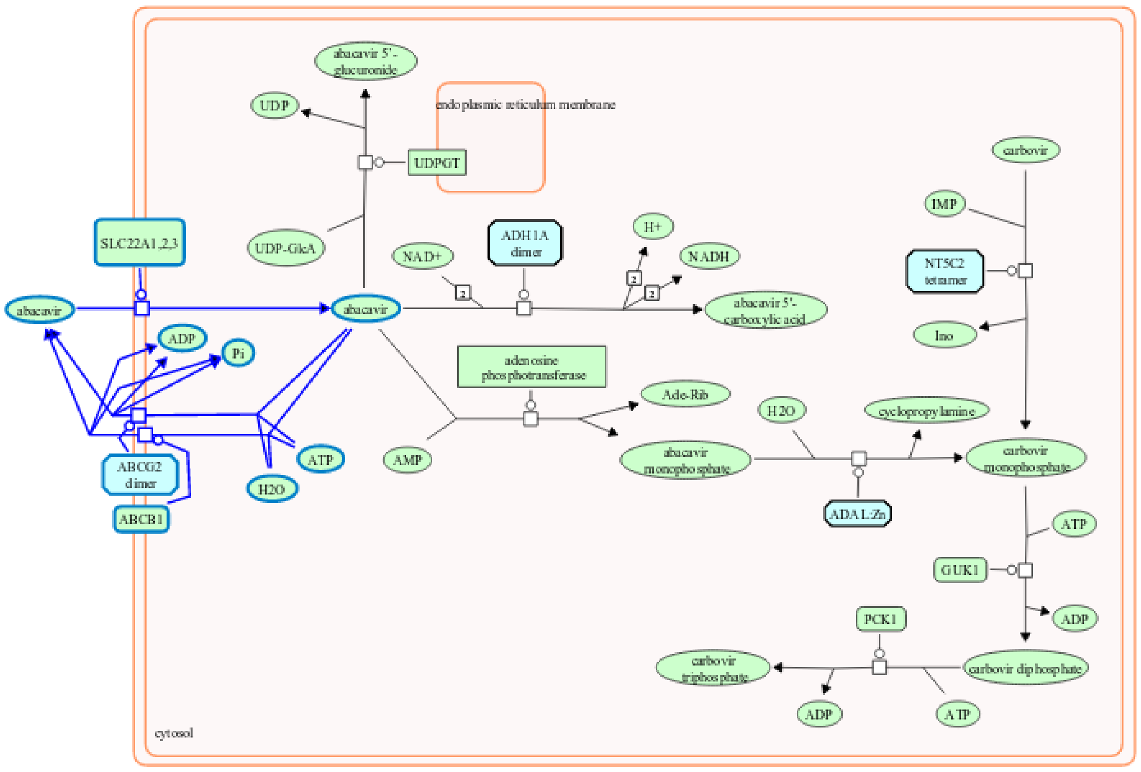


Figure 2.5: Abacavir transport as shown in Reactome [21]. We can see which reactions take place in different parts of the cell (e.g. cytosol, endoplasmic reticulum membrane). The user has selected one of the pathways contained in the picture, which is highlighted in blue.

3 Requirements

As we have mentioned in Chapter 2, there are many data elements which can be visualized in metabolic pathways. In this chapter, we will define the requirements for our application.

We develop an approach to explore pathways, by not only looking at a single pathway, but by also being able to mix in other pathways and examine them as a whole. Most importantly, we want the user to maintain his mental map of the pathway he has studied when adding new elements.

3.1 Pathways

The following requirements hold for the currently selected pathway:

1. Currently selected pathways are shown.
2. Compounds are identifiable by their names.
3. Main compounds and side compounds are visually distinct.
4. Reactions, including reaction directions and enzymes are shown.
5. Neighbouring pathways are shown.
6. Pathway structure is easily understandable.

When we require the pathway to be drawn as a simple graph, we can easily satisfy requirements 2-4. We then make every compound a node in the graph, with directed edges showing reaction directions. Edge labels represent enzymes. This very closely resembles the textbook examples we wish to emulate.

The following data elements, while important to biologists, are outside the scope of our application:

- Location of reaction in the cell.
- Organisms in which the reaction or enzyme is found.
- Chemical structure of compounds.
- Gene information.

3.2 Interaction

The user should be able to browse the metabolic network in a pathway-driven manner. To achieve this, we have the following requirements:

1. The user can select a pathway to replace the current graph.
2. The user must be able to select adjacent pathways to add to the current graph.
3. When adding a pathway to the current graph, the layout adjustment focusses on maintaining the mental map.
4. The user is able to redraw the current graph to obtain an entirely new layout.

Requirements 3 and 4 are based upon the findings by Purchase et al. [17], which indicate that we should either focus on drawing the graph nicely, or on maintaining the mental map, but we should not combine the two goals. See Section 2.2.2 for more details.

4 Static Network Layout

Automatic drawing of metabolic pathways has been studied extensively [1, 2, 9, 12, 18, 19]. In this section, we will describe three methods to nicely draw these pathways. We have selected these methods by their ability to be combined with the mental map preserving techniques described in Chapter 5.

The method proposed by Karp and Paley [9] has been developed for MetaCyc, we describe it in Section 4.1. Becker and Rojas [2] build upon this method, as can be seen in Section 4.2. Schreiber [18] takes a completely different approach, transforming the network into a directed acyclic graph and using a layered layout algorithm. We can see this in Section 4.3.

4.1 Karp (MetaCyc)

Karp and Paley [9] have introduced a method which recursively layouts the metabolic network. They identify the type of network as *Linear*, *Branched*, *Cyclic* or *Complex* and use different layout algorithms accordingly. Figure 4.1 shows examples of these types. The basis of this method relies on the fact that the layout of a complex pathway can be made by first defining a layout for the smaller parts (which are easy to lay out) and then linking them together.

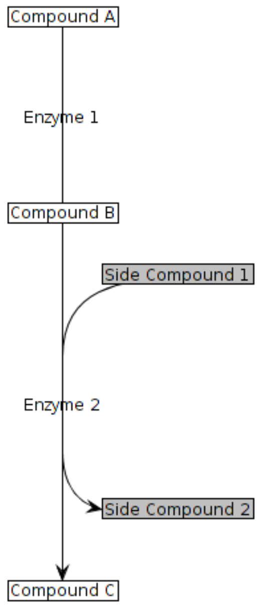
First, the side compounds of the network are stripped. Then, the remaining network is identified as one of four types, and a layout is created accordingly. Afterwards, the side compounds are positioned. This requires the main layout step to reserve space for the side compounds.

We will describe the three basic layout algorithms as well as the complex algorithm.

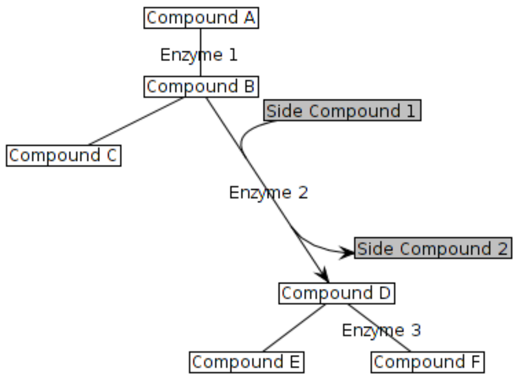
4.1.1 Linear Layout

A linear pathway is one where the main compounds form a simple path. Only side compounds can cause branches in this path.

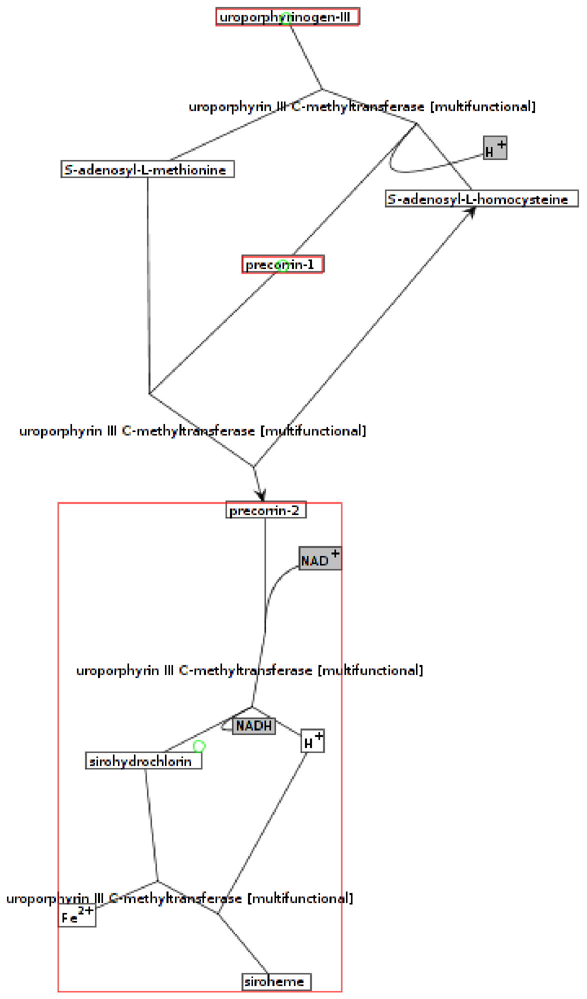
When creating a layout which must fit inside a predefined rectangular area (e.g. in a book or on a screen) the layout of these types of pathways may result in a snake layout, where the path is drawn across multiple lines. Karp and Paley give some edge cases for this type of layout. However, since we do not have such size restrictions, we can just draw the vertices on a line and place the side compounds accordingly.



(a) Linear pathway.



(b) Branched pathway.


uroporphyrinogen-III
uroporphyrin III C-methyltransferase [multifunctional]
S-adenosyl-L-methionine
H+
S-adenosyl-L-homocysteine
precorrin-1
uroporphyrin III C-methyltransferase [multifunctional]
precorrin-2
NAD+
uroporphyrin III C-methyltransferase [multifunctional]
NADH
sirohydrochlorin
H+
uroporphyrin III C-methyltransferase [multifunctional]
Fe2+
siroheme

(c) Complex pathway. Each supernode highlighted by a red bounding box and a green circle to mark its center.

Figure 4.1: Network types. In all figures, side compounds are drawn grey. In (a) we can see a linear pathway, in (b) a branched pathway. In (c) we can see the main cycle, along with three supernodes (with red bounding boxes). The bottom supernode is itself complex.

4.1.2 Branched Layout

A branched pathway is one where the main compounds form a tree. By definition, side compounds cannot cause cycles in a pathway (since that would require them to be connected to multiple main compounds).

A branched pathway uses a tree layout, with the additional requirement that side compounds can be placed without overlapping main compounds. Since the side compound placement algorithm is known, the algorithm can reserve space where needed.

4.1.3 Circular Layout

A pathway is circular when the main compounds form a cycle without branches.

The algorithm chooses a node to be at the top of the cycle, then divides the nodes into two sets: A and B. These sets will be placed on the left and right sides of the circle, respectively. The top node will be present in both sets. Then, a diameter D for the circle is chosen such that all compounds can be placed on the circle without overlap. D is chosen to be the maximum height of A or B, where the height of a set is the sum of the height of each node, plus some vertical space after each node.

Then, each node is assigned its y-coordinate: $y_i = \frac{iD}{2n}$. Since the algorithm included vertical spacing after each node, the two bottom nodes will not overlap.

4.1.4 Complex Layout

A pathway is complex when it is not one of the previous types. This means that when after the network is stripped of side compounds, there is at least one cycle left, as well as some branches which do not belong to this cycle.

First, the largest cycle in the pathway is computed. In general, this problem is NP-complete. However, since the pathways have a limited number of nodes and generally only a few cycles, it is possible to find the largest cycle efficiently.

Let S be the set of all nodes in the largest cycle. Let R be all nodes not in the cycle. A layout for S is then created, as well as layouts for each connected component in R. If any components in R are complex, the algorithm recursively creates a complex layout for those components. Side compounds are placed in these layouts as well.

R is divided in two groups, I and O, which will be placed inside and outside the cycle, respectively. All components in R connected to S by one edge will be placed in O (and drawn outside the circle), while any component connected to S by more than one edge will be placed in I (and drawn inside the circle).

Since all components in R are connected only to nodes in S (by definition), a tree layout algorithm is used to place all components.

4.1.5 Enzymes and Side Compounds

Karp gives suggestions about the placement of enzymes and side compounds. In general, side compounds should be placed above or to the right of the main reaction edge, with any enzymes placed on the opposite side. In the case of a cycle, both the enzymes and side compounds should be placed on the outside of the circle.

4.2 Becker

Becker and Rojas [2] improve upon the method by Karp and Paley [9]. Like Karp and Paley, they lay out the largest cycle, and decide which components must be drawn on the inside and the outside of this cycle. Then, they choose an initial placement for all supernodes and let a spring embedder create a desired layout for the supernodes. A post-processing step is performed to heuristically improve the final result.

4.2.1 Center of Mass and Preferred Orientation

Every supernode (which already has been laid out) has some vertices which connect to the main cycle. The average location of each of these nodes determines a supernode's center of mass. This center of mass is an indication of where most edges will connect to the supernode.

The *preferred orientation* of a node is determined by the difference between the center of a supernode's bounding box and its center of mass. For example, if the center of mass lies directly beneath the center of the supernode, we would prefer this supernode to be on the northern side of the cycle. This placement tries to minimize the number of edge crossings, since edges will most likely connect to the southern part of the supernode.

4.2.2 Initial Placement

To calculate the initial layout, Becker and Rojas first calculate the layout of all supernodes and determine whether they will be placed on the inside or the outside of the main cycle. They use a different criterion for this than Karp and Paley do: a supernode will be placed inside the main cycle if it consists of exactly one node and is connected to the main cycle by at least two edges. This means that any non-trivial supernode will be placed on the outside of the cycle.

Then, the preferred orientation of each supernode is calculated. Since it is impossible to satisfy the preferences of all supernodes simultaneously, the preference of the largest supernode is satisfied. To do this, the main cycle is rotated until one of the nodes connected to the largest supernode lies on the line from the center of the cycle, through the center of mass to the center of the supernode.

Then, all other outer supernodes are placed according to their preferred orientation. All supernodes which are placed inside the cycle are initially placed at its center, the spring embedder will create a suitable layout for these nodes.

4.2.3 Spring Embedder

Most standard spring embedding algorithms assume the nodes are points. For metabolic pathway rendering this is not good enough, since nodes cannot to overlap; especially since the supernodes can become very large. Thus, a customized spring embedder which takes node sizes into account is used.

After all nodes have been set to their initial positions, the spring embedder positions them in a visually pleasing way, ensuring no nodes overlap.

A spring embedder is a force directed layout algorithm which simulates physical forces on nodes and edges. At every step, all forces are computed, which are then applied to the nodes. This step is then repeated for a set number of iterations, or until the changes in the layout are small enough.

Typically, nodes repel each other, while edges attract their endpoints towards each other. The forces on edges can also be chosen so the edge has a preferred length, pushing its endpoints away when the edge becomes too small.

The forces on a node v can be chosen as follows and weights can be assigned to them. A higher repulsion constant results in more wide-spread graphs, while a higher attraction constant results in shorter edges and a more compact graph.

- *Node-node repulsion* For each other node, a force proportional to the inverse square of the distance is applied to push v directly away:

$$F_{\texttt{repulsion}} = \sum_{u \in V \setminus \{v\}} \frac{1}{d(u,v)^2} \cdot \frac{v-u}{|v-u|}$$

- *Edge attraction* For every edge e, connecting v to u, a force proportional to the length of e is applied to attract v to u. Let C_v be the set of all nodes connected to v via an edge. The attraction force then is:

$$F_{\texttt{attraction}} = \sum_{u \in C_v} u - v$$

This algorithm finds a local optimum for the given layout, instead of a global one. As a consequence, the configuration of the graph before the spring embedder is run has great influence on the output. Since the initial placement of the nodes is actually a well thought-out layout, this is a very desirable property.

4.2.4 Post-processing

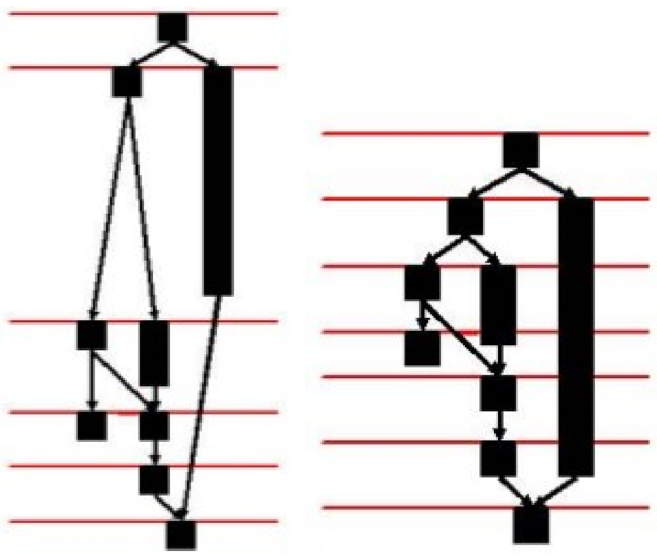
Once the spring embedder has finished, an attempt is made to minimize the number of edge crossings. Each supernode is checked, seeing if its center of mass can be brought closer to the cycle by flipping it on its x and/or y axis.

4.3 Schreiber

Schreiber introduces a method to draw metabolic pathways [18]. It is based upon the layered layout algorithm by Sugiyama et al. [20].

The standard layered layout algorithm accepts a directed acyclic graph and returns a layout where each node is placed in a layer below all its predecessors. This can be seen in Figure 4.2a. Schreiber extends this algorithm to take into account large node sizes, where large nodes can span multiple layers.

In addition to the node sizes, Schreiber introduces constraints on the graph. These constraints are then adhered to by the layout algorithm. Schreiber identifies four constraints which can be enforced: *top-bottom* and *left-right* constraints, which enforce the vertical and horizontal ordering between two nodes; *horizontal* constraints which place two nodes in the same layer; and *vertical* constraints which give two nodes the same *x*-coordinate.



(a) Original layered layout. *(b) Adapted layered layout.*

Figure 4.2: Layered layouts. In (a) we see the layout as proposed by Sugiyama et al. [20] Schreiber [18] has adapted the algorithm, spanning long nodes over multiple levels. This can be seen in (b). Images taken from Schreiber [18].

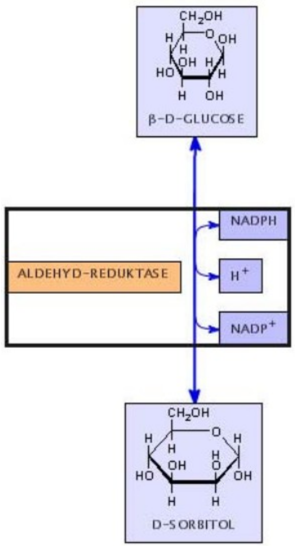


Figure 4.3: Clustered supernode of the enzymes and side compounds of a reaction. Image taken from Schreiber [18].

The main idea is to transform the network into a directed acyclic graph which can be drawn using the layered layout algorithm. To achieve this, the following structures need to be identified and transformed:

1. *Side compounds and enzymes*: Side compounds and enzymes of a reaction are clustered into a supernode. This supernode is then inserted in the middle of the edge representing the reaction. The contents of the supernode are laid out separately, with enzymes being placed to the left of the edge and side compounds to the right. An example of such a supernode can be seen in Figure 4.3.

2. *Closed cycles*: Since the layout algorithm expects an acyclic graph, we need to transform cycles. For each cycle, we reverse an edge. Then, if it is possible to do so without introducing a new cycle, the algorithm rotates the adjacent supernode (containing side compound and enzyme information) by 180° and reverses all other edges adjacent to the supernode. Now, this reaction will be laid out bottom-to-top instead of top-to-bottom. If the supernode cannot be rotated without introducing new cycles, each edge adjacent to the supernode is split by an extra node, allowing additional bends.

3. *Open cycles*: In textbooks, open cycles are normally drawn as spirals. Schreiber argues that in an electronic setting, this is not optimal and proposes a more compact visualization which places each loop of the spiral next to the previous one. See Figure 4.4 for an example. In this case, constraints are added to ensure each loop is vertically aligned, while corresponding compounds between loops are horizontally aligned. Extra nodes are added between each loop, allowing the layout algorithm to route the edges between loops nicely. In Figure 4.4, every bend in an edge was created by adding a temporary node.

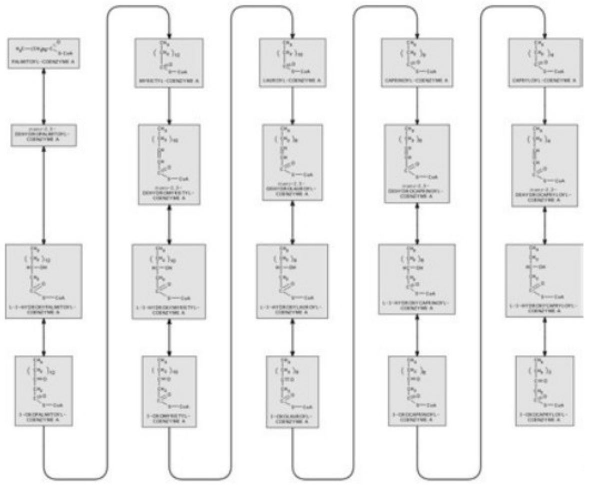


Figure 4.4: An open cycle. Instead of drawing it as a spiral, each consecutive loop is placed to the right of the previous loop, where every loop is vertically aligned and corresponding compounds in each loop are horizontally aligned. Image taken from Schreiber [18].

4.3.1 Algorithm

The main algorithm is given a metabolic pathway, along with an indication of open and closed cycles in that pathway. If the cycles are not given, we can calculate the closed cycles by running a cycle detection algorithm first. The algorithm then produces a layout for this network in the following steps:

1. For each reaction, cluster the side compounds and enzymes into a supernode, and create a local layout for this supernode.
2. Insert layout constraints and temporary nodes to create appealing layouts for open and closed cycles.
3. Handle closed cycles by reversing edges or creating extra nodes.
4. Assign nodes to horizontal layers, with all edges being directed from top to bottom and minimizing the distance between a node and its predecessors. In this way, we assign an y-coordinate to each node.
5. For every edge which spans multiple layers, add a temporary node on each layer.
6. Calculate a permutation of the nodes in each layer that minimizes the number of edge crossings while ensuring that all vertical and left-right constraints are met.
7. Calculate the x-coordinate of each node, maintaining the order in each layer and adhering to the vertical constraints.
8. Decluster the supernodes.
9. Layout all edges by using the temporary nodes as bending points for the edges.

This results in very structured and compact drawings, as can be seen in Figure 4.5.

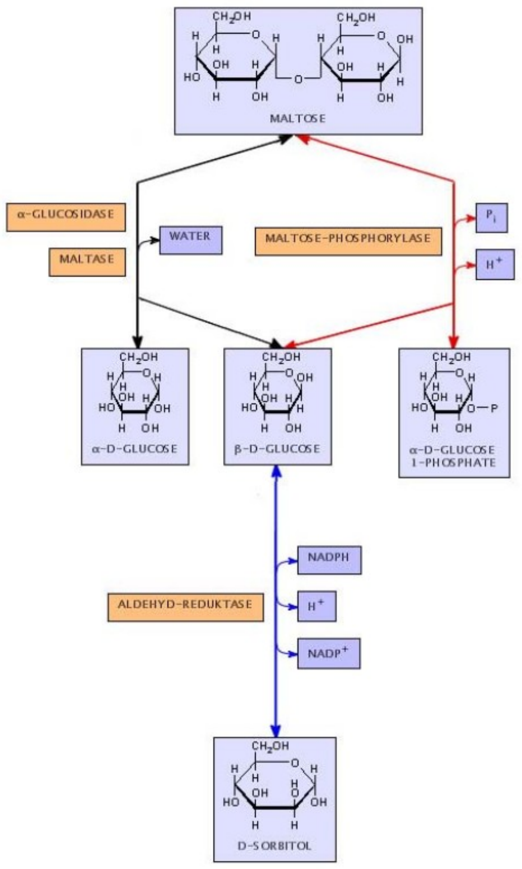


Figure 4.5: A layout of a metabolic pathway, created by Schreiber's algorithm. Image taken from Schreiber [18].

5 Mental Map

Maintaining the mental map when changing a graph allows the user to more quickly recognise the old structures he already studied. We describe two different methods which try to preserve the mental map after a layout change.

The force scan algorithm by Misue et al. strictly maintains the vertical and horizontal ordering of nodes, but does not take into account any of the other criteria detailed in Section 2.2.1. We describe the force scan in Section 5.1.

The simulated annealing approach introduced by Lee et al. [13] can accommodate any criterion, but does not guarantee that a criterion is strictly met: parts of the final layout may violate constraints if this benefits the overall outcome enough. We describe this algorithm in Section 5.2.

5.1 Force Scan

Ideally, when trying to preserve the mental map, we want to retain the old layout as much as possible when inserting new nodes. The force scan algorithm, introduced by Misue et al. [15], accomplishes this by guaranteeing that nodes do not overlap, while horizontal and vertical ordering of nodes is preserved.

First, we create a layout for the original graph. Then, when new nodes are added to this graph, we just place them (according to some layout algorithm). This new placement might result in overlapping nodes, or nodes which are too close to each other. Then, we calculate forces between every pair of nodes. These forces must be chosen in such a way that when they are applied, no nodes will overlap.

Misue et al. [15] prove that the Force Scan algorithm ensures that nodes are not overlapping, while maintaining both horizontal and vertical ordering of the nodes. This is a desirable property, since the spatial ordering of nodes is important in preserving the mental map of a user. However, there is no guarantee that the algorithm maintains clustering in any way.

5.1.1 Forces

For each node we consider its bounding box only. This allows us to use any shape for the nodes. We can then also use the supernodes from Karp's algorithm as input for the force scan. For a given node v we denote the center of its bounding box as p_v.

For every pair of nodes u and v, we take the distance $d_{u,v}$ to be the Euclidean distance between p_u and p_v. The desirable distance $k_{u,v}$ is the distance we would ideally want the pair of nodes to have.

Let r be the first point on the line from p_u to p_v, where when we move v to be centred at r, the nodes do not overlap. The desirable distance is the distance between p_u and r. Thus, $k_{u,v}$ is the minimal distance between p_u and p_v at which u and v do not overlap (when we keep the orientation between u and v constant). Figure 5.1 illustrates this.

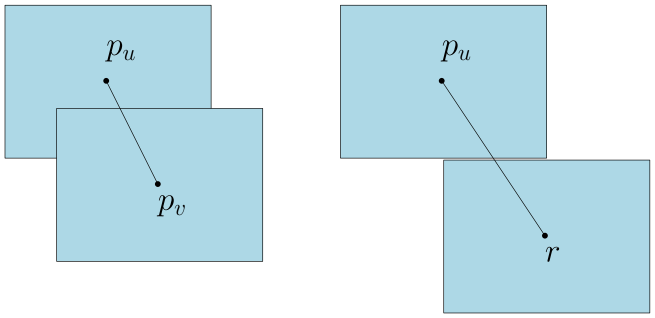


Figure 5.1: The desirable distance between two nodes. Given two (overlapping) nodes, the desirable distance is the distance between the centers at which the nodes do not overlap anymore.

We only want to move nodes away from each other when they overlap, while preserving the orientation between them. If we take $i_{u,v}$ to be the unit vector from u pointing to v, the force $f_{u,v}$ between u and v is:

$$f_{u,v} = \max(0, k_{u,v} - d_{u,v}) i_{u,v}$$

We denote the x and y components of this force by $f^x_{u,v}$ and $f^y_{u,v}$.

If we want to have a minimum distance between two nodes (instead of merely disjoint node images), we can increase $k_{u,v}$ by some constant.

5.1.2 Scan

We can use the resulting forces to push nodes to a new position. These forces are applied in two scans: first in the horizontal direction, then in the vertical direction.

For the horizontal scan, order the nodes by x-coordinate. We scan from left to right. At each step, we select the leftmost node we have not handled yet, v, and find all nodes with the same x coordinate. These nodes form a set S. All nodes strictly to the right of v go in a set R. Then, we find the force $f_{\max}$ from any node in S to any node in R which has the maximal x component. We then apply the x component of $f_{\max}$ to every node in R.

In essence, we search for the maximal displacement S causes in R. Then we apply this displacement to all nodes in R, thus making sure the horizontal ordering is preserved. Algorithm 1 describes this algorithm in pseudocode.

The final Force Scan algorithm then calculates all forces and performs the horizontal scan followed by an analogous vertical scan.

Algorithm 1 Horizontal Scan

```
1:  function HORIZONTALSCAN
2:      Sort all nodes in $V$ by $x$-coordinate
3:      $i \leftarrow 0$
4:      while $i < |V|$ do
5:          $S \leftarrow \{V_i\} \cup \{V_j \mid V_j[x] = V_i[x]\}$
6:          $i \leftarrow i + |S|$
7:          $R \leftarrow \{V_j \mid j \geq i\}$
8:          $f_{\max} \leftarrow \max f^x_{s,r}$ for $s \in S, r \in R$
9:          for $r \in R$ do
10:             $r[x] \leftarrow r[x] + f_{\max}$
```

5.2 Simulated Annealing

Simulated annealing is a generic optimization technique independently introduced by both Černý [5] and Kirkpatrick et al. [11]. It tries to find the optimal solution for a given cost function. Lee et al. [13] use simulated annealing to maintain the mental map. They define multiple costs which model changing the mental map.

In this section, we will first explain the generic simulated annealing technique. Then, we will describe the cost function as given by Lee et al.

5.2.1 Algorithm

Simulated annealing is based upon the physical process where liquids or metals cool down and settle into a stable state. At first, when the temperature is high, the system is unordered and state varies wildly. As the system cools down, it becomes more stable and ordered.

The algorithm starts with a given configuration of the system δ and a temperature T. Then a new state δ' is selected at random from the neighbourhood of δ. This state is always accepted if it has a lower cost. If the new cost is higher, we accept it depending on T and the costs of δ and δ'. After the step (whether we accepted δ' or not), we lower T according to a cooling schedule.

The higher the temperature, the larger the chance that δ' is accepted even if it has a higher cost. This means that when the algorithm starts, it can more easily circumvent local optima. When the algorithm is nearly finished, it essentially becomes a standard hill-climbing algorithm, finding a local optimum.

Algorithm 2 shows these steps formally. Here, we assume the presence of a cost function `E`, the acceptance probability `P` and a maximum number of iterations i_{max}. We also assume a cooling schedule `temp`, a function which takes a fraction of the remaining iterations and returns a temperature. The larger the fraction, the lower the temperature. Lastly, we can choose a random neighbour of a state by calling `neighbour`.

The acceptance probability `P` is often defined as $e^{-(c'-c)/T}$. It was introduced by Kirkpatrick et al. [11] and justified because of the analogy with the physical annealing process.

The `neighbour` function generates a random state in the neighbourhood of δ. Lee et al. propose to generate a neighbour by picking a random node and assigning it a new, random position.

Algorithm 2 Simulated Annealing

```
1: function SIMULATED ANNEALING
2:     $\delta \leftarrow \delta_0$, $c \leftarrow \mathtt{E}(\delta)$
3:     $i \leftarrow 0$
4:     while $i < i_{max}$ do
5:         $T \leftarrow \mathtt{temp}\left(\frac{i}{i_{max}}\right)$                              ▷ Temperature
6:         $\delta' \leftarrow \mathtt{neighbour}(\delta)$                    ▷ Neighbouring state and energy
7:         $c' \leftarrow E(\delta')$
8:         if $c > c'$ or $\mathtt{P}(c, c', T) > \mathtt{random}()$ then
9:             $\delta \leftarrow \delta'$            ▷ Accept $\delta'$ if it has lower cost, or depending on P
10:        $i \leftarrow i + 1$
```

The initial temperature T is often chosen to be very high, allowing the algorithm to move to any configuration. However, given an initial layout which resembles the desired result, T can be chosen to be lower, so the algorithm is less likely to make uphill moves.

The cooling schedule Lee et al. use is a geometric one. After thirty moves at the current temperature T, the new temperature T' is set to γT, where γ is typically chosen to be in between 0.6 and 0.95.

5.2.2 Cost Function

There are multiple criteria which determine if the mental map has been maintained, or whether a graph was drawn nicely. Since Lee et al. [13] use simulated annealing for both purposes, they have defined costs to reflect these criteria. Each cost i is given a weight λ_i. The total cost then becomes $\sum \lambda_i \cdot \mathtt{cost}_i$.

Lee et al. separate the costs associated with creating a nice layout and maintaining the mental map. We will describe both.

Layout costs

Lee et al. identify five costs associated with a good layout:

1. *Node distribution*: Nodes should be distributed evenly across the available drawing space. For each pair of nodes i and j, $\frac{1}{d_{i,j}^2}$ is added to the cost, where d is the euclidean distance. The closer nodes are together, the higher the cost becomes.

2. *Borderlines*: Lee et al. prefer nodes to stay within a certain area. Using node distribution alone, the graph would expand indefinitely. The borderlines cost prevents nodes from reaching the borders. For each node i, $\frac{1}{r_i^2} + \frac{1}{l_i^2} + \frac{1}{t_i^2} + \frac{1}{b_i^2}$ is added to the cost. Here, r_i, l_i, t_i, b_i are the distance between node i and the right, left, top and bottom borders.

3. *Edge lengths*: The cost for edge lengths is not determined by their actual length, but rather by how much the edge lengths differ. Therefore, the edge length cost is s^2, where s is the standard

deviation of the edge lengths.

4. *Edges crossings*: To reduce the number of edge crossings, the number of edge crossings are counted. First, a weight e_i is assigned to every edge i. Then, if two edges i and j cross, $e_i \cdot e_j$ is added to the cost.

5. *Node-edge distances*: The node distribution and edge crossing costs try to distance nodes from nodes and edges from edges to improve readability. Edges which are very close to nodes they are not connected to decrease readability. Thus, for each edge e and each node v which is not connected to e, the distance $d_{e,v}$ is calculated. Then, $\frac{1}{d_{e,v}^2}$ is added to the cost.

Mental Map Costs

Lee et al. identify six costs associated with maintaining the mental map. They base these costs on criteria defined by Bridgeman et al. [3]. Section 2.2.1 describes these criteria in more detail.

The costs are based upon the difference between the original layout and the new layout. The original point set and graph layout are denoted by P and D, respectively. Their counterparts after adjusting the layout are P' and D'. The counterpart of p in the new layout is denoted by p'.

The costs are then as follows:

1. *Rankings*: This cost deals with the horizontal and vertical ordering of nodes. Let $\texttt{right}(p)$ and $\texttt{above}(p)$ denote the number of nodes to the right and above p. Let p' be the point corresponding to p in the new layout. The ranking cost then becomes:

$$rank(P,P') = \sum_{p \in P} |\texttt{right}(p) - \texttt{right}(p')| + |\texttt{above}(p) - \texttt{above}(p')|$$

   Lee et al. also give a weight to be used for this cost: $\lambda_{rank} = 1.5(|P|-1)$, normalizing the rank cost with respect to the number of nodes.

2. *Average relative distance*: The distance between pairs of nodes should not change too much between layouts. This is captured by the average relative distance cost:

$$rdist(P,P') = \frac{1}{|P|\,(|P|-1)} \sum_{p,q \in P} \left| d(p,q) - d(p',q') \right|$$

   If the distance between a pair of nodes changes, this cost becomes higher.

3. *Shape*: Edge orientation should be preserved between layouts. Given an edge e, its orientation $Or(e)$ is a discrete representation of the angle the edge makes. Lee et al. [13] confine the orientation to one of the eight cardinal and intermediate directions (N, W, S, E, NW, NE, SW, SE). The more edges change their orientation, the more the shape of the layout is altered. The associated cost becomes:

$$shape(P,P') = \frac{1}{|E|} \sum_{e \in E} \texttt{edited}(e,e')$$

With $edited(e,e') = \begin{cases} 1 & \text{if } Or(e) \neq Or(e') \\ 0 & \text{otherwise} \end{cases}$

In essence, we count the number of edges with changed orientation and normalize by the number of edges in the graph.

4. λ-*matrix*: P and P' are said to have the same order type if every triple of points (p,q,r) is oriented clockwise if and only if (p',q',r') is also oriented clockwise. Lee et al. use a simplified version, where the number of nodes left of a pair (p,q) is calculated.

   Let $\lambda(p,q)$ be the number of nodes to the left of the line through p and q. The cost is then defined as:

$$\lambda(P,P') = \frac{1}{norm} \sum_{p,q \in P} \left| \lambda(p,q) - \lambda(p',q') \right|$$

   Where $norm = n \left\lfloor \frac{(n-1)^2}{2} \right\rfloor$

5. *Nearest neighbour within*: After the layout has been altered, the nearest neighbour of a node should not change. Let $nn(p)$ be the nearest neighbour of node p (making $nn(p)'$ the corresponding node of $nn(p)$ in P'). Let $nearer(p)$ be the amount of nodes closer to p' than the original nearest neighbour of p:

$$nearer(p) = \left| \{ q \in P \mid d(p',q') < d(p',nn(p)'),\ q \neq p,\ q \neq nn(p) \} \right|$$

   The nearest neighbour within cost then counts the amount of nodes which have been placed closer to a node than its nearest neighbour, and normalizes that amount:

$$nnw(P,P') = \frac{1}{|P|} \sum_{p \in P} nearer(p)$$

6. *Nearest neighbour between*: Given a node at point p in the original layout and at p' in the new layout, it is ideal if there are no other nodes closer to p than p' is. This allows the user to glance at the old position of a node and quickly find the node he was looking for.

   Let $nearer'(p)$ be the amount of nodes closer to p than p' is:

$$nearer'(p) = \left| \{ q \in P \mid d(p,q') < d(p,p'),\ q \neq p \} \right|$$

   Like the nearest neighbour within cost, we count the amount of nodes which violate this constraint and normalize the result:

$$nnb(P,P') = \frac{1}{|P|\,(|P|-1)} \sum_{p \in P} nearer'(p)$$

These six costs combined will let the simulated annealing algorithm balance the different criteria on maintaining the mental map. Simulated annealing does not guarantee that any of these criteria are met, only that the total cost is as low as possible. Thus, some constraints may be violated if sufficient gain is obtained in other criteria. This differs very much from the force scan algorithm by Misue et al. [15] which enforces that orthogonal ordering criterion but lacks all others.

6 Dynamic Metabolic Pathway Layout

In this chapter we will introduce our approach for combining static pathway layout and dynamic changes while preserving the mental map. We will first discuss some design choices we have made, then we describe which methods we have used and any alterations made to those methods. We will also mention some other possibilities to maintain the mental map.

6.1 Design choices

6.1.1 Data source

We have chosen to use the MetaCyc knowledge base [4] as the basis for our application.

Our pathway data structure closely resembles the one used by MetaCyc, where a pathway is formed by a predecessor list of reactions. This means that for every reaction, we know which reactions directly precede it. Given a set of reactions which belong to a pathway, we can then calculate which reactions connect at which compounds, resulting in a graph representation of the pathway.

We have imported all compounds, reactions and pathways from EcoCyc [10], the E. Coli database provided by MetaCyc.

6.1.2 Graph Format

We choose to not use hypergraphs, since some of the layout algorithms expect standard graphs as input.

Since each reaction in a pathway can have multiple reactants, products and enzymes, the edges in a metabolic pathway are hyperedges. We will deal with these by adding one or two virtual nodes to each hyperedge, creating multiple edges with only two endpoints.

6.1.3 Graph Library

We use JUNG (`http://jung.sourceforge.net/`) as a basis for our visualization system. It provides user interaction on the graph, a very extensible graph model and a modifiable rendering system.

It also provides some standard layout algorithms. However, these algorithms assume nodes to be points. This means that the algorithms are not suitable for our purposes, since they will not prevent overlap between nodes.

6.1.4 Main Compounds and Side Compounds

We identify main compounds and side compounds in the pathway. Each pathway has some compounds which form the main reaction; other compounds are merely present as an energy source (e.g. ATP and ADP), or supply some basic building blocks (e.g. sodium or hydrogen). The former compounds are main compounds, the latter compounds are side compounds. In textbook examples main and side compounds can be identified by their position: main compounds are connected to multiple other compounds or lie directly in line with other main compounds. Side compounds are placed next to the reactions they are a part of, and are often connected to other compounds with curved edges instead of straight ones. Most importantly, side compounds are never incident to more than one edge. Multiple occurrences of the same side compound result in duplicated nodes. See Section 2.1.1 for more details.

We identify side compounds heuristically. Basically, a compound is a side compound if it frequently occurs in reactions and does not play an important role in the current pathway. The first criterion makes sure that only compounds which are often used are considered as side compounds. The second states that even if a compound is often used, it can still be a main compound. For example, if ATP occurs in reactions, it is often used as an energy source, resulting in a waste product of ADP. However, if we consider a pathway which converts energy and ADP into ATP; we cannot consider ADP or ATP side compounds.

We label a compound as a side compound if and only if it has only in-edges or only out-edges, and it is involved in enough reactions. In our implementation, we have chosen ’enough’ to mean more than 2% of all reactions in the database. This is the percentage which we deemed a reasonable threshold for the data we obtained from MetaCyc.

Side compounds are always accompanied by a main compound, which is on the same side of the reaction as the side compound. Thus, side compounds are always part of a hyperedge we have split, connecting to a virtual node. This virtual node in turn connects to at least one main compound on the same side of the reaction. We will always try to place side compounds parallel to the edge between the virtual node and the accompanying main compound.

This also implies that side compounds and enzyme labels are never drawn parallel to the same edge: if a reaction has side compounds, a virtual node is always inserted between the enzyme label and the part of the edge where the side compound is parallel to. Figure 6.1 shows this. Here, the diamond nodes represent virtual nodes inserted into an edge.

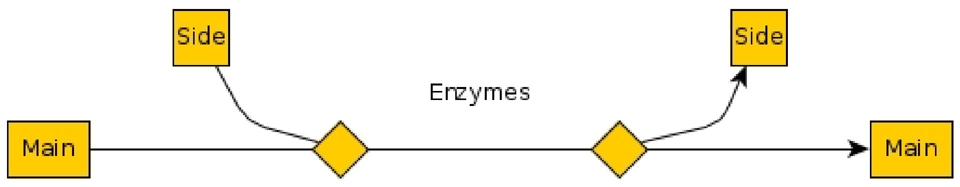


Figure 6.1: A model of reaction with side compounds and enzymes. Enzymes are never parallel to the same edge as side compounds are.

6.2 Layout

Our algorithm essentially consists of two parts. The first part deals with creating a static layout which is as nice as possible. This applies for example when a user examines a new pathway, or when a combined pathway - drawn with a focus on maintaining the mental map - is redrawn with a focus on clear layout. The second part deals with additions to the current pathway, in which case we would like to preserve the mental map.

6.2.1 Static Layout

Our algorithm is based on the method by Karp and Paley (see Section 4.1), along with the extensions as defined by Becker and Rojas (see Section 4.2). We will detail the changes we have made to the algorithm.

Linear pathway layout: Karp and Paley describe a snake layout for linear pathways, which is useful when the final layout must be kept within a certain area. Our approach assumes there are no such size constraints (because the user can pan the graph), thus we can draw linear pathways on a straight line.

Side compound placement: Side compounds are placed as Karp suggests (i.e. to the right or above edges, or on the outside of any cycle the compound belongs to), but we let JUNG decide where the edge labels should be. JUNG always places edge labels horizontally, above the middle of the edge. In the case of horizontal edges this corresponds to Karp's placement. In the case of vertical edges, the edge label intersects the edge itself. However, since the text itself is horizontal, we think this improves readability.

Since any enzymes will be placed along an edge without any side compounds (see Section 6.1.4), the enzyme text will never overlap any side compounds.

Inner and outer supernodes: For each component connected to the main cycle, we must decide whether to place it inside or outside the main cycle. Becker and Rojas only place single nodes with multiple connections to the main cycle on the inside. Karp and Paley place all nodes with multiple connections on the inside.

Placing nodes with multiple connections on the outside of the cycle possibly leads to more unnecessary edge crossing, since nodes placed within the cycle can be connected to different cycle nodes without needing to intersect the cycle. However, placing large supernodes inside the cycle requires us to give the cycle a very large diameter. Presumably, there is an optimal maximum size of the supernodes we can place inside the cycle. Finding this optimum is not trivial. We have elected to use the choice Becker and Rojas made, to avoid intersecting the cycle.

Supernode placement: After the main cycle has been laid out and all supernodes have had their internal layouts created, the supernodes are positioned alongside the cycle. Karp and Paley suggest using a tree layout algorithm to accomplish this, while Becker and Rojas place the nodes according to their preferred orientations. Like Becker and Rojas, we rotate the main cycle such that the largest supernode can be placed according to its preferred orientation.

However, instead of placing the other supernodes according to their preferred orientation, we place them according to the cycle node the supernode is connected to. Since the supernode is located on the outside of the cycle, it is connected to exactly one node v in the cycle. We place the supernode on the line from the center of the cycle through v. We do this so we the edge from v to the supernode does not intersect the cycle. In the post-processing step we can still flip the supernode to minimize edge crossings within the node itself.

Spring layout: Becker and Rojas use a spring embedder which acts on all supernodes and on all nodes of the cycle. When running the spring embedder, we lock the cycle nodes in place, ensuring that the cycle maintains its circular layout. This may result in overlap between nodes, since the cycle may not be large enough to accommodate all supernodes. To remove this overlap, we run the force scan algorithm we also use during dynamic layout.

6.2.2 Dynamic Layout

The dynamic part of the layout process comes into play when a pathway is added to a pathway which has already been laid out. The process consists of two parts: first we determine the position of newly added nodes. Then, we remove any overlap between nodes by performing a force scan (see Section 5.1). Finally, we perform the post-processing step by Becker and Rojas.

Positioning New Nodes

As in the static layout, each connected component in the set of newly added nodes becomes a supernode. We create a layout for each supernode using the static method. Then, we assign positions to each new supernode as follows:

- A supernode which is connected to exactly one node of the main cycle will be placed as in the static layout.
- A supernode which is connected to exactly one supernode will be placed according to its preferred orientation relative to its neighbour.
- A supernode which is connected to multiple nodes will be placed at the average position of those nodes. This means that if a node is connected to multiple nodes of the main cycle (but not connected to other supernodes), it will be positioned somewhere inside the cycle.

After all positions have been assigned, we run the spring embedder, with all old nodes locked in place. Since the old layout does not change in this step, we have not affected the mental map of the user.

Running the spring embedder can improve the layout, but because we have locked the entire old layout in place, the spring embedder may run into problems. We will go into more detail below.

Removing Overlap and Post-Processing

When the new nodes have been positioned, we may end up with a graph with overlapping nodes. Since we want to preserve the mental map, we must take care when removing the overlap between nodes. We do so by performing a force scan on the graph. This force scan will act on the supernodes and the nodes in the main cycle, keeping the layout inside the supernodes intact.

The force scan ensures that there is no overlap between nodes. It also maintains the orthogonal ordering of the nodes.

All nodes now have their final positions. We now perform the post-processing step Becker and Rojas use to minimize edge crossings: for each supernode, we check if flipping the node on its x or y axis would bring the center of mass closer to its neighbouring node. If so, we flip the node.

However, supernodes can now be connected to more than one node. For a given flip, let C be the set of neighbours which are closer to the center of mass after the flip. Let F be the set of neighbours which are further away from the center of mass after the flip. We perform the flip if $|C| > |F|$, i.e. if more nodes are closer to the center of mass after the flip than there are nodes which are further away, we flip.

Spring Embedder Details

Running the spring embedder to place newly added nodes results in poor layouts. The initial idea was that the spring embedder would place the new nodes roughly where they should be, while the force scan algorithm would move the nodes when necessary. In practice, the spring embedder ensures that there is no overlap between nodes. Even if we place the larger supernodes on the inside of a cycle, they are just pushed out if there is not enough space. In Figure 7.5 we can see how no changes are made to the original layout when adding a new pathway. We can see two solutions to this problem.

One way to solve the problems caused by the spring embedder is to remove it altogether. We argued that the initial placement of the supernodes (along with the flipping) as described in Section 6.2.1 results in few edge crossings. We can then let the force scan algorithm take care of overlapping nodes, while maintaining the mental map.

When we do use the spring embedder, it can result in convoluted layouts, since it does not take edge crossings or edge-node overlap into account. Edge crossings are hard to solve using a spring embedder since moving nodes slightly will not improve the result: we need to move nodes in such a way that the edge crossing is removed entirely. In other words, there is no gradient in the cost function for the algorithm to follow. Edge-node overlap can be solved by the spring embedder by introducing a new force which repels nodes from edges.

When using this last option, we can choose not to lock the old nodes into place. This allows new nodes to push the old layout aside to remove overlap. However, we would like the spring embedder to affect the new nodes the most. Thus, we assign weights to all nodes: old nodes are heavier and are less affected by the forces of the spring embedder. In addition, we do not want the original layout to affect itself, thus we remove any forces only acting on already laid out nodes. This method focusses on minimizing the displacement of nodes (as described in Section 2.2.1).

6.3 Other Possibilities

We have combined the methods by Karp [9] and Becker [2] with the force scan algorithm by Lee et al. [13] In this section we will briefly explore two different avenues to pursue, which provide different possibilities to maintain the mental map.

6.3.1 Schreiber

The method Schreiber [18] developed also lends itself to mental map preservation while combining pathways. Section 4.3 gives more detail about his algorithm.

Schreiber briefly mentions mental maps in the step-by-step explanation of the algorithm: as a last step, it is possible to add constraints to maintain the mental map when adding more pathways. However, he does not elaborate on this. The four constraints Schreiber describes deal with relative positions of the nodes: either two nodes must have the same x or y-coordinate (horizontal and vertical constraints), or a node must be above or to the right of another node (top-bottom and left-right constraints).

We can calculate which constraints we need when we have just created a layout. For every pair of nodes (u, v) we check the following and add appropriate constraints:

- If u and v are in the same layer, add a horizontal constraint between them.
- If u and v share the same x-coordinate, add a vertical constraint between them.
- If u is to the left of v, add a left-right constraint.
- If u is in a higher layer then v, add a top-bottom constraint.

Since these constraints are transitive, not every pair of nodes needs to be explicitly assigned these constraints. Essentially, for a layer with m nodes, we need $m-1$ left-right and horizontal constraints. We also need to add m top-bottom constraints, ensuring that all nodes in this layer are always below the first node in the layer right above the current one. Lastly, we need at most $|V|$ vertical constraints to align all nodes with the same x coordinates. This results in a total of $O(|V|)$ constraints, instead of the naive $O\left(|V|^2\right)$.

We must also store which edges we have reversed when removing cycles, so we can reverse the same edges later. Allowing different edges to be reversed will result in the algorithm trying to place the cycles differently, but this conflicts with some of the newly added top-bottom constraints.

Adding another pathway to the layout then results in the algorithm adding extra layers and extra vertical space to accommodate new nodes, while keeping the old layout mostly intact. The orthogonal ordering between nodes is conserved after a layout change - every pair of nodes has constraints which enforce this. Other criteria, like the distribution or displacement of nodes between layouts, are not considered.

6.3.2 Simulated Annealing

The simulated annealing technique described in Section 5.2 inherently allows us to assign different weights to a good layout and preserving the mental map.

The initial layout can be performed by any algorithm. If we use simulated annealing for this purpose, we can ignore any costs associated with preserving the mental map. Based on the findings of Purchase et al. [17] (see also Section 2.2.2), we can assign a low (or zero) weight to the good layout costs and a high weight to maintaining the mental map.

When combining simulated annealing with Schreiber's algorithm, we can add the constraints from Schreiber to the cost of maintaining the mental map. We can then experiment with finding the balance between good layout, maintaining the mental map and satisfying the constraints.

This technique allows us to enforce arbitrary constraints and easily change the balance between different costs. However, the results are dependent upon this balance and finding the right weights may take much experimentation.

7 Results and Future Work

We have built a prototype application to test the dynamic layout algorithm. In this chapter, we will discuss the layouts created by our algorithm, as well as the preservation of the mental map when exploring pathways.

We focussed on creating the dynamic layouts properly, ignoring other aesthetic aspects. As a result, the images generated by our prototype are very unpolished. Most of these issues can be addressed by tweaking the rendering mechanism in JUNG. In particular, virtual nodes inserted to break up hyperedges can result in sharp angles in edges. This can be solved by rendering the edges as splines instead of straight lines.

7.1 Static Layouts

Our static layout algorithm generates readable layouts, where cycles, branches and linear components are easily identified. However, the tree layout algorithm handles back edges poorly, which can result in unclear drawings.

Our algorithm which combines reactions into pathways based upon a predecessor list can use some work. It will always connect reactions when they are connected, but sometimes it will incorrectly connect two side compound nodes where they should have been split. Here we can recognize two different cases.

In the first case, a side compound and a main compound are produced in a reaction, then in the next reaction, the same compounds are also used as substrates. In this case, our implementation connects both the main and the side compounds. In Figure 2.1 we can see this as well: when citrate is converted to isocitrate, water is both produced and used. It is shown only once.

In the second case, a side compound is used in the first reaction as a substrate, then in the following reaction it is produced as a compound. Again, our algorithm combines these side compounds, creating a cycle. While the combining of this compound is not *wrong* in the metabolic sense, it does convolute the drawing needlessly and focusses attention on cycles where there should be none.

We can see both cases in Figure 7.1, where the cycle contains both H_2O and H^+.

Otherwise, side compound detection works fine: branched pathways which include often-used compounds only mark compounds as side compounds whenever they are not essential to the reaction or pathway. We conclude that the 2% threshold we introduced in Section 6.1.4 is reasonable for our current database.

Our implementation of the static layout is adequate but not very polished: we can still improve upon

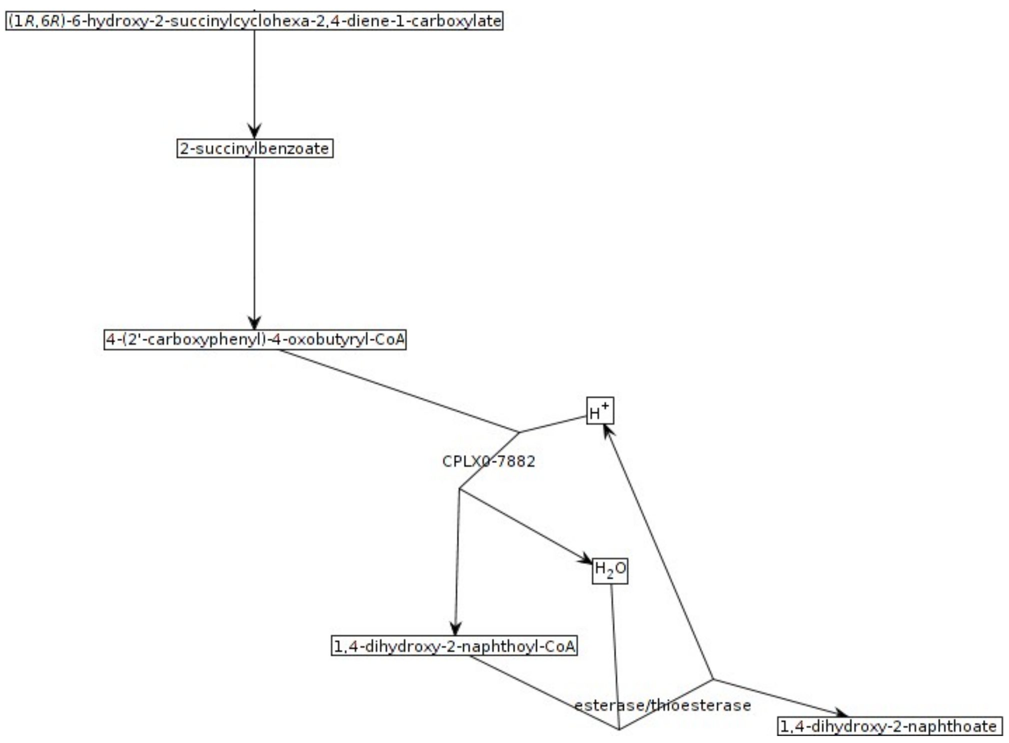

(1R,6R)-6-hydroxy-2-succinylcyclohexa-2,4-diene-1-carboxylate
2-succinylbenzoate
4-(2'-carboxyphenyl)-4-oxobutyryl-CoA
H^+
CPLX0-7882
H_2O
1,4-dihydroxy-2-naphthoyl-CoA
esterase/thioesterase
1,4-dihydroxy-2-naphthoate

Figure 7.1: Layout of naphthoate biosynthesis. Both the H^+ and H_2O side compounds have been connected, introducing a cycle to this pathway.

connecting side compounds and the crossing edges in branched layouts. We can also improve upon the overall appearance of the prototype, as mentioned before.

Figure 7.2 shows the placement of side compounds in a complex pathway.

7.2 Mental Map Preservation

We have examined several variations on our algorithm to maintain the mental map, in particular we have changed the settings concerning the spring embedder. We will illustrate how these settings affect the final layout. We will not show the placement of side compounds since this distracts from the main structure of the graphs.

We have succeeded in creating an algorithm which preserves the mental map when adding pathways to an existing metabolic network. We feel that the variant in which the spring embedder uses edge-node repulsion forces and weighted nodes gives the best results.

In Figure 7.3 we can see the static layout to which we add a reaction. The new layout can then be computed in several ways. If we do not use the spring embedder, but rely solely upon the force scan algorithm to remove overlap, we push old nodes away. This can be seen in Figure 7.4. The layout algorithm has no regard for node-edge distances, which can be seen in both trimethylamine compounds. The structure of the old graph is preserved, but the cycle has been broken up.

If we do use the spring embedder and lock the old layout in place, the mental map is preserved better

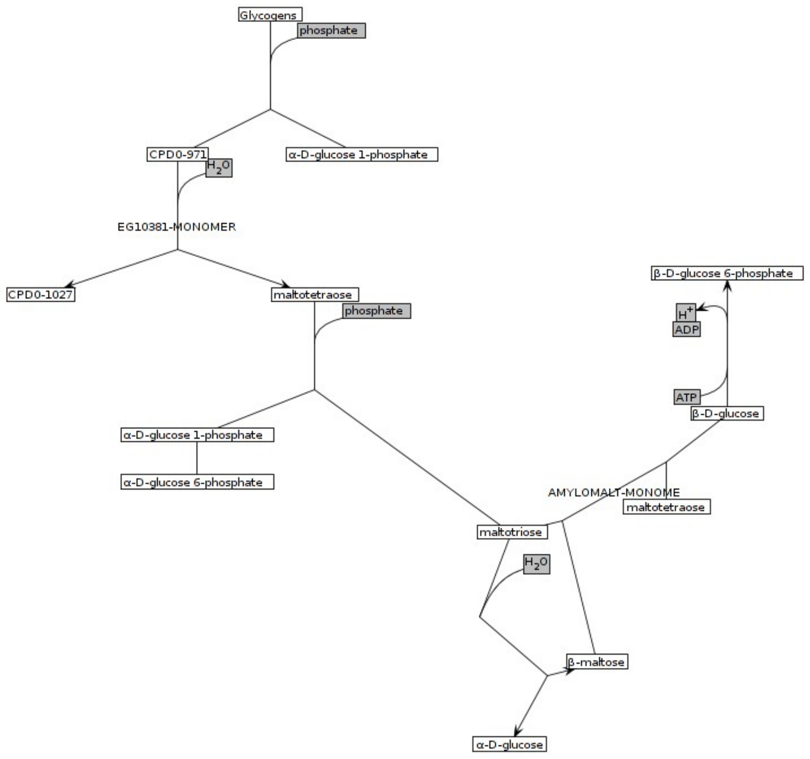


Figure 7.2: Layout which shows placement of side compounds.

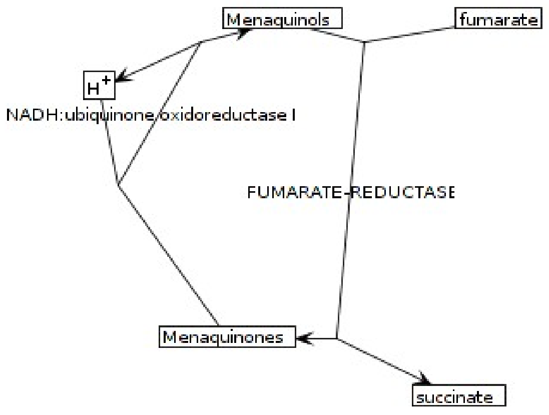


Figure 7.3: Original pathway.

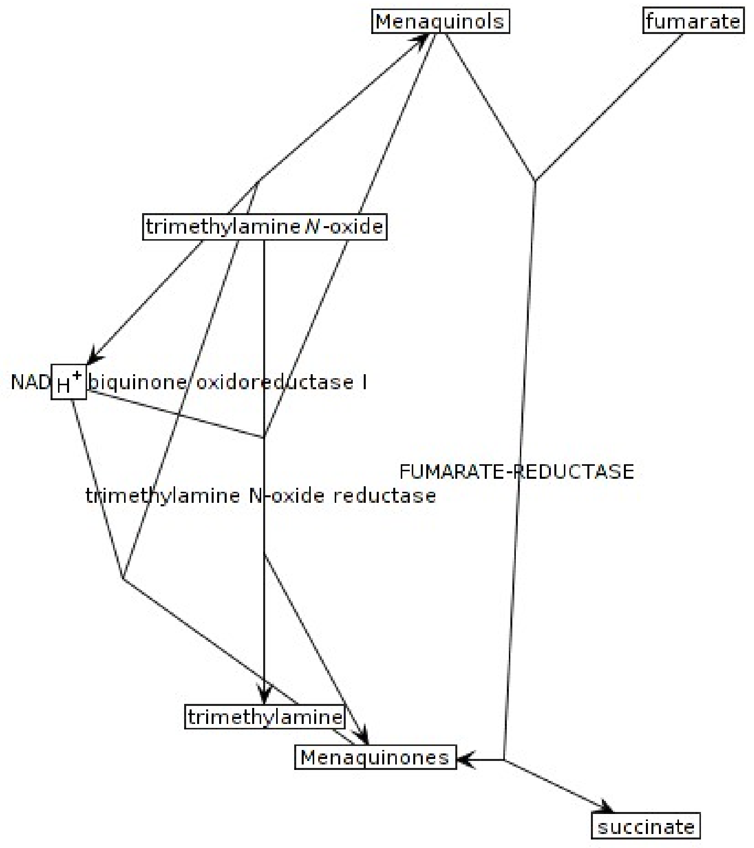


Figure 7.4: Dynamic layout, not using the spring embedder but relying on the force scan algorithm alone.

but we still have node-edge crossings. This can be seen in Figure 7.5.

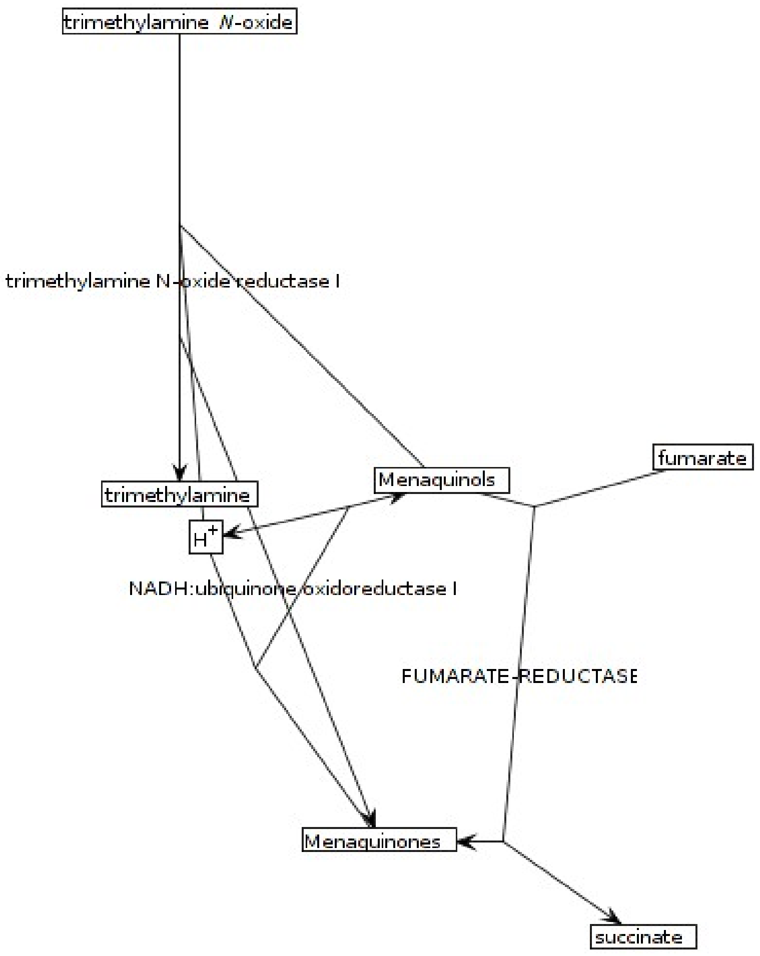


Figure 7.5: Dynamic layout, using a locked old layout and no edge-node repulsion for the spring embedder.

We can try to remedy the node-edge crossings by adding repulsive forces between edges and nodes. The nodes are now clearly separated from the edges, while still maintaining the mental map. We can see this in Figure 7.6.

If we do enable the spring embedder to move nodes in the original layout, we can assign a weight to all old nodes. The results then vary according to this weight. A weight of 10 results in a cycle which is expanded, leaving much room between new and old nodes. When we increase the old node weight to 100, the final layout resembles initial layout much better, while old nodes are still moved to make room for the new reaction.

When we let the spring embedder adjust the original layout and introduce node-edge repulsion forces, we obtain the results as seen in Figure 7.8. Ironically, trimethylamine does overlap edges here; we believe this is a result of the initial placement of this node. Moving the node more to the left or right in-

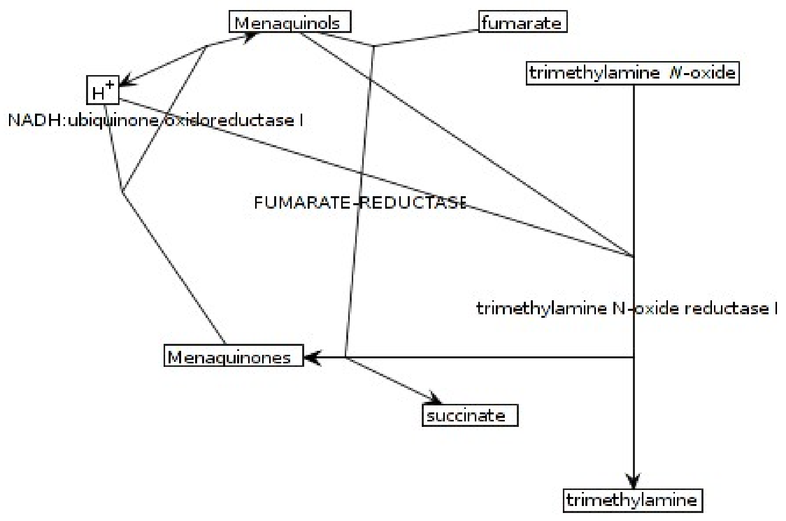
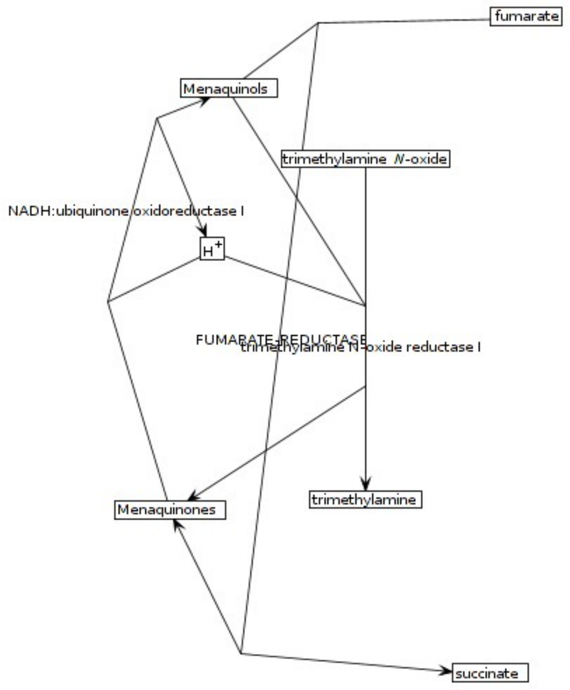
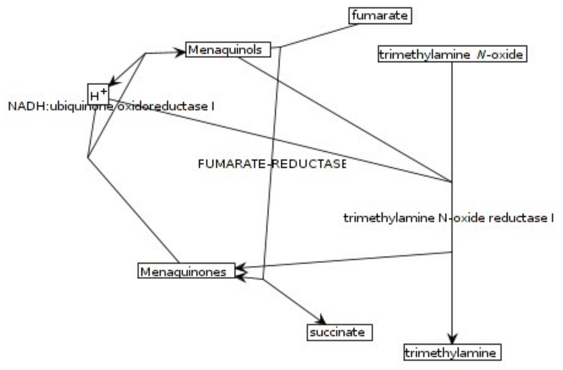


Figure 7.6: Dynamic layout, using a locked old layout and edge-node repulsion for the spring embedder.



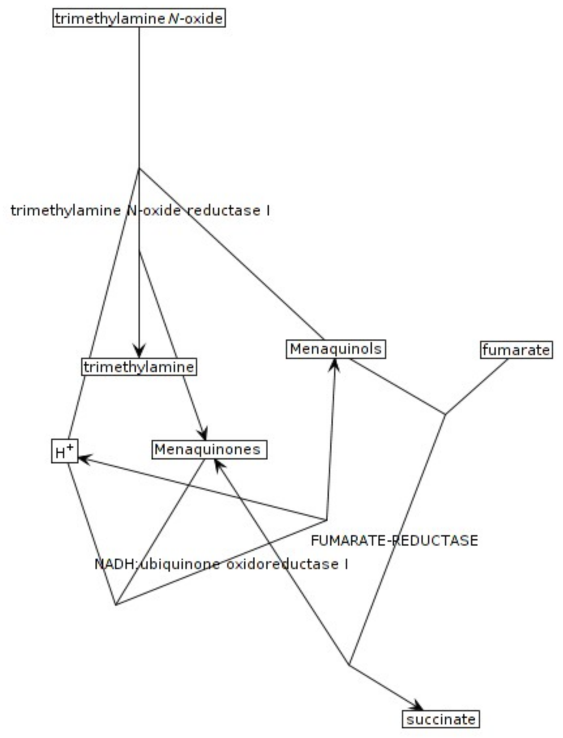
(a) Old nodes have weight 10.



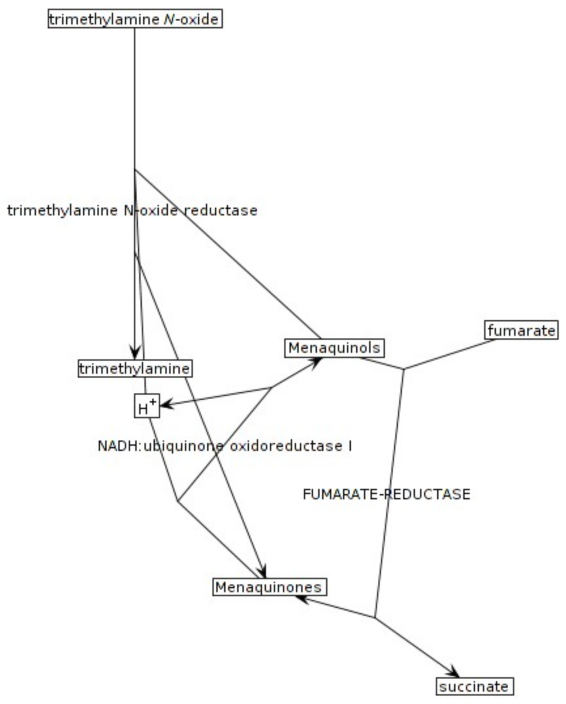
(b) Old nodes have weight 100.

Figure 7.7: Dynamic layout, using heavy old layout nodes and no edge-node repulsion for the spring embedder.

creases the forces of the H^+ and virtual nodes on the edge between trimethylamine and menaquinones. This result very much resembles the one we obtained by running the spring embedder with a locked old layout (Figure 7.5).



(a) Old nodes have weight 10.



(b) Old nodes have weight 100.

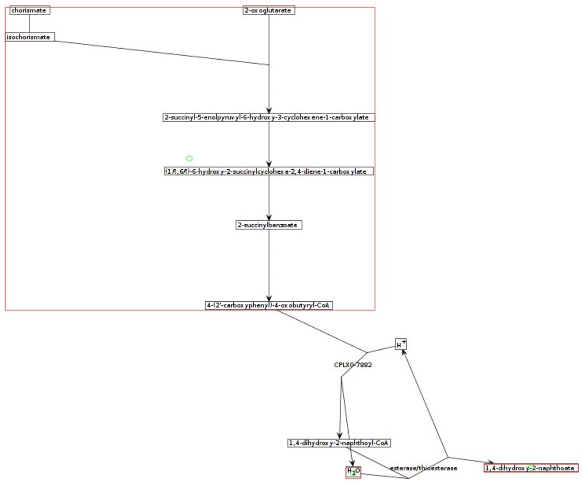
Figure 7.8: Dynamic layout, using heavy old layout nodes and edge-node repulsion for the spring embedder.

7.3 An Example Sequence

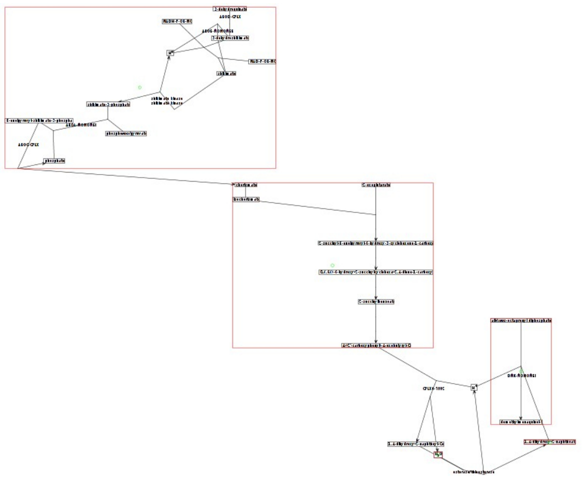
As an example of how our algorithm works when combining multiple pathways with each other, we have constructed the sequence as can be seen in Figure 7.9. This sequence has been created by running our algorithm with the spring embedder enabled, including edge-node repulsion forces. We have assigned a weight of 5 to the old nodes, giving the spring embedder the opportunity to move old nodes to make room for newly added nodes.

We can see that the old layout can be changed somewhat when adding a new pathway, but from one picture to the next we maintain the mental map. When we reset the layout of the pathway after different steps, we obtain the results in Figure 7.10. It is clear that the structure of these graphs is completely different and the mental map is in no way maintained.

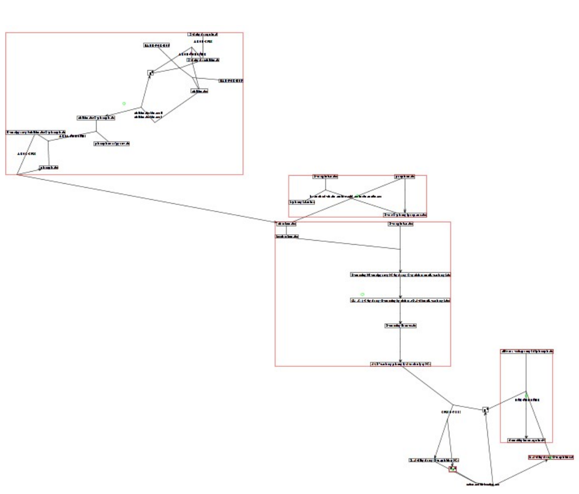
We feel that in this sequence the user has a clear understanding of which parts are affected when adding a new pathway to the current network. Furthermore, this sequence was created interactively:


chorismate
2-oxoglutarate
isochorismate
2-succinyl-5-enolpyruvyl-6-hydroxy-3-cyclohexene-1-carboxylate
(1R,6R)-6-hydroxy-2-succinylcyclohexa-2,4-diene-1-carboxylate
2-succinylbenzoate
4-(2'-carboxyphenyl)-4-oxobutyryl-CoA
H+
CPLX0-7882
1,4-dihydroxy-2-naphthoyl-CoA
H2O
esterase/thioesterase
1,4-dihydroxy-2-naphthoate

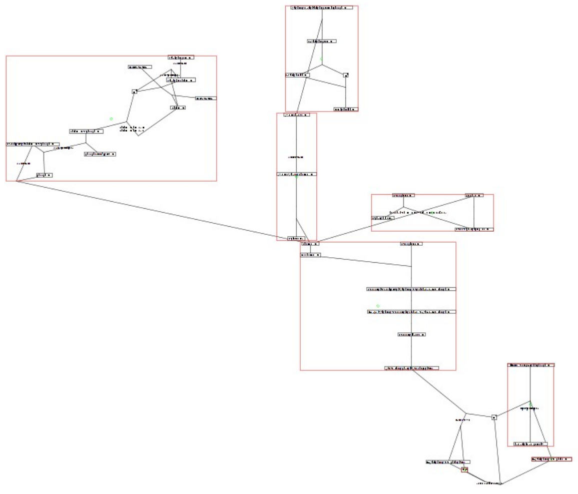
(a) Initial layout.



(b) Added two pathways.



(c) Added one more pathway



(d) Added two more pathways.

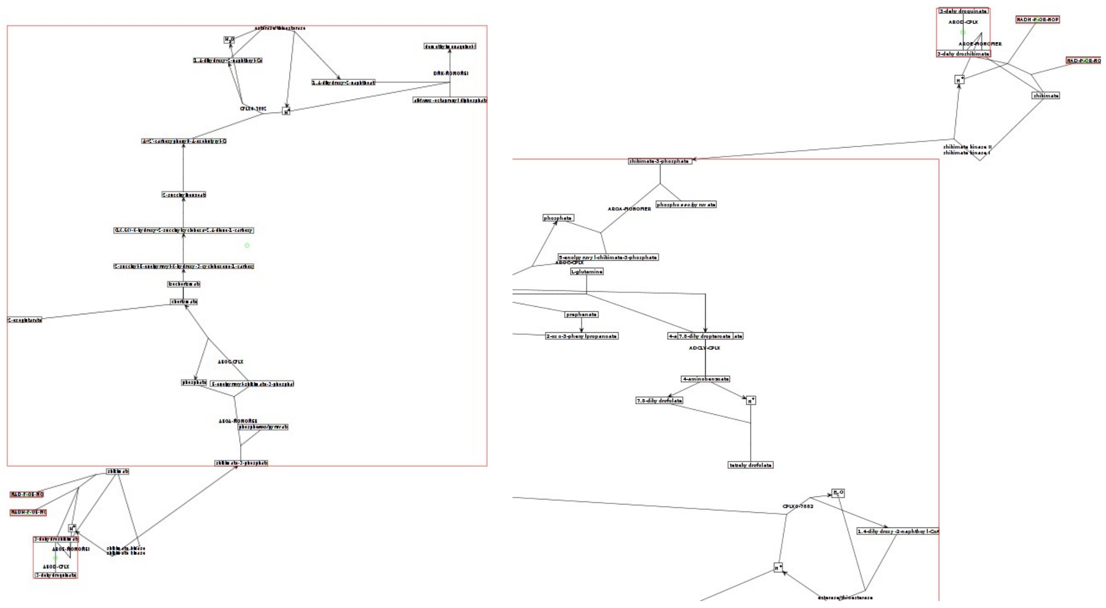
Figure 7.9: Dynamic layout, using heavy old layout nodes and edge-node repulsion for the spring embedder. The supernodes are surrounded by their bounding boxes in red, with small green circles indicating the center of the bounding box.

every step was calculated without any noticeable delay to the user.

7.4 Future Work

We have touched on some areas where improvements to our method are possible. Here we will list these and other ideas for future work.

- Investigate the parameters which determine whether a supernode should be placed inside or outside a cycle, finding a good balance between the suggestions of Karp and Becker.



(a) Reset after 7.9b. *(b) Reset after 7.9d.*

Figure 7.10: Redrawing the pathways in Figure 7.9 without maintaining the mental map. (a) shows the entire graph, while (b) shows only a part.

- Implement Schreiber's layered layout and the simulated annealing techniques to compare them with our algorithm.
- Selection of pathways to combine. The current prototype allows the user to select a pathway from a list of all neighbouring pathways of the current layout. However, other options are possible. It is interesting to determine the use cases in which biologists want to combine pathways.

  In general we may want to present the user with the ability to select pathways based on some criteria. For example, users may want to see all pathways involving a certain compound also present in the current pathway; or combine pathways which are located in the same parts of the cell. Partial matches can be sorted and presented to the user, resulting in more ways to explore the metabolic network.
- Using foresighted layout. Diel et al. [6] introduce a way to maintain the mental map where the layout of the current graph takes into account changes that the user makes. This requires us to be able to know the actions the user makes from the start. In our case, this is clearly not feasible, however we can make some educated guesses. We know the entire metabolic network from which the user can select smaller pathways to combine. The pathways the user selects will always be connected to the currently drawn network. Moreover, the user only ever adds pathways to the graph, never removing them, restricting the possible changes even more.

  If the user is able to search for certain criteria and then select pathways based upon their search, we can make an even better guess, since we can only use the top results when calculating the foresighted layout.

  While it is not trivial to combine foresighted layout with metabolic pathway exploration, it is certainly worth investigating.

- Experiment with the balance between a good layout and maintaining the mental map. In our algorithm, this can be done by experimenting with the mass of old nodes in the spring embedder. When implementing simulated annealing, we can do this by giving different weights to the mental map cost and the good layout cost.

8 Conclusion

In this thesis, we have examined different possibilities to create static layouts for metabolic pathways. We have detailed the method by Karp and Paley [9], along with the extensions created by Becker and Rojas [2]. We have also explained how Schreiber [18] uses layered layouts for this purpose.

We have explained the concept of a mental map and given formal criteria for which we can optimize. We have discussed the Force Scan algorithm by Misue et al. [15], as well as the adaptation of the simulated annealing technique by Lee et al. [13]

Our approach relies on a combination of the algorithms by Karp and Paley [9], Becker and Rojas [2], and Misue et al. [15] We take special care when trying to find a good balance between maintaining the mental map and creating a clear layout. This results in an algorithm which is good at maintaining the mental map when combining multiple metabolic pathways.

The prototype application which implements this algorithm generates these layouts quickly, without any noticeable delay. While the prototype is algorithmically complete, it can use some aesthetic improvements.

Bibliography

[1] Ron D Appel, Amos Bairoch, and Denis F Hochstrasser. A new generation of information retrieval tools for biologists: the example of the expasy www server. *Trends in biochemical sciences*, 19(6):258–260, 1994.

[2] Moritz Y Becker and Isabel Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, 17(5):461–467, 2001.

[3] Stina S. Bridgeman and Roberto Tamassia. A user study in similarity measures for graph drawing. *J. Graph Algorithms Appl.*, 6(3):225–254, 2002.

[4] Ron Caspi, Hartmut Foerster, Carol A Fulcher, Pallavi Kaipa, Markus Krummenacker, Mario Latendresse, Suzanne Paley, Seung Y Rhee, Alexander G Shearer, Christophe Tissier, et al. The metacyc database of metabolic pathways and enzymes and the biocyc collection of pathway/genome databases. *Nucleic acids research*, 36(suppl 1):D623–D631, 2008.

[5] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985. 10.1007/BF00940812.

[6] Stepahn Diel, Carsten Görg, and Andreas Kerren. Preserving the mental map using foresighted layout. In *Proceedings of the 3rd Joint Eurographics-IEEE TCVG conference on Visualization*, pages 175–184. Eurographics Association, 2001.

[7] Peter Eades, Wei Lai, Kazuo Misue, and Kozo Sugiyama. *Preserving the mental map of a diagram*. International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.

[8] Minoru Kanehisa, Susumu Goto, Shuichi Kawashima, Yasushi Okuno, and Masahiro Hattori. The kegg resource for deciphering the genome. *Nucleic Acids Research*, 32(suppl 1):D277–D280, 2004.

[9] Peter D Karp and Suzanne Paley. Automated drawing of metabolic pathways. In *Proceedings of the 3rd International Conference on Bioinformatics and Genome Research*, pages 225–238. Citeseer, 1994.

[10] Ingrid M. Keseler, Amanda Mackie, Martin Peralta-Gil, Alberto Santos-Zavaleta, Socorro Gama-Castro, César Bonavides-Martínez, Carol Fulcher, Araceli M. Huerta, Anamika Kothari, Markus Krummenacker, Mario Latendresse, Luis Muñiz-Rascado, Quang Ong, Suzanne Paley, Imke Schröder, Alexander G. Shearer, Pallavi Subhraveti, Mike Travers, Deepika Weerasinghe,

Verena Weiss, Julio Collado-Vides, Robert P. Gunsalus, Ian Paulsen, and Peter D. Karp. Ecocyc: fusing model organism databases with systems biology. *Nucleic Acids Research*, 41(D1):D605–D612, 2013.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[12] Antoine Lambert, Jonathan Dubois, and Romain Bourqui. Pathway preserving representation of metabolic networks. In *Computer Graphics Forum*, volume 30, pages 1021–1030. Wiley Online Library, 2011.

[13] Yi-Yi Lee, Chun-Cheng Lin, and Hsu-Chun Yen. Mental map preserving graph drawing using simulated annealing. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, pages 179–188. Australian Computer Society, Inc., 2006.

[14] G Michal. Biochemical pathways (poster). *Boehringer Mannheim, Penzberg*, 1993.

[15] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of visual languages and computing*, 6(2):183–210, 1995.

[16] Helen C Purchase, Eve Hoggan, and Carsten Görg. How important is the "mental map"?–an empirical investigation of a dynamic graph layout algorithm. In *Graph drawing*, pages 184–195. Springer, 2007.

[17] Helen C Purchase and Amanjit Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In *Diagrammatic Representation and Inference*, pages 60–73. Springer, 2008.

[18] Falk Schreiber. High quality visualization of biochemical pathways in biopath. *In Silico Biology*, 2(2):59–73, 2002.

[19] Falk Schreiber, Tim Dwyer, Kim Marriott, and Michael Wybrow. A generic algorithm for layout of biological networks. *BMC Bioinformatics*, 10(1):375, 2009.

[20] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2):109–125, 1981.

[21] Imre Vastrik, Peter D'Eustachio, Esther Schmidt, Geeta Joshi-Tope, Gopal Gopinath, David Croft, Bernard de Bono, Marc Gillespie, Bijay Jassal, Suzanna Lewis, Lisa Matthews, Guanming Wu, Ewan Birney, and Lincoln Stein. Reactome: a knowledge base of biologic pathways and processes. *Genome Biology*, 8(3):R39, 2007.