A DRAWING TOOL FOR PROTEIN INTERACTION MAPS IN KOHN NOTATION

by

Mine EDES

B.S., Manufacturing Systems Engineering, Sabancı University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computational Science and Engineering
Boğaziçi University

2011

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Assoc. Prof. Dr. Can Özturan for his contributions.

I also would like to thank Prof. Türkan Haliloğlu, Itır Karaç, Nihal Korkmaz and Melih Sözdinler for their support and help throughout my thesis and giving me the opportunity to work with them.

ABSTRACT

A DRAWING TOOL FOR PROTEIN INTERACTION MAPS IN KOHN NOTATION

Cancer is one of the most lethal and common diseases of today. Causes of cancer have been studied widely as it is an essential part of cancer research. Cancer is mainly caused by malfunctioning of the tumor suppressor proteins. Generally, these proteins take part in protein-protein interaction networks. Hence, to understand the suppressor proteins, it is important to study these networks. Results from this study will have great impact on the cancer research, drug design and protein engineering. For that reason, these networks are represented in network diagrams called Protein Interaction Maps to visualize and understand them better.

Today, there is still a need for the standard for visualization of the protein interactions. For this reason, Kohn and his group's MIM (Molecular Interaction Map) notation is considered to be the answer to that need. Even though there are some tools for graphical visualization of protein interactions, there is no tool that can draw protein interactions with MIM notation with full support. Thus, in this study we aimed to design a tool that can draw with Kohn's notation. We developed MIMTool; a drawing tool for manually drawing protein interaction maps in Kohn notation. Later, as one of the most important part of our study, we added a semi-automatic map drawing feature to the tool. This feature automatically draws the interactions between physical entities using Dijkstra's shortest path algorithm.

With MIMTool, it will be much faster to draw, update and exchange molecular interaction maps. Use of this tool will save time and decrease work load.

ÖZET

KOHN NOTASYONU İLE PROTEİN ETKİLEŞİM HARİTALARINI ÇİZME ARAÇI

Kanser günümüzün en sık görülen ve en ölümcül hastalıklarından biridir. Bu nedenle, kanserin nedenleri en çok araştırılan konulardandır. Kanserin nedenlerinden bir tanesi, çeşitli sebeplerle, tümör baskılayıcı proteinlerin, işlevlerini gerektiği gibi yerine getirememesidir. Bu proteinler genellikle işlevlerini karşılıklı etkileşimlerle gerçekleştirirler. Bu sebeple, baskılayıcı proteinleri anlamak için , bu etkileşimleri incelemek önemlidir. Bu incelemenin sonuçlarının, kanser aratırması, ilaç tasarımı ve protein mühendisliği gibi alanlarda önemli etkileri olacaktir. Bu nedenle, protein etkileşimleri Protein Etkileşim Haritalarının aracılığıyla görselleştirilmekte ve bu sayede daha iyi incelenebilmektedir.

Günümüzde protein etkileşimlerinin toplu ifadesini gösterecek bir sembol standardına ihtiyacı vardır. Bu sebepten, Kohn ve grubunun MIM notasyonunun, bu ihtiyacı karşılayacağı ön görülmektedir. Protein etkileşimlerinin grafik gösterimi için yazılmış bazı yazılımlar olmasına rağmen, MIM notasyonunu tüm ayrıntıları ile destekleyen bir araç daha geliştirilmemiştir. Bu nedenle, bu tezde, Kohn notasyonu ile protein etkileşimlerini çizebilen bir araç geliştirmeyi hedefledik. Öncelikle, elle çizebilen bir araç geliştirdik. Daha sonra bu araca, araştırmamızın en önemli kısımlarından biri olan, yarı otomatik grafik çizme kısmını ekledik. Bu kısımda Dijkstra'nın en kısa yol bulma algoritmasını kullandık. Bu sayede, fiziksel varlıklar arasındaki etkileşimler otomatik olarak çizilebiliniyor.

Bu araç sayesinde, Protein Etkileşim Haritalarını çizmek, güncellemek ve değiştirmek çok daha hızlı olacak. Bu da, harcanan zamanı ve gereksiz iş yükünü azaltacaktır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
BMP	Bitmap
EPS	Encapsulated PostScript
GML	Graph Modelling Language
GO	Gene Ontology
GPML	GenMAPP Pathway Markup Language
GUI	Graphical User Interface
JPEG	Joint Photographic Experts Group
MIM	Molecular Interaction Map
MIMML	Molecular Interaction Map Markup Language
OBO	Open Biological and Biomedical Ontologies
PDF	Portable Document Format
PNG	Portable Network Graphics
PSI-MI	Proteomics Standards Initiative Molecular Interactions
SBGN	Systems Biology Graphics Notation
SBI	The Systems Biology Institute
SBML	Systems Biology Markup Language
SBW	Systems Biology Workbench
SIF	Simple Interaction Format
SVG	Scalable Vector Graphics
XGMML	Extensible Graph Markup and Modelling Language
XML	Extensible Markup Language

1. INTRODUCTION

Cancer is one of the most lethal and common diseases of today. It is mainly caused by malfunctioning of the tumor suppressor proteins. The most widely studied tumor suppressors are p53 and p73. To study these proteins, it is important to understand them in detail. Generally, they take part in protein-protein interaction networks. Hence, it is important to study the networks they participate in. These networks are represented in diagrams which are called Protein Interaction Maps. These maps visualize the networks and make them easier to understand. Results from this study can have great impact on the cancer research, drug design and protein engineering.

However, there is still no widely accepted standard for visualization of the protein interactions. There are several proposals for this standard. They include Pirson [1], Cook [2], Kitano [3], Kurata [4], and Kohn [5].

Kohn proposed MIM (Molecular Interaction Map) notation and it is one of the most influential proposals. Unfortunately Kohn notation has a downside; lack of drawing software. One way to increase recognition and usage of a visualization notation is through software support. Biologists turned towards sofware because it makes it easier to store, draw and update maps which saves time and decreases workload. As a result, there has been a vast increase in the number of tools developed for biological network visualization. However, there is no tool that has a full support of MIM. There are some tools that have limited MIM notation support. However they lack flexibility and efficiency since they do not have full support. In conclusion, for MIM notation to be widely accepted and used, software support should be increased.

1.1. Contribution of Thesis

In this thesis, we aimed to design a software tool called MIMTool, that can draw molecular interaction maps in MIM notation. It is designed just for MIM so that it can be structured according to MIM notation and can be modified easily for future

updates. We wanted to design a tool that is both flexible and easy to use. Also we wanted it to be compatible with other visualizations tools as much as possible.

After studying Kohn notation, we realized that it mainly consists of molecular entities drawn as rectangular shapes and interaction lines that connect those entities. Therefore we used graph theory concepts for storing and analyzing MIM. After relating Kohn notation to graph theory, we realized MIMTool had to have an easy to use manual orthogonal line drawing. Thus, we added manual orthogonal line drawing and movement. Secondly, we wanted to add basic automatic drawing ability to the software. As a first step, we added the semi-automatic drawing algorithm. It enabled user to draw interactions lines between two placed entities automatically.

For the semi-automatic drawing algorithm, we used Dijkstra's weighted shortest path algorithm. With this algorithm, we aimed to draw the interaction line with minimal crossings and minimal number of bends.

We wanted MIMTool to have a clean user interface and useful features that can compete with other visualization tools. It is open source and supports MIMML and SBML. MIMML is developed by Kohn's group to store graphical information of MIM [6]. Our tool is one of the few tools that support MIMML. With this tool, it will be much faster to draw, update and manipulate molecular interaction maps, which will save time and decrease unnecessary work load.

1.2. Outline of Thesis

Related work is presented in the next three chapters. First Kohn notation is explained briefly. MIM notation drawing rules are presented. We also describe the relationship between biochemistry and graph theory. Secondly, basic graph theory and graph aesthetics definitions are given. Shortest path problem is explained briefly. In addition, Dijkstra's shortest path algorithm is explained. Third, some of the other drawing tools similar to our MIMTool have been represented and compared. In chapter 5, we describe our MIMTool. We describe its features and structure in detail. Then we

describe our semi-automatic shortest path algorithm. We compare it to the original Dijkstra's algorithm and describe changes needed for our MIMTool. In chapter 6, we test our semi-automatic drawing algorithm with a couple of examples. We compare the results aesthetically. In chapter 7, we draw our conclusions.

2. KOHN NOTATION

In the last decade, a lot of data has been gathered about molecular interactions that regulate cell behavior. These data need to be organized and analyzed. For that reason there is a need for a standard notation to describe bimolecular interaction networks. There are some generally accepted notions about what a node and an edge can represent but still lack of a standard creates a problem. There are several efforts to solve this problem. Two most controversial notations are SBGN [7] and Kohn's MIM notation. Kohn notation's advantage is that, it has some important capabilities making it a suitable candidate for a graphical standard: (1) ability to include contingencies, (2) enabling a user to specify known molecular data like protein modifications and complex formation, (3) displaying the complex set of regulatory network interconnections in rich detail, and (4) ability to capture different cell types and cell states since it can show the interactions of molecules that exist at the same time and place. These properties make this notation more suitable for biologists. Also MIM is considered better for computer simulation, organizing information and representation of complex combinatorial schemes [8, 9].

2.1. Rules of MIM Notation

Rules of MIM notation outline how a MIM should be drawn. Figure 2.1 lists the rules of MIM notation.

2.2. The MIM Notation

In MIM notation, there are two basic elements. One of them is physical entities, also formerly known as molecular species. These entities can be elementary or complex [5].

Rules and definitions of the MIM notation

1. A named molecular species generally appears in only one place on a map. (Exempt from this rule are molecules, such as GTP or ubiquitin, that act in a similar manner in a large number of different reactions. For clarity, the named species and its interactions must sometimes be duplicated upon translocation from one cell compartment to another.)

2. Interactions between molecular species are shown by different types of connecting lines, distinguished by different arrowheads or other terminal symbols.

3. Interaction lines can change direction (but not by more than 90° at a corner—this restriction prevents ambiguities at branch points).

4. When lines cross, it is as if they do not touch.

5. Symbol definitions are not affected by color. Color is optional: it can be used as an independent visual parameter to guide the eye and/or emphasize particular features of the network. We use red for inhibitions and other negative actions; the net effect of a sequence of interactions (whether positive or negative) can then be determined by whether the number of red-colored steps is even or odd. We use green for stimulatory or catalytic actions, blue for covalent modifications, and purple for transcription/translation.

6. A small filled circle ('node') on an interaction line indicates the consequence or product of the interaction. Thus, the consequence of binding between two molecules is production of a dimer, which is represented by a node on the binding interaction line. The consequence of a modification (e.g., phosphorylation) is production of the modified (e.g., phosphorylated) molecule; the phosphorylated product is represented by a node placed on the modification line.

7. Multiple nodes on an interaction line represent exactly the same molecular species. To avoid ambiguity, a node should not be placed at a line crossing.

8. An isolated node (a node that is not on a line) is an abbreviation that represents another copy of the same molecular species that is defined at the other end of the line pointing to the node (to avoid ambiguity, only one arrow should point to an isolated node).

9. Molecular interactions are of two types, reactions and contingencies. Reactions operate on molecular species; contingencies operate on reactions or on other contingencies.

10. A line without arrowheads is a 'state-combination' symbol. A node on this line represents the combination of states defined by the symbols at the two ends of the line.

11. MIMs may be interpreted as 'explicit', 'heuristic', or 'combinatorial.'

Figure 2.1. Rules of MIM Notation as stated by Kohn [5].

Elementary species are proteins and DNA elements. A proteins node symbol is represented with a rectangle with a rounded corner. If protein has domains, then domains are drawn in a separate rectangle and are placed left of the protein. DNA elements node symbol is a box (a rectangle). Name of the DNA element is written inside the box unless it has a consensus sequence [5].

Complex molecular species are represented by filled circles. These species can be placed on the interaction line to indicate it is the complex species of the interaction. These complex species must reside on the interaction line. Some complex dots can be independent of an interaction line to indicate a homodimer or a state-combination. Figure 2.2 show an example of each of these molecular species [5].

Second basic element is molecular interactions between species. There are two types of interactions: reactions and contingencies. Reactions operate on molecular species; they only connect two molecular species together [5]. Contingencies on the other hand are between molecular species and interactions. They emerge from species to an interaction (edge) to show the effect of that species to the interaction. The species can stimulate, be required for, inhibit or accelerate the interaction they affect. Kohn notation has different line ending for each kind of reaction. List of molecular interaction symbols are given in Figure 2.3 [5].

There are three interpretations of MIM notation: explicit, heuristic and combinatorial. In explicit interpretation, meaning of MIM is straight forward. Interaction lines affect only those species that are connected to it. There is no uncertainty. Hence explicit MIMs are suitable for computer simulation. On the other hand, heuristic and combinatorial maps are vaguer. In heuristic maps, the interaction line symbolizes the interactions that can occur if proper circumstances are present. Heuristic MIMs show all the available information and hence can be used to organization purposes. In combinatorial interpretation, all of the interactions can occur unless it is prevented by a contingency [8]. These three interpretations can give different meaning to the same MIM. Therefore, a map should always be stated as explicit, heuristic or combinatorial [8].

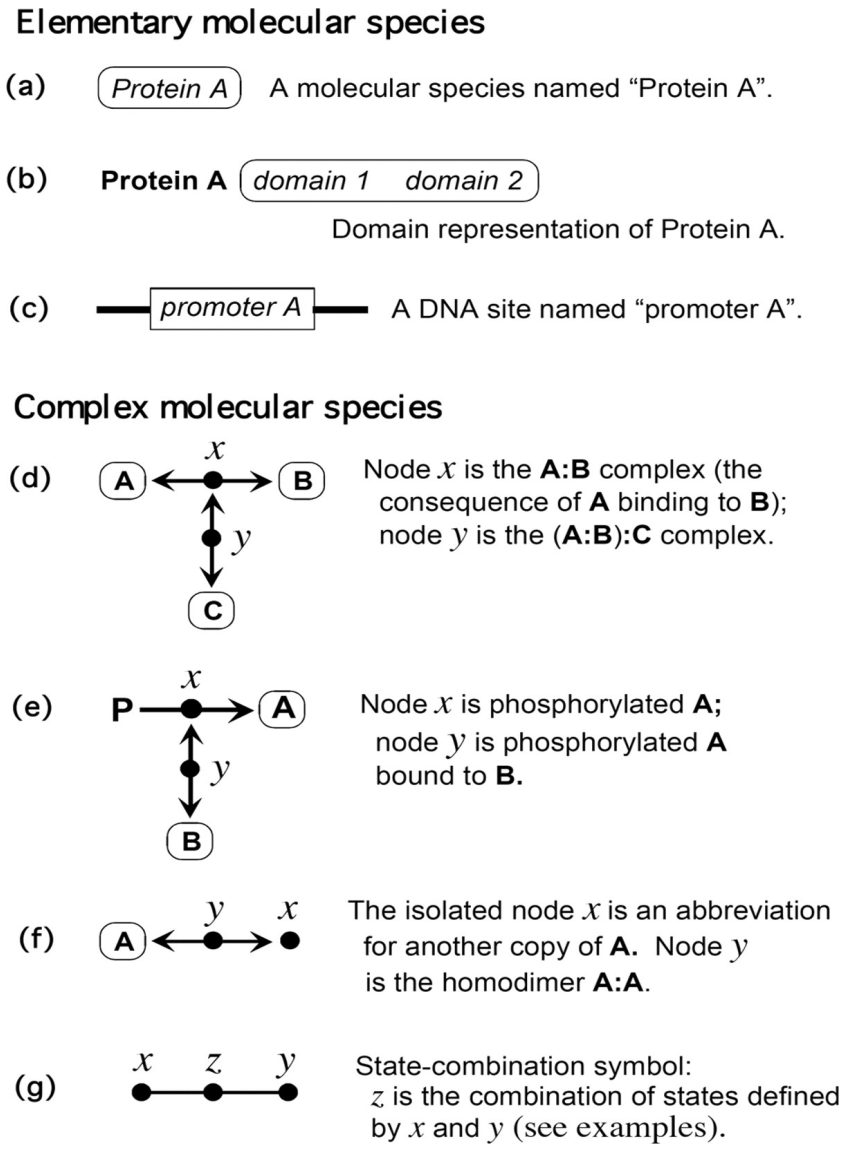

Figure 2.2. Notation of Molecular Species as stated by Kohn [5].

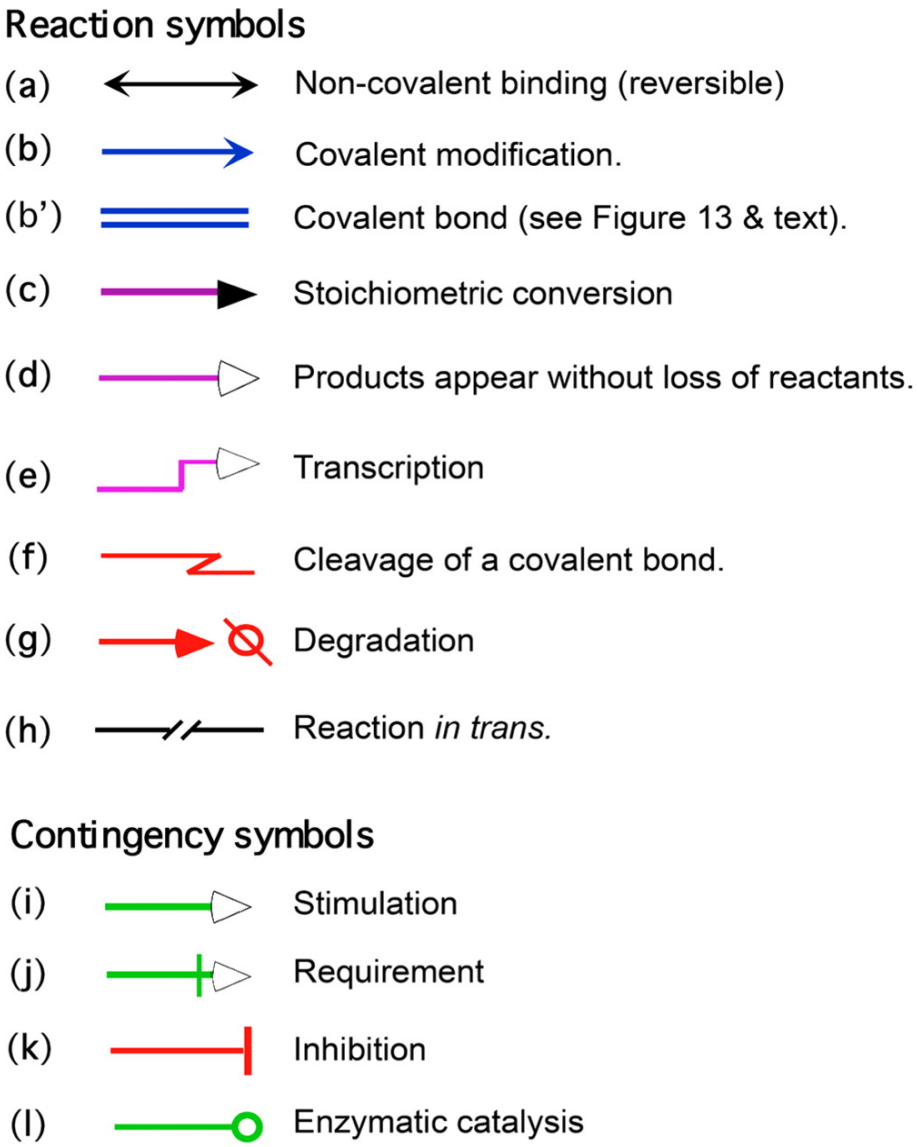

Figure 2.3. Molecular Interaction Symbols as stated by Kohn [5].

Kohn notation is very flexible and compact. The only challenge is to learn the notation [8]. Once it is learned, it is very fast and efficient. Now, with the tool that we have created, it is even faster and easier to use. Figure 2.4 is an example of a typical MIM.

2.2.1. Graph visualization of biological networks

Graph theory is a branch of combinatorial analysis. It has so many applications and impacts on other areas that, its development benefits not just mathematics. Biochemistry is one those fields. It is one of the breeding grounds for graph theory. The reason is that, chemistry has suitable subjects that can be shown with graphs and can relate to its features; like points, lines, connectedness, neighbors, etc. Also combinatorial derivation of graphs can relate to combinatorial derivation of many graphical concepts in chemistry. The result is that graphical visualization has a lot of benefits for chemists. Collaboration of graph theorists and biochemists will continue for years to come [10].

There are two kinds of general compatibility between graph theory and biochemistry. One of them is when a graph corresponds to a molecular geometry; nodes are atoms and edges are chemical covalent bonds. Second one is when a graph corresponds to a reaction mixture; nodes are chemical species and edges are conversions between them. Kohn notation falls into the former category; a graph corresponds to a biological network where the nodes are molecular species and the edges are interactions between them [10]. Using graph theory, we can also find the shortest path between two nodes that is needed for our semi-automatic drawing algorithm.

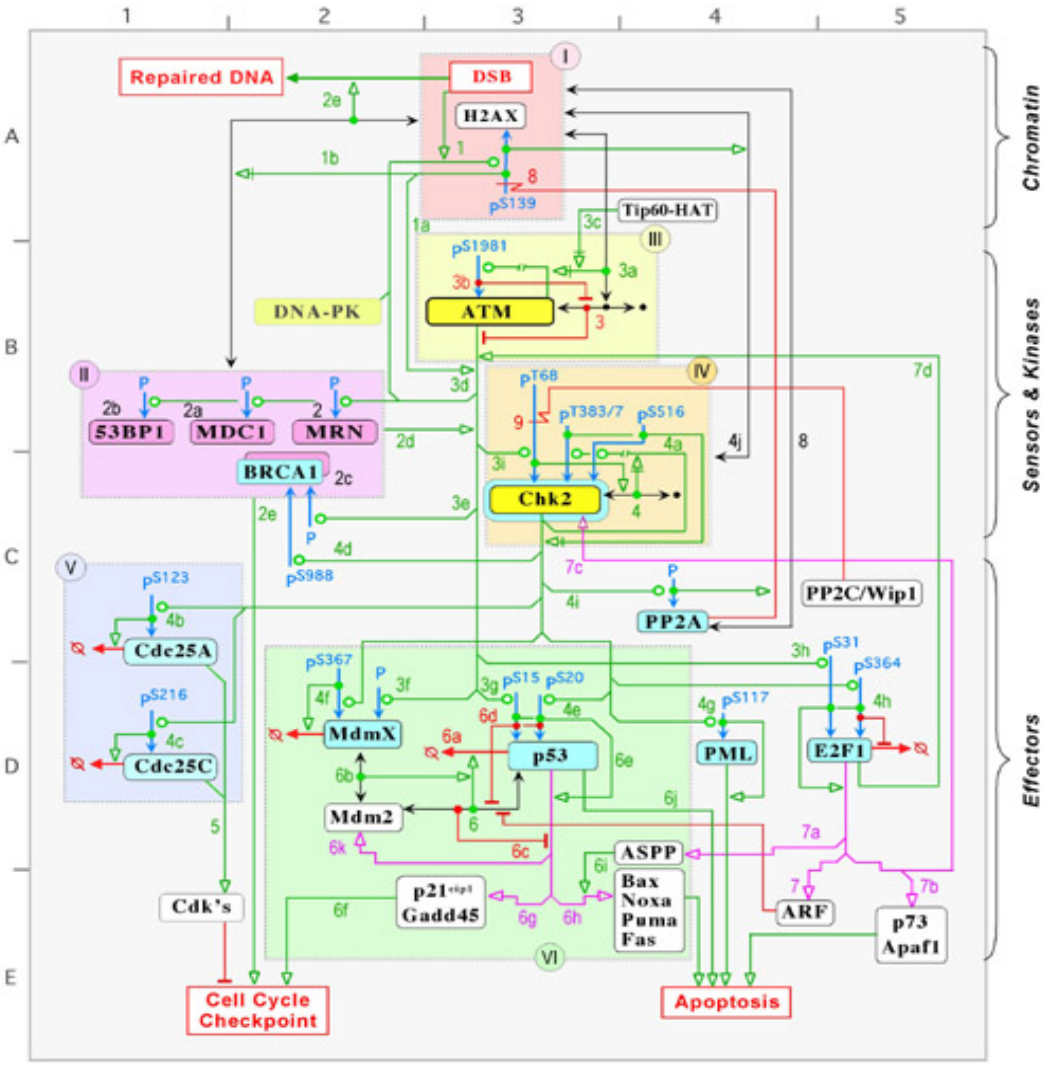

Figure 2.4. Chk2 activation in response to double stranded DNA breaks implemented by Kohn [6]

3. GRAPH THEORY CONCEPTS AND SHORTEST PATH PROBLEM

3.1. GRAPH THEORY CONCEPTS

Graph theory has grown rapidly in the last couple of decades even though it is a young subject. That is because it has so many applications in so many different fields. Tollis [11] gives some examples; software engineering, database design, information systems, real-time systems, decision support systems, VLSI, artificial intelligence, logic programming, medical science, biology, chemistry, civil engineering and cartography.

In this chapter, first some terminology used for Kohn notation are explained. Then aesthetics and graph drawing standards are expalined that are used for MIM drawing. Next section will focus on shortest path problem that is used in our semi-automatic drawing algortihm.

3.1.1. Terminology

A graph is represented by G=(V,E) where V are vertices and E are edges. Vertices can also be called nodes and edges can be called links, arcs or connections according to the application. If these vertex edge pairs are ordered, then this graph is called directed graph, or diagraph. Directed edges are usually drawn with arrows to indicate the direction. If edges do not have a direction, then it is called an undirected edge. A graph consisting of only undirected edges is called undirected graph. The indegree of a vertex is the number of incoming edges and outdegree of a vertex is the number of outgoing edges. Degree of a vertex is the total number of edges going in and out of that vertex. Two vertices are adjacent if they are connected by a common edge e = (u, v). Two edges are called incident if they are connected by a common vertex. A walk is group of ordered edges, where each edge is incident to the next edge. A path is a walk where a vertex can only appear once. A graph is connected if there is a path from any

vertex to any other vertex. If a graph is not connected, it is called disconnected. A planar graph is a graph where none of the edges cross. Further details on graph theory terminology can be found in [1, 11, 12, 13].

3.1.2. Graph aesthetics

Readability of a graph is its ability to transfer the meaning easily by its drawing. Therefore graph theory is not only interested in different types of graph but also on the placement of the vertices and edges so that the graph is easily understood; or readable.

Aesthetics of a graph are some properties that a graph should have to be more readable. Tamassia, Battista and Batini [11, 13] stated taxonomy of graph aesthetics. They include: *crossings, area, total edge length, maximum edge length, uniform edge length, total bends, maximum bends, uniform bends* and *symmetry.*

Crossings is minimization of total number of crossings between edges in the graph. If there are no crossings, then the graph admits planar drawing and is a planar graph. *Area* is the minimization of the area that graph embodies. Minimizing the area too much, can lead to reduced readability. Therefore it should be thoughtfully optimized for the application. *Total edge length* is the minimization of the total lengths of all the edges in the graph. *Maximum edge length* is the minimization of the longest edges length in the graph. *Uniform edge length* is optimization of the variance of the lengths of edges. *Total bends* is the minimization of total number of bends. This one is applied to many orthogonal drawings. *Maximum bends* is the minimization of the total number of bends on an edge. *Uniform bends* is the minimization of variance between bends totals on different edges. *Symmetry* is the optimization of the graph element placement on the graph [11].

All these aesthetic elements are applied according to the application. Ordinarily just one aesthetic criterion is not enough. Therefore, combination of these aesthetics are attempted to be optimized on the same graph. Unfortunately, most of the time an optimization suitable for one aesthetic does not optimize the other. Therefore, if

more than one aesthetic is considered for an application, then an algorithm should be written that can assign different weight to different aesthetics to achieve a more optimized solution [13].

3.1.3. Graphs drawing standards

Graphs can be described by their drawing standard. Drawing standards describe the vertex and edge placement. These include polyline drawing, straight-line drawing, orthogonal drawing, grid drawing, planar drawing and upward (resp. downward) drawing. A graph can have more than one drawing standard [11, 13].

Straight-line drawing is the simplest and one of the widely used standards. With straight-line drawing, each edge is drawn as a straight line. In polyline drawing, each edge is drawn as a polygonal chain of straight lines. In orthogonal drawing each edge is a chain of vertical and horizontal lines. This drawing is also called rectilinear drawing or Manhattan drawing. It is used in many applications such as in VLSI design or in entity-relationship diagrams. It is also used in Kohn notation. In grid drawing, vertices and edges are placed on a grid; vertices are placed on corners and edges bend at corners; so that they have integer coordinates. In planar drawing, no edges cross. Finally upward drawing is applied to acyclic graphs; each edge is drawn in monotonically non-decreasing fashion. In downward drawing, it is the opposite; each edge is drawn as monotonically non-increasing [11].

3.1.3.1. Orthogonal Grid Drawing. Kohn notation supports only orthogonal drawing standard. It means that the interaction lines in a molecular interaction map must consist of vertical and horizontal lines.

Orthogonal drawing standard has got a lot of attention because of its application to many real life systems. These include VSLI, database design, circuit design, etc.

Many of the research have focused on drawing of planar orthogonal graphs. Exam-

ples include Tamassia [14], Battissa [15] and Fößsmeier and Kaufmann [16]. However, these algorithms generally apply to graphs without a given layout. Even if there is a given layout, then the algorithm places the nodes(mostly on a grid), then draws all the interaction lines with respect to their position simultanously. For our semi automatic drawing problem, the nodes are already placed on the map. Also MIMs are not planar and contain nodes with degree more than four. In addition, we want to draw only a path between two nodes. Therefore, our drawing problem is closer to edge routing problem or shortest path problem.

3.2. SHORTEST PATH PROBLEM

Finding shortest path is one most studied topics of network analysis. One of the reasons is that shortest path problem has many applications in both research and also in real life applications. Shortest path problems can relate to numerous different topics. Another reason is shortest path problems can be solved efficiently and easily. There are algorithms for various types of shortest path problems which run in polynomial time [12].

Consider a connected directed graph G= (N, E) where N is nodes and E is edges. Each edge has an associated length l(i,j) where i and j are end nodes of that edge. Let s be one of the chosen nodes (source) and d be another chosen node (destination) on graph G. Length of a path is the sum of edge values in that path. Shortest path problem is finds the path with the minimum length from the chosen source s to the destination node d [12, 17].

The length of the shortest path is called shortest distance. Shortest distance from a node to itself is zero [15].

There are three general types of shortest path problems that have been studied:

(i) Single source shortest path problem: finding shortest path from a chosen source node s to all the other nodes or finding shortest path from a chosen

source s to a chosen destination node d

(ii) All-pairs shortest path problem: finding shortest path from every node to every other node.

(iii) Single destination shortest path problem: equal to the reverse of single source shortest path problem.

There are also variations of each these types; finding single source shortest path problem with non negative edge costs, finding single source shortest path problem with arbitrary edge costs, finding shortest path in acyclic networks, etc. Variations occur from the characteristics of the application.

Even though shortest path problem seems simple, it can be applied to variety of problems. Its direct and indirect implementations are numerous to list. Some well known problems stated in [12] are knapsack problems, urban traffic planning, DNA sequencing, inventory planning, project management, minimum cost flow problem, telephone operator scheduling problem.

There are some known algorithms to solve shortest path algorithm like Bellman-Ford algorithm, Floyd-Warshall Algorithm and Dijkstra's algorithm. In this thesis our focus will be on Dijkstra's algorithm. For more information about other algorithms, reader can check [12, 18].

3.2.1. Dijkstra's Algorithm

E. W. Dijkstra wrote his algorithm in 1959. It solves the single source shortest path algorithm in a connected graph where lengths of the graph are non-negative.[19]

In Dijkstra's algorithm, nodes of the graph are partitioned into two sets: labeled nodes and unlabelled nodes; A and B respectively. Every node has a distance parameter which is the minimum distance from source to that node. If the graph is unweighted, we can set the lengths of all the edges to the same constant number [17, 18].

First, every node is in set B and has a distance of infinity except the starting node whose distance is zero. The algorithm then selects the node with the minimum distance. (It is always the source node at the first iteration). Selected node is transferred to set A. As nodes are selected, distances of the neighbor nodes are updated according to the lengths of nodes. If there is an improvement in the distance of the already visited nodes (in set A), they are also updated. This process is repeated until all the nodes are in set A, which indicates that all the distances are calculated. In another words, the algorithm starts from source node and fans out in all directions until it has reached all the nodes. Its pseudo code is given in Figure 3.1 [12].

```
G=(N,E)
s = source node
c(i,j) = length(cost) of the edge between nodes i and j
d(i) = distance from source to node i
A(n) = adjacency list of node n

begin
   A = ø;
   B = N;
   d(i) = ∞ for all i element of N;
   d(s) = 0;
   pred(s) = 0;
   while B != ø
      Select node n with d(n) = min {d(i): i element of B }
      B = B - {n};
      A = A ∩ {n};
      for each = j element of A(n)
         d(j) = min{d(j), d(n) + c(n, j)};
      end;
   end;
end;
```

Figure 3.1. Dijkstra' algorithm as stated in [12]

Dijkstra's algorithm assumes that the graph is connected meaning that there is a path from every node to every other node. However if this is not the case, edge length between unconnected nodes could be taken as infinity.

Running time of Dijkstra's algorithm is $O(N^2)$. There are two major operations

in Dijkstra's algorithm:

(i) Node selection: selecting the node with minimum distance
(ii) Distance update: updating distances of all the adjacent nodes

Running time is limited by node selection which takes $O(N^2)$ time. [12]

Dijkstra's algorithm has quadratic growth rate. It is optimal for dense graphs where $|E| = O(|N|^2)$. However, it is very slow if the graph is sparse; $|E| = O(|N|)$ [17]. Therefore some improvements can speed up the algorithm for sparse graphs.

There has been research on how to speed up the Dijkstra's algorithm. Some speed-up techniques include using heap data structure, bidirectional search [12], goal directed search, limiting the searched nodes [20] and multi-level search [21]. These techniques have been generated for different kinds of shortest path problems; mostly for railroad, public transport or GPS routing. In our algorithm, we limited the number of searched nodes. Some complicated area limiting techniques have been proposed for transportation problems; like graph partitioning [22] and bounding box method [23]. However, these techniques have preprocessing overhead. It is about a factor of two compared to the plain Dijktra's algorithm [23]. Therefore we have applied a different area limiting search method. In addition, we applied bidirectional search method and implemented binary heap data structure.

3.2.1.1. Dijkstra's Algorithm with Binary Heap. One way to speed up Dijkstra's Algorithm is to use heap (priority queue) data structure. In heaps, items have priorities and they are arranged in the queue according to these priorities [12, 17]. They store the items in a binary tree structure. Example of a binary tree is given in Figure 3.2.

Heap data structure has at least two operations: deleteMin and insert. There can be more operations according to the application. To speed up Dijkstra's algorithm, two operations are added [17].

- deleteMin : removes the node with minimum distance
- insert: a new node is inserted to the heap
- buildHeap : creates an empty heap
- decreaseKey : updates the distance value of the node. New distance value must be smaller than the old value.

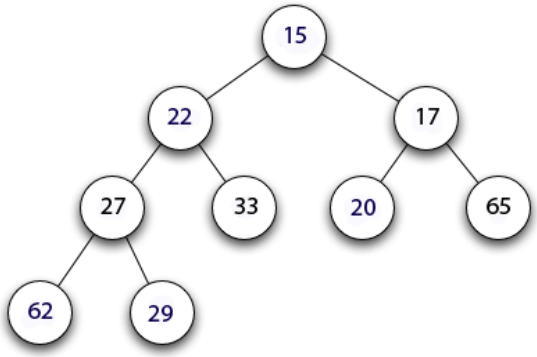

Figure 3.2. Complete binary tree

In MIMTool, we applied binary heap implementation of Dijkstra's algorithm. First an empty heap is created. Then the nodes with shortest distances are inserted to the heap. Then, they are selected one by one with deleteMin operation and removed from the heap. Neighbour nodes distances are updated using decreaseKey operation. Algorithm ends when the heap is empty and every nodes' distance to the destination has been calculated.

3.2.1.2. Bidirectional Dijkstra's Algortihm. Dijkstra's algorithm finds shortest path from each node to every other node. However, some applications only need shortest path from a fixed node to another fixed node. For these applications, Dijkstra's algorithm can be modified so that it can reduce the number of visited nodes and save time. This modified algorithm is called bidirectional Dijkstra's algortihm [12].

In the bidirectional Dijkstra's algorithm, we simultaneously run the algorithm from both source and destination nodes. We stop when the algorithm labels the same node from both sides [12]. We applied the bidirectional algortihm in our semi automatic drawing algortihm. Generally, it reduces the number of visited nodes by a factor of 1.5 [21].

4. BIOMOLECULAR NETWORK VISUALIZATION TOOLS

In this chapter, some known network visualization tools are reviewed. These tools have guided us in the development of MIMTool. There are a lot of different tools with similar functionalities. Some of them are: Interviewer [36], PATIKA [37], BioPax [38], CellDesigner [32], Cytoscape [28], Pathvisio [30]. In this thesis, we have limited our discussion to the latter three tools because they are the widely used and are more similar to MIMTool.

Except for PathVisio, none of these tools can be considered as a competitor of MIMTool. The reason is that, they cannot draw maps in MIM notation. CellDesigner can draw diagrams using Kitano's SBGN and Cytoscape support many different formats except for MIM notation. PathVisio has added a MIM plug-in last year. However it cannot draw complicated MIMs since it has limited support. Hence MIMTool cannot be compared to PathVisio also. However, they all have some similarities and basic features. Before describing them, some basic terminology needs to be explained.

XML is short for Extensible Markup Language. It has been designed by independent World Wide Web Consortium (W3C) [24]. It is a markup language designed to store and transfer data. One of its most essential qualities is its capability to define new data types and its cross-platform data transfer. Therefore, once it is learned, users can create their own tags and format for storing data. Then, they can transfer the data independent of the platform. Some of its advantages are listed below:

- Transports and stores data
- Easy to learn
- Platform independent
- Store many different types of data
- Stores data in a hierarchical format which facilitates fast data query

XML is not just a language; it is a basis to define a language that can store new data types. It contains text, data and tags. One of widely used XML based languages to define biological processes is SBML.

SBML is short for Systems Biology Markup Language. It is an XML based language for describing biological processes [25]. It was first developed in 2000 and has been evolving since. It is now being supported by more than one hundred and eighty software packages [26].

Advantages of supporting SBML are listed below:

- It is supported by many different tools. Thus, SBML data can be processed from any one of these tools without rewriting.
- SBML data can be shared among researchers in varying software environments
- Data can survive independent of the software
- It is more than a graphical modeling format. It also stores quantitative data. This data can be used for simulation and analysis.

There has been vast increase in the number of biological network simulation or drawing tools. Since there is no accepted standard, many of these tools create their own format with similar features. As a result, there is a lot of code duplication and without constant support or upgrade, most if these tools are unfortunately short lived. SBW was created to solve this code duplication problem. It is a framework designed to enable communication among tools via a binary encoded message system. As a result, developers can work together with existing tools instead of addins the same features to their tools over and over again. It is open source and can be downloaded at [27].

4.0.2. Cytoscape

One of the first and most sophisticated biological network visualization tools is Cytoscape. Its first version was created in 2002 to improve biological research. However it has grown so much that it can be used for general complex network analysis and

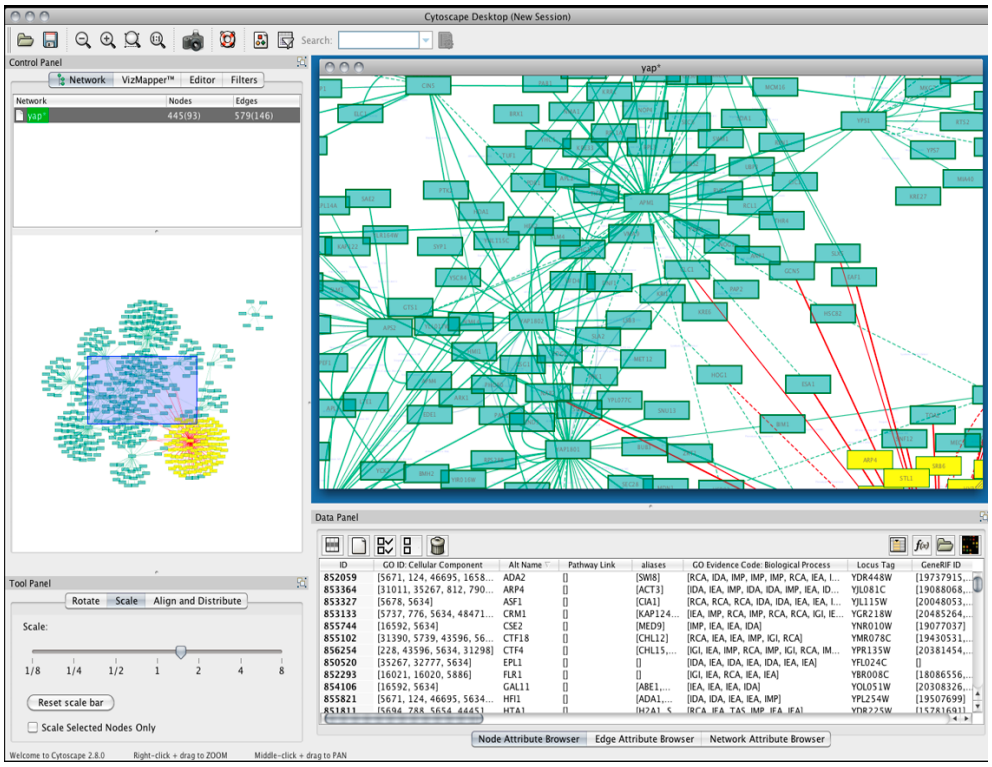

Figure 4.1. Screenshot of Cytoscape [28].

visualization [28].

Its basic distribution offers data integration and visualization. Other functionalities are added with plug-ins over the years. Since it is open source and cross-platform, plug-ins can be added easily. For now, available plug-ins enable online database connection, additional file format support, additional layout types, scripting, network and molecular profile analyzing. It can support many file formats: SBML, BioPax, OBO, SIF, GML, XGMML, PSI-MI and GO Annotation. It also supports text files and Excel spreadsheet. It can also save maps in many different image formats: PDF, EPS, SVG, PNG, JPEG, and BMP files [29].

For visualization of networks, different layout algorithms are available. These include cyclic, tree, force-directed, edge-weight, and yFiles Organic layouts. They also developed a session file called cytoscape session (.cys). It saves information about your current cytoscape file specifications: your settings, networks, attributes, plug-in states and properties. Cytoscape has a lot of useful features and is one of most advanced software platforms in bioinformatics. Its new versions are still being developed [29].

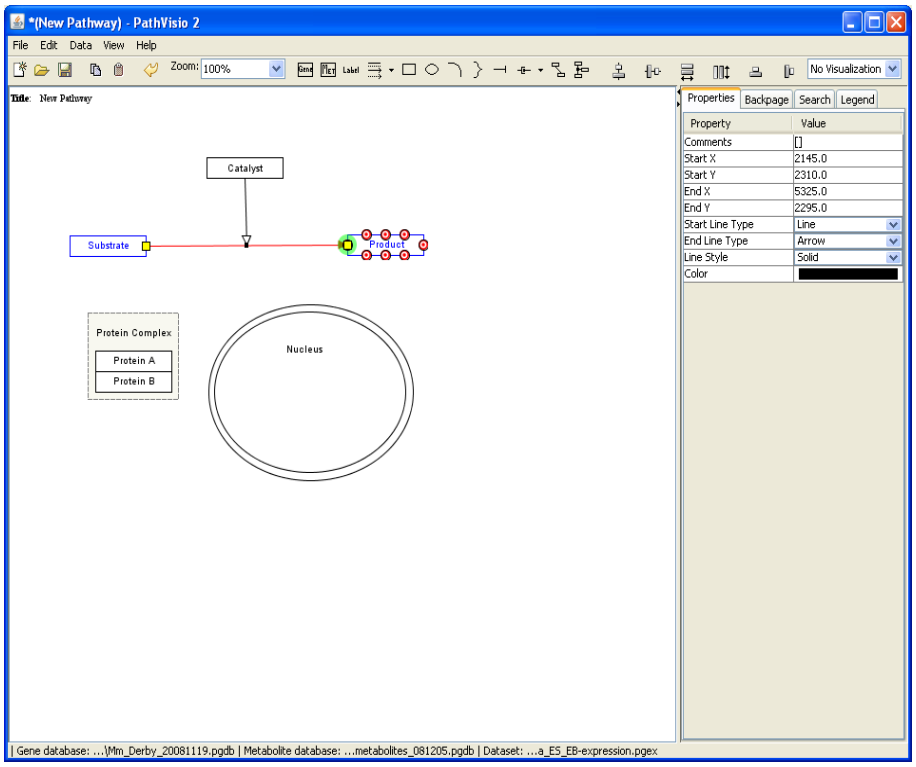

Figure 4.2. Screenshot of PathVisio [30].

4.0.3. PathVisio

Visualization of biological pathways is important for biological research. Software tools to create these pathways can help research to advance. PathVisio has created for this purpose. It was developed to enhance GenMAPP, a biological pathway visualization tool [30]. PathVisio is a simple and a user-friendly tool intended to be used by computational biologists to:

- Organize and store information
- Analyze data within a biological context by exploring pathways and pathway statistics.
- Augment available pathway information by storing true interactions(node-edge relations)

PathVisio is written in Java. Thus, PathVisio integrates with other Java based tools and supports addition of plug-ins. It uses Java Graphics2D API for visualization

of pathways. Java Graphics2D API has the ability to create different kinds of shapes and to be flexible for future changes. It is also open source and can be used by both academic and non-academic community [30].

A new XML based file format has been created for PathVisio called GPML (GenMAPP Pathway Markup Language). It is a flexible language that stores relational pathway information. Unlike SBML, it does not store information to be used for computational modeling. To increase the use of GPML, a plug-in is added to Cytoscape so that biological pathways created with PathVisio can be transferred to Cytoscape. Another advantage of this plug-in is Cytoscape's ability to handle networks. PathVisio is optimized for pathway visualization. It is not designed for network analysis. Hence, pathway information can be augmented with transferring pathway models to Cytoscape [30, 31].

Another aspect of PathVisio is that it is not designed for any specific biological graphical notation. Its data model uses commonly accepted notation; nodes represent biological entities and edges represent different interactions. Hence PathVisio does not have any restrictions and is flexible for future updates. An example is the addition of MIM notation plug-in last year. For now, it has limited support for MIM but support is planned to be improved.

PathVisio and MIMTool are similar. They are both easy to use and intended for computational biologists. Also PathVisio is the only tool that supports MIM notation. Another common property is the lack of automatic layout. PathVisio does not have any automatic layout algorithm and supports only manual drawing. However, a big advantage of PathVisio over MIMTool is its ability to access of online genome databases [30].

4.0.4. CellDesigner

Cell Designer is designed to model gene-regulatory and biochemical networks. It is a Java based, cross platform tool. Its main features are: graphical notation, model

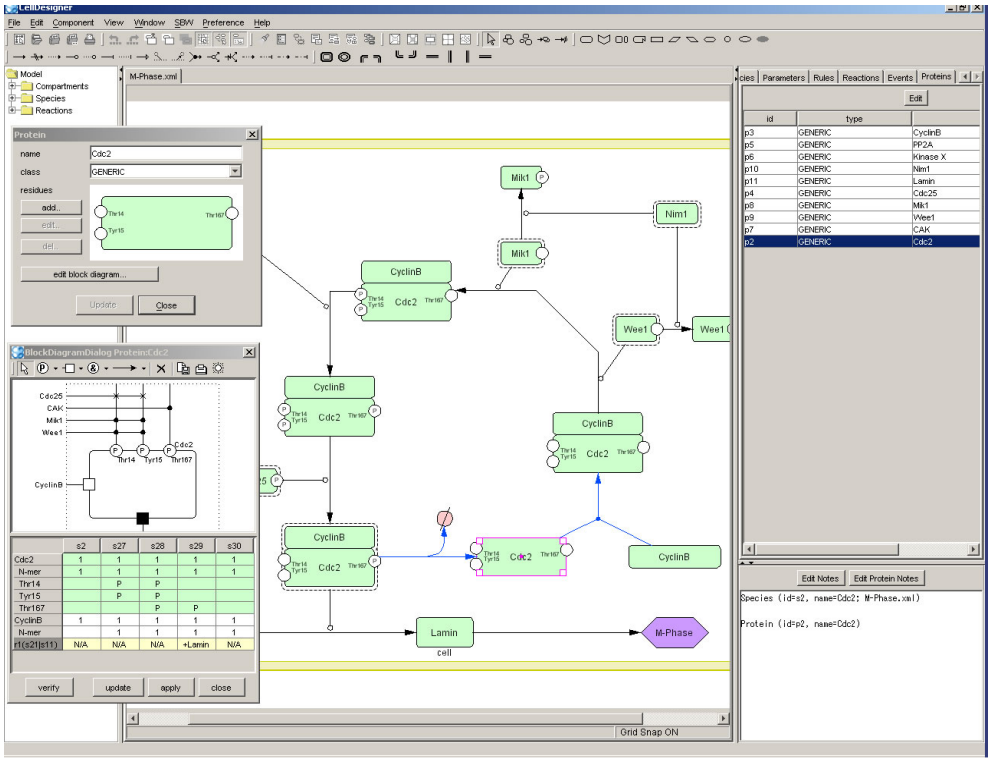

Figure 4.3. Screenshot of CellDesigner [33].

description and application integration environment [32].

For graphical notation and model description, CellDesigner supports Kitano's SBGN notation; formerly known as process diagram notation. This notation is another candidate for bio-regulatory network diagram notation standard besides Kohn's MIM. With SBGN, Kitano created a standard notation that is simple, clear and can contain sufficient information [7]. For application integration, CellDesigner has SBML and SBW support.

Two important features of CellDesigner are its SBML compliancy and SBW support. CellDesigner can only save and work with SBML files. Hence, users do not need to deal with a different file format and can transfer their work to other SBML compliant tools. Being SBW enabled and working with SBML format, CellDesigner users can integrate with all SBM enabled simulation modules. CellDesigner itself does not have a built in simulation engine but with SBW, users can browse or modify an existing model from databases and can run simulations. Most of these SBM modules are online and free [33].

SBML format is not designed to contain information about the layout and the graphical information since it is intended for more quantitative purposes rather than visual. However, CellDesigner is a visualization tool; it needs to store this information in SBML. Hence, CellDesigner has extended the SBML format to contain this necessary information in the form of annotation tags. Once saved, CellDesigner can access layout and graphical information from this tag. Nonetheless, if it has to open an SBML file which does not contain this additional information, CellDesigner has a built-in automatic layout algorithm, which creates the models and places the nodes and the edges automatically.

Other features of CellDesigner include connection to online databases, plug-in support and capability to save models as image files. Supported image formats are: PNG, JPEG, SVG, PDF, and EPI.

CellDesigner is a well developed visualization and simulation tool. It is similar to MIMTool in the sense that they are both designed for a specific graphical notation. Advantage of CellDesigner is that, it was first created in 2003 and is currently developed in collabration with The Systems Biology Institute (SBI); a non-profit research institute founded by Kitano [39].

5. MIMTooL

In this chapter, we will introduce our tool; MIMTool. It is an open source, protein-protein interaction drawing software for Kohn notation [40].

5.1. Gui

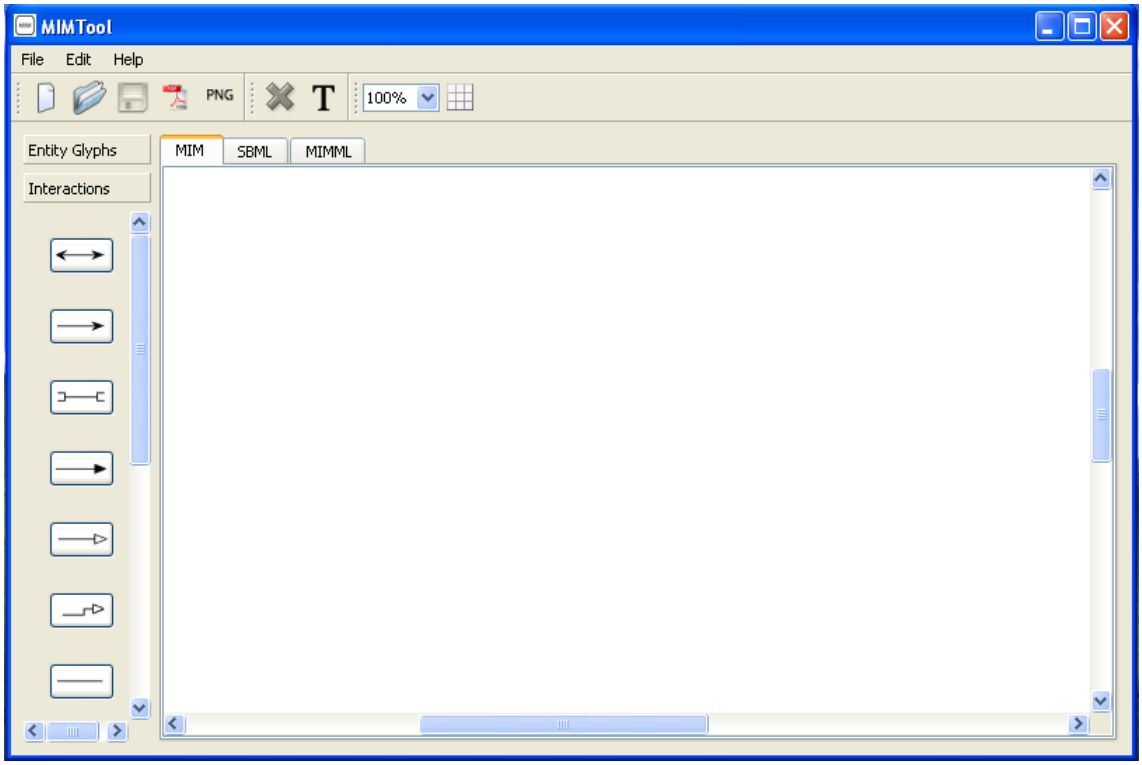

Figure 5.1. MIMTool

MIMTool uses Qt graphical user interface GUI module with the C++ language. Qt is a cross platform application and a graphical user interface framework. Its GUI module has the necessary tools to create an advanced GUI application [34]. Figure 5.1 gives a screenshot of MIMTool.

MIMTool's user interface consists of three working areas. First one is a drawing area. The next two are read-only text areas for viewing SBML and MIMML codes of the drawn maps.

On the left is a button toolbox for drawing MIM notation symbols. Every button

corresponds to a symbol in the MIM notation.

On top of the drawing area, there are toolbars that contain actions to manage MIMTool. The first three are the familiar "new", "open" and "save" actions. The next two generate images of the drawn maps in PDF and PNG format. The final four are "delete", "enter text", "zoom" and "grid" features. For more information about the functionalities of these toolbars, refer to the manual in Appendix A.

5.2. Features

The features of MIMTool can be enumarated as follows:

(i) Support for different operating systems: MIMTool can run in Windows and Linux based operating systems.

(ii) Fully interactive: MIMTool is designed to be very flexible both for drawing and modifying maps. Molecular species can be placed anywhere on a map area and they can be dragged anywhere with a mouse. Drawing interaction lines is also very flexible. They are drawn manually and can also be dragged by the mouse. When an item is dragged on the scene, all the other connected items; whether they are interactions or a molecular species; are also dragged. These movement features give user complete freedom while creating maps. It also makes it much easier to make changes if necessary.

(iii) Semi-automatic reaction interaction line drawing: When a map becomes crowded or large, it can be hard to place an interaction line with mouse. To resolve this kind of complications and fasten the drawing, we have added a semi-automatic reaction interaction line drawing feature. The algorithm draws the interaction line after two items that wish to be connected are selected. This automatic orthogonal line placement makes use of Dijkstra's shortest path algorithm where the number of bends and crossings are minimized.

(iv) MIMML: MIMTool has full support of MIMML; an XML based saving format developed by Kohn and his group. It saves layout information of MIMs.

(v) Export to SBML: MIMTool can export maps as SBML files. The ability to export SBML files increases the compatibility with many other tools that can import SBML. We used the libSBML [35] library to manage SBML files.

(vi) Code viewing: There are two tab windows besides the map drawing window. They are read only text editors. MIMML and SBML codes of maps can be viewed within the tool.

(vii) Export images: Besides XML based saving formats, MIMTool can save the MIMs as image formats for direct visualization. Currently supported formats are PDF and PNG.

5.3. Design Entities

For designing 2D items, we have used Qt's QGraphicsView framework. There are three base classes in this framework; QGraphicsScene, QGraphicsView and QGraphicsItem. QGraphicsScene stores and manages graphic items. QGraphicsView object visualizes the scene through the main application window. Finally QGraphicsItem is the base class of graphical items in the scene[33].

After studying Kohn notation and Qt's QGraphicsFramework's build-in classes, we decided to create six classes as follows:

- MainWindow
- Scene
- DiagramItem
- Edge
- propertiesDiagram
- propertiesDialog

For MIMTool, we need a scene called a Scene object to visualize the map. Additionally, we need a user interface object that holds the scene called the MainWindow object. For MIM symbols, we need two different graphics items; one for entities and one for interaction lines called DiagramItem and Edge objects.

5.3.1. MainWindow

MainWindow class is developed as the main application window. It is derived from Qt's QMainWindow class. It keeps information about the structure and visual elements of the tool and connects all other classes together.

Its first function is to create application window and help the user to manipulate the tool efficiently and easily. It holds buttons, toolbars, toolboxes and menu bars. In brief, it contains necessary components for efficient tool management. Layout of a main window is given in Figure 5.2.

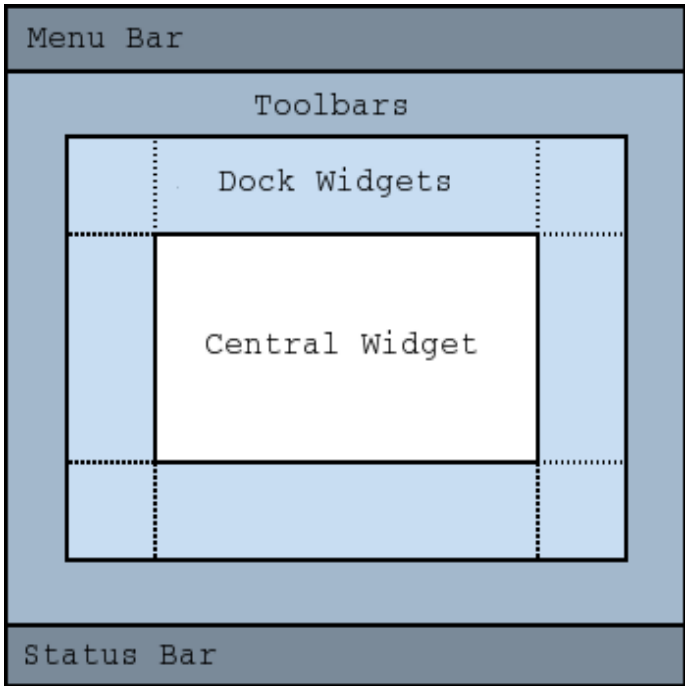

Figure 5.2. Layout of main window of Qt[33]

For MIMTool, we needed three central widgets that can be switched by tabs. One widget for viewing and drawing MIMs, and two text editor widgets for viewing SBML and MIMML codes. Central widget for MIM drawing is a QGraphicsView item. QGraphics view class visualizes the contents of the scene. The other two tabs contain read-only text boxes. We also needed a button toolbox for drawing MIM notation. Finally, we wanted to have toolbars at the top of the central widget for easy access to some frequently used actions.

MIMTool starts by creating the main window object. MainWindow object then creates the QGraphicsView object, the Scene object, and the text edit objects for code viewing. Additionally it creates menus, toolbars, toolboxes, buttons and actions. Actions are objects of a QAction class. They are part of Qt's signal and slot mechanism. These actions connect MainWindow buttons to the necessary functions. In a way, when a button is clicked, action signal is triggerred and the signal calls the connected slot. Thus when a button is clicked, it calls the connected function.

MainWindow object cannot directly access the objects in the scene. It can only manipulate its scene and what is shown through the main window. Its duties are given as follows:

(i) It creates and modifies scene and two text windows
(ii) It creates buttons, toolbars, toolboxes, menus.
(iii) It opens, saves and creates new text files for text windows.
(iv) It sets status bar.
(v) It manages all appeared dialogs.
(vi) It connects all buttons to necessary functions.

5.3.2. Scene

Scene class object creates a visual scene for MIM. This class manages all 2D items. Scene is derived from QGraphicsScene class. QGraphicsItems objecs are drawn in the scene and QGraphicsView object visualizes the scene through the main window.

Scene class has several functions. The first one is to store the graphical items, in our case the molecular entities and molecular interactions. Its second function is the management of those items. All items can be accessed and modified through the Scene class object. Features like positions, sizes, shapes, colors of graphics items can be managed by calling graphical items' member functions. Another function of the scene object is the updating of the general setting of the scene. By changing these settings, user can zoom in and out of the scene. Additionally grid lines can be added

for better organization of the objects in the scene.

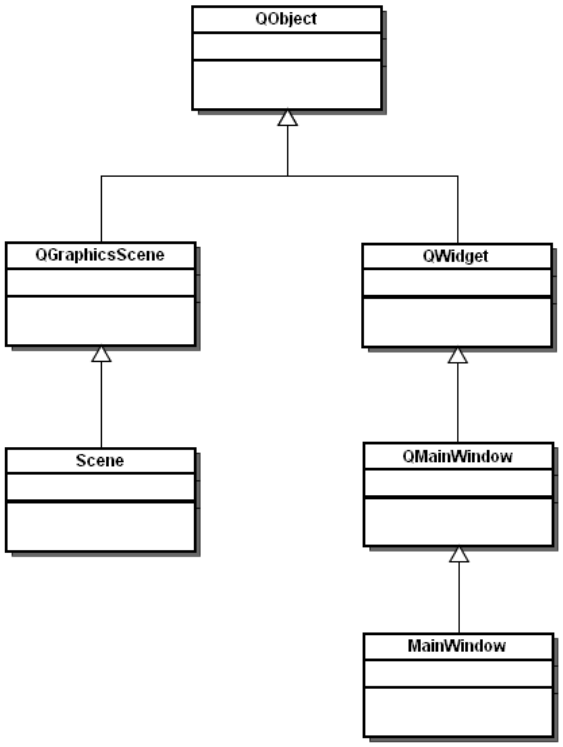

Figure 5.3. Inheritance graph of MainWindow and Scene classes

Scene object is constructed as a 2000x 2000 pixel rectangle. It stores the DiagramItem and Egde objects. These objects can be inserted, drawn and removed from the scene by the scene object. Scene object is like a bridge between the MainWindow and the objects in the scene. User gives commands through the buttons on the MainWindow object. MainWindow then calls appropriate functions of its scene object to modify the DiagramItem and Edge objects.

By managing graphical items, Scene object has some member functions that carry out important features of MIMTool. These include saving MIMML and SBML files, saving image files and drawing semi-automatic interaction lines.

Scene object also handles mouse movements like dragging, double clicking, etc.

The mouse movements can behave differently according to a feature of the Scene object called mode. Mode of scene keeps state of the scene. There are five modes; insert DiagramItem, insert Edge, insert contingency Edge, move and enter text. First three of these modes are set when the user clicks on any button on the Kohn notation toolbox. For example, if user clickes on a protein symbol, the mode of the scene is set to insert DiagramItem. Then when the user clicks a point of the scene with mouse, this signals scene to create a DiagramItem at the selected point in the scene.

After an item has been drawn into the scene, scene object signals that an item has been inserted and the mode of the scene is updated. It is then set to move mode. In the move mode, mouse can drag objects on the scene. Finally the scene is set to enter text mode when user clicks enter text button. Then, when the user clicks a point of the scene, a text box is inserted.

5.3.3. DiagramItem

MIM notation symbols can be broadly partitioned into two classes. First group is molecular entities. These can be considered as nodes of the molecular interaction graphs.

DiagramItem class is created to represent molecular entities in MIM notation. It is derived from QGraphicsPolygonItem class of Qt which is derived from QGraphicsItem.

There are six types of molecular interactions; simple physical entity, entity feature, conceptual entity, modifier, source/sink, restricted copy dot and complex dot [6]. These types have very similar properties. They are defined as enum types in diagramItem class. The most noticeable difference between them is their shapes. List of molecular species in MIMTool and their symbols are given in Figure 5.4

Shapes of diagramItem objects are drawn into the scene by paint member function. Paint function is a built-in function of the QGraphicsItem object. This function

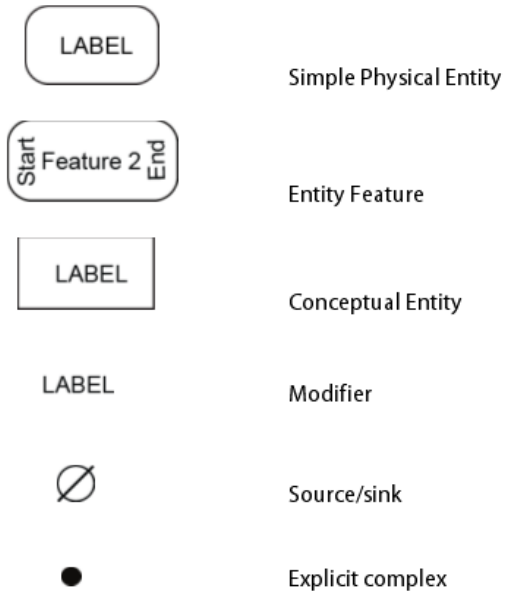

Figure 5.4. Molecular Species symbols of Kohn notation as stated in [6]

is continuously called until the program is exited. It contains information about the 2D shape of the item. Therefore, the appereance of an item can be modified or repainted by calling this function.

When a diagramItem object is right clicked, a context menu appears. From this menu, the users make some changes on the shape of the item. Context menu is actually part of MainWindow object because it is part of the user interface. When an action is clicked on that menu, appropriate Scene and DiagramItem member functions are called. In the context menu, there are four different actions:

(i) Delete : removes that item from the scene.

(ii) selectAll : selects all items in its scene.

(iii) Filter the interactions/ Unfilter the interactions: updates colors of interaction lines connected to that DiagramItem object. This action updates the color of Edge objects connected to that DiagramItem object. Connected edges are stored in vector objects as a data member of DiagramItem.

(iv) Properties: a properties dialog appears. This dialog is a propertiesDiagram

object. From this dialog, DiagramItem objects' positions, width and height values can be updated.

As mentioned earlier, DiagramItem objects can be considered as nodes of molecular interaction graph. Its most important feature is its ability to connect to molecular interaction lines. A diagramItem object keeps a list of interactions that is connected to it. This is needed because every connected interaction has to move if the molecular species move. Therefore diagramItem object has a signal called itemChange. When the item goes through any kind of change; like a move or a change in size; then the connected interactions should also adapt to this change.

5.3.4. Edge

Edge class object represents molecular interactions in MIM notation. It is derived from Qt's QGraphicsPathItem class. It can be viewed as lines or edges of molecular interactions maps.

Like with DiagramItem, Edge class has edge type enum to classify different molecular interaction symbols. Type attribute is defined as enumarated because the only difference between different types of edge objects is their appearance. All the other actions and attributes are the same.

Since molecular interactions are of QGraphicsPathItem base class, they are kept as a path in the scene. However they need to be orthogonal (MIM notation necessity). Additionally they should move as diagramItems since they are connected to movements. Finally they should change color and adapt to changes of size and shape of diagramItems.

Edge type is similar to DiagramType. Edge class has twelve enumarated types. To place an edge into the scene, there has to be at least two different items on the scene. For reactions, the first one of the molecular species is clicked and then bending points of the edge is clicked. Finally, path drawing stops when another molecular species

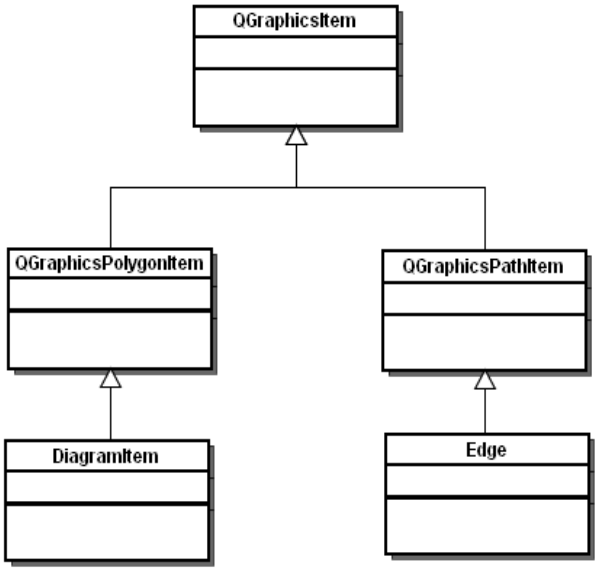

Figure 5.5. Inheritance of DiagramItem and Edge classes

is clicked. For contingencies, interactions between a molecular species and another interaction line, drawing stops when another edge is clicked. After it has been drawn, edge can be moved by dragging with mouse. While moving, edge's path lengthens or shortens. Connections with other objects are not cut.

The two selected items on the scene are stored as startItem and endItem of an edge. They keep information about the molecular species or interaction line that it is connected to. Edge object is actually a path; i.e., a collection of connected lines. Since Kohn notation implies that the interactions lines must be orthogonal, each edge is made up of vertical and horizontal lines. Also, since edges must follow the objects that it is connected to, that path must be dynamic. It means that path must change direction, shortened or lengthen. Therefore instead of keeping separate line information, we keep the bending points of the orthogonal line. Points are kept as points object and are stored in a vector. Figure 5.6 shows an example of an orthogonal path and its bending

points.

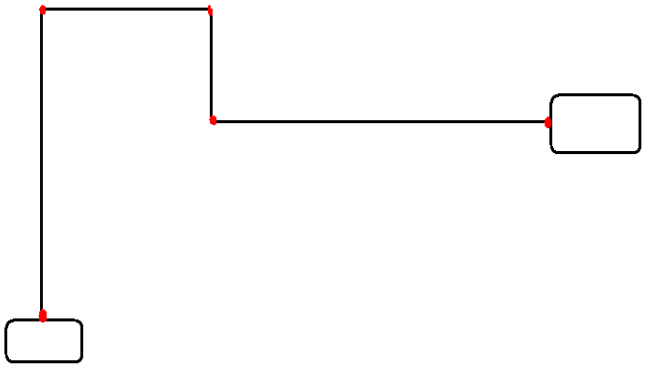

Figure 5.6. Points of a Edge object

In MIMTool, connections cannot be disturbed by moving objects. When any object moves in the scene, all the connected objects also move accordingly. Interactions between any two object is not cut with movement. Therefore, if start or end item of an edge moves, the edge also moves accordingly. This is established by using the QGraphicsItem objects ItemHasChanged signal. When an item changes its shape or moves, it sends a signal and all the connected edges that receive the signal are redrawn according to the change.

Like the DiagramItem, Edge has a property that can be changed by context menu. Its color can be updated. By default, all the edges are black. According to Kohn notation, edges can have a different color according to the behavior of their interaction. Therefore, a properties dialog appears with a color palate to update the color of an edge at any time.

5.3.5. propertiesDiagram

Qt GUI framework provides classes that create dialog objects. The base class of these classes is QDialog class. We created propertiesDiagram object as a properties dialog to edit the properties of a diagramItem object. propertiesDiagram class is

inherited from QDialog class and is created by QDesigner. It has buttons for changing the position, width and height of the item. An example dialog box is given in Figure 5.7.

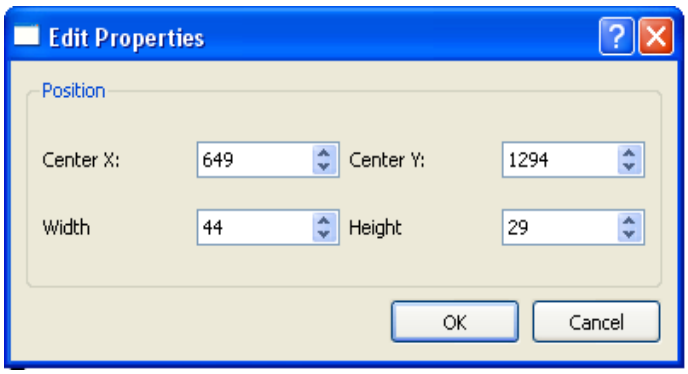

Figure 5.7. Properties Diagram object

5.3.6. propertiesDialog

propertiesDialog object is very similar to propertiesDiagram object. It is derived from Qt's QDialog class and we have used QDesigner to create it. It appears when properties action is clicked from edge object's context menu. It is created to change the color of an edge object.

Class diagram of MIMTool is given in Figure 5.8. All the member functions and data members of classes are listed in Appendix B.

5.3.7. Semi-automatic drawing algorithm

We wrote a semi-automatic drawing algorithm using Dijktra's shortest path algorithm. After two DiagramItem objects (molecular species) are selected on the scene, if an interaction button is clicked, MIMTool runs the semi-automatic drawing algorithm and draws the shortest path between the two molecular species with minimum number of bends and crossings.

5.3.7.1. Nodes and edges. First we had to adapt MIMs to the Dijkstra's algorithm. We had to decide what will be the nodes and paths used for the algorithm between

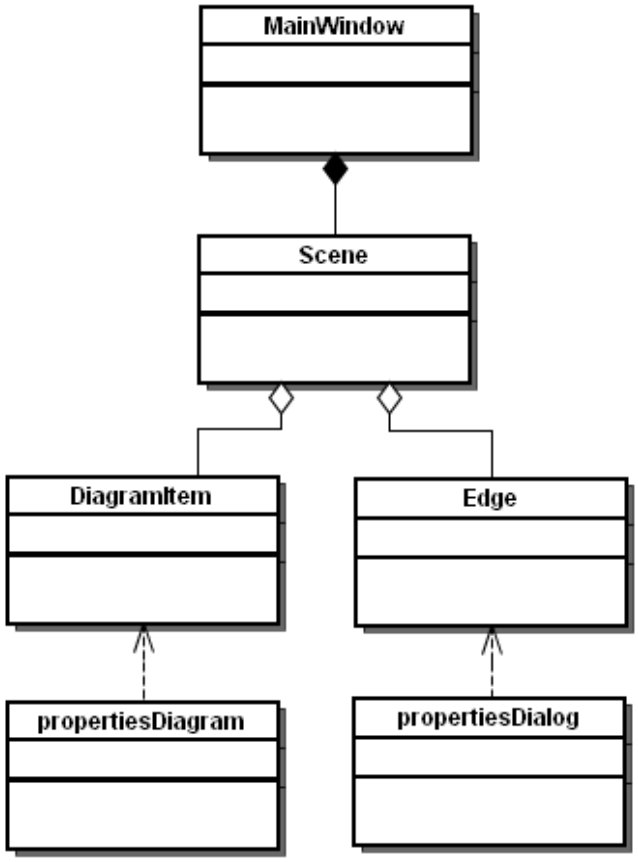

Figure 5.8. Class diagram of MIMTool

the two selected nodes. The reason is, we want to find the shortest path that is not already in the scene. In another words we want to draw a new line and not use the existing interaction lines. Therefore, we have to define paths in empty areas that the algorithm can select.

We decided to place the selected nodes on an imaginary grid of size 5x5. Grid drawing is suitable because it supports orthogonal drawing and it forms the needed structure for the algorithm. We assume there are nodes on the intersection points and the paths are the grid lines. The algorithm has to select one of the paths on the grid. Example 5x5 grid between proteins D and F with blue imaginary nodes is given in

Figure 5.9.

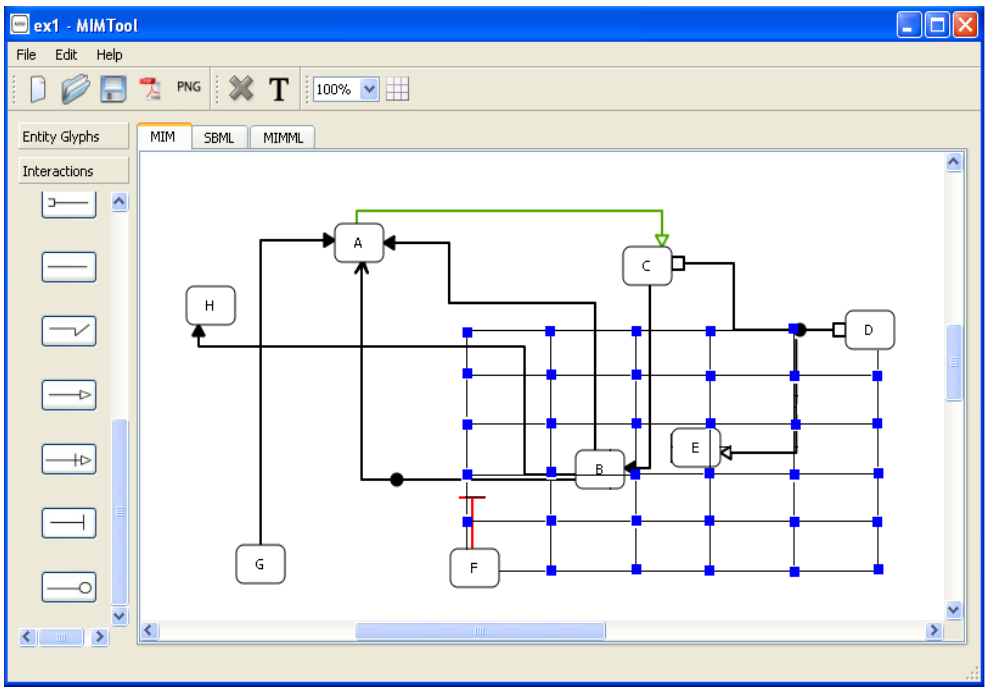

Figure 5.9. Screenshot of MIMTool with imaginary grid

The size of the grid is chosen as 5x5, meaning a total of 36 nodes. While choosing the grid size, we take into account the running time of the algorithm and aesthetic appeal. Table 5.1 lists the running times of different grid sizes on a more complicated example on the same map; finding path between A and B. If we increase the number of grids, then the algorithm runs slower. When we set it to 10x10, there are 121 nodes and algorithm becomes more complex. Running time is over 1 second and this is unappealing to the user. Additionally, when there are more nodes and path, the algorithm can avoid other items in the scene more easily. However, to avoid these items, the path starts to bend, hence the total number of bends is increased. If we decrease the grid size, the algorithm runs faster but it cannot avoid most of the items. In a 3x3 grid, it becomes harder to avoid other items in the scene. Therefore we have chosen 5x5 grids. The algorithm always runs in less than half a second and it can chose the shortest path from 60 line segments.

We also have to take into account the other proteins and interaction lines on the map. To create visually aesthetic maps, the new shortest path should not cross other interaction lines or other proteins. In Kohn notation, since the degree of a node can

Table 5.1. Running time with different grid sizes

	Running time (sec)
3x3	0.18
4x4	0.29
5x5	0.48
6x6	0.73
8x8	0.92
10x10	1.9

exceed four, crossings of interaction lines cannot be avoided. It can only be minimized. On the other hand crossing of an interaction line and a molecular species is forbidden by the notation due to the fact that it can look as if the interaction line is coming out of that molecular species. This problem can be solved by length assignment for aesthetics appeal.

5.3.7.2. Aesthetic considerations. Another issue we had to decide was the length of a line on the grid. For this decision, we had to take into account our aesthetics constraints; minimization of total number of crossings and minimization of total number of bends. We decided that a length of a line between imaginary nodes should be 5 if that line does not cross any item. If it crosses an interaction line, the length of the path increases by 20. If it crosses a molecular species of any kind, then the length will be increased by 2000. Assignment of such a big number can lower the chance of selection of that line as part of the shortest path.

To test if selection of the line creates a bending, we have to check the previous line. In Dijkstra's algorithm, we store information about the previous nodes. Thus we conduct a test of bending. If the line bends, then length is increased by 20, the same as crossing.

5.3.7.3. Speed-up techniques. After we have decided our nodes, lines and length values, we applied two speed-up techniques; heap implementation and bidirectional search method.

We used heap data structure to store the molecular species (nodes) of the graph. Heap structure allows us to perform operations that are used by Dijktras's algorithm faster. The operations we have implemented are given below.

- Create heap
- find-min
- insert
- delete-min
- decrease-key

Pseudo code of our algorithm with binary heap is given in Figure 5.10.

```
G=(N,E)
s = start node
d(i) = distance of edge i from start node
c(i,j) = cost of the edge between nodes i and j
cross(j) = checks a line to node j would cross any other object in the scene
bend(j) = checks if line to node j causes bend
A(n) = nodes that are orthogonally adjacent to node n

begin
   Create-heap(H);
   A = ø ;
   d(i) =  ∞ for all i element of N;
   d(s) = 0;
   pred(s) = 0;
   insert(s,H);
   while H != ø do
      find-min(i,H);
      delete-min(i,H);
      for each (i,j) element of A(n) do
         value : d(i) + c(i,j);
         if ( cross(j) )
            cost += 20;
          if ( bend(j) )
            cost += 20;
             if ( d(j) > value ) then
                if ( d(j) = ∞ then d(j) = value, pred(j) = i, and insert(j,H)
                else set d(j) = value, pred(j) = i, and decrease-key(value, i, H);
      end;
   end;
end;
```

Figure 5.10. Our modified Dijkstra's Algorithm

Also we know the starting node and the end node (selected molecular species).

Thus our algorithm is bidirectional and we can improve the algorithm using bidirectional search method. The algorithm starts from both the start node and the end node. The algorithm stops when the distance of a node is calculated at both sides.

5.3.7.4. Example. In this last section we give a simple example of how our algorithm works. We want to draw a non-covalent modification arrow between nodes A and B in Figure 5.11.

First, the algorithm calculates the imaginary grid. From the coordinates of A and B, we draw equally spaced 5x5 grid in Figure 5.11. Algorithm assumes there are nodes on the intersection points. They are denoted by blue squares. Then the algorithm starts from node A. After first iteration, backward algorithm starts from B. It alternates between A and B until a node is visited by both sides; the only blue node Figure 5.12. Calculated edges are also given in the same figure. Finally the algorithm draws the path with the shortest lenght with minimum bends and minimum crossings. Final graph is given in Figure 5.13.

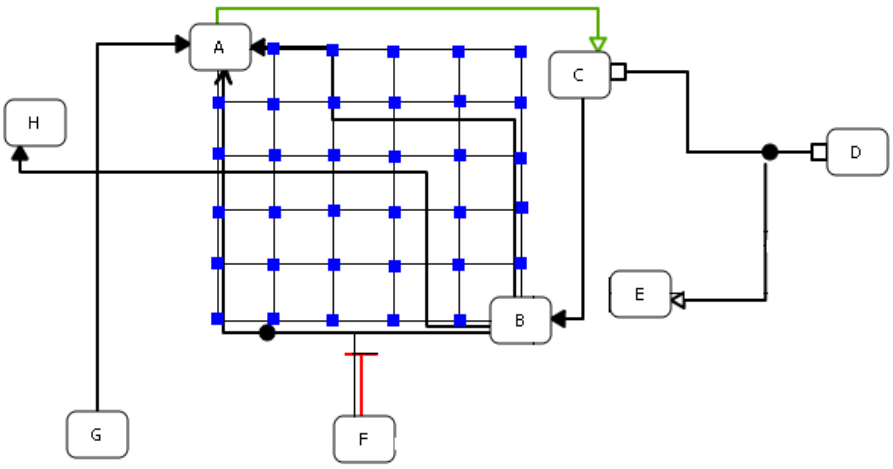

Figure 5.11. MIM with imaginary grid

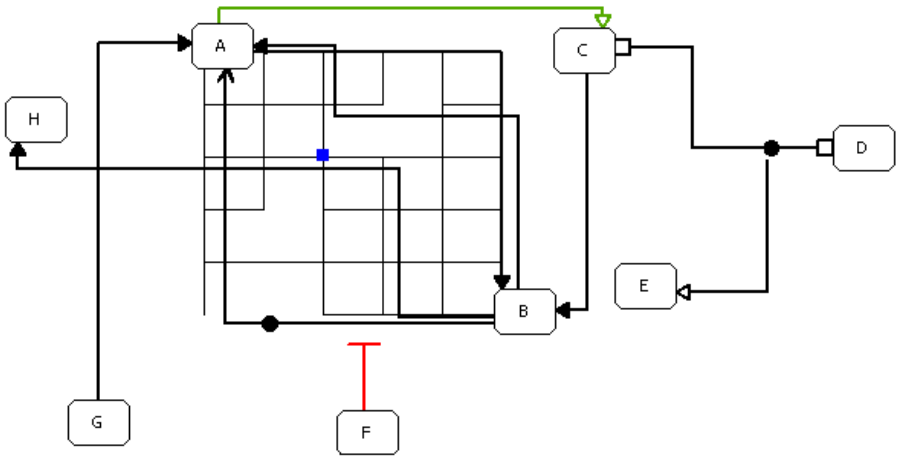

Figure 5.12. MIM showing visited lines

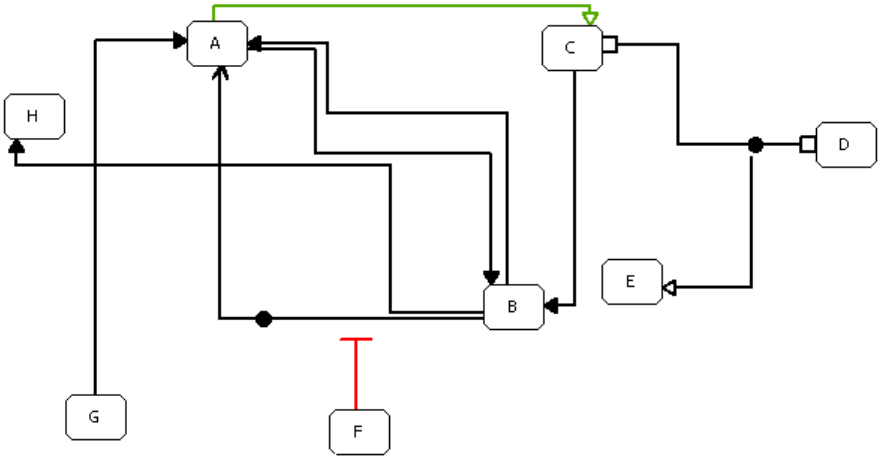

Figure 5.13. Final MIM

6. EXAMPLES

6.1. Example 1

In this section we draw interaction lines using our algorithm on some MIMs. First we will give original map with two selected molecular species. Then we will show the map after the algorithm insert the new interaction line.

In the first example given in Figure 6.1, we want to draw a reaction line between molecular species E and B. These species are drawn using dashed lines because they are selected.

It is observed that there is no empty area between E and B. In cases like this, the algorithm has to select the path with minimum crossings and bends. Figure 6.2 shows the drawn path with one crossing and one bending.

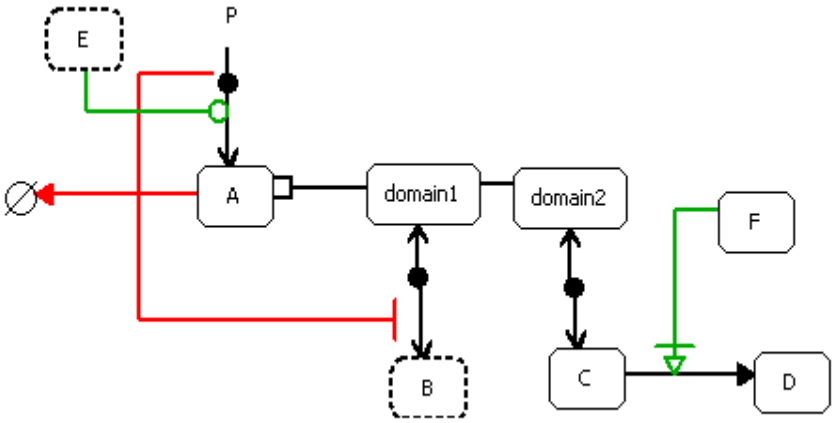

Figure 6.1. Example MIM before algorithm

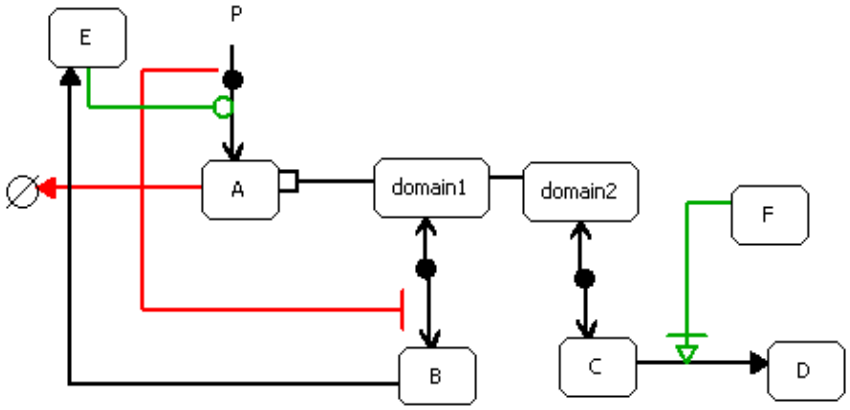

Figure 6.2. MIM after the insertion of stoichiometric conversion

6.2. Example 2

We have selected two molecular species ATM and Mdm2 as shown in Figure 6.3. Like in example 1, there is no empty space between the two species.

In this case the path has to bend twice because protein p53. If the algorithm would have drawn a path like in example 1, the path would go under p53 which could give the impression that the path is coming out of p53. To avoid this, path bends twice and goes over p53. The algorithm avoids crossings by going just above p53 so that total number of crossing is minimized. Final interaction is shown in Figure 6.4.

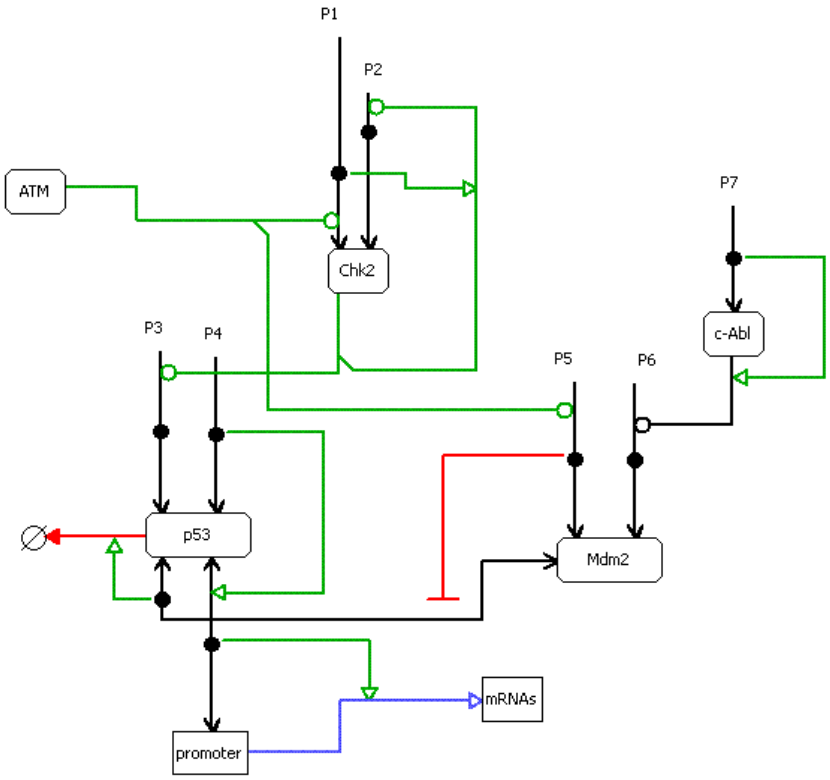

Figure 6.3. Example MIM before algorithm

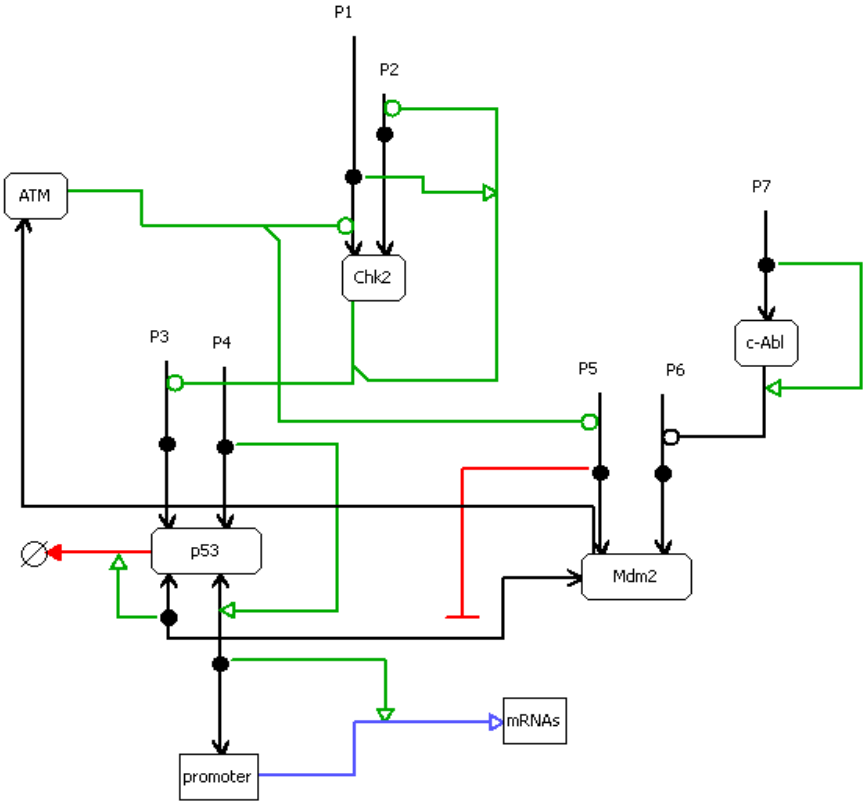

Figure 6.4. MIM after the insertion of covalent binding

7. CONCLUSION

7.1. Overview

In this thesis, we developed MIMTool for drawing protein interaction maps in Kohn notation. We added features that would allow the user to have the freedom to draw and manupulate MIMs. In this context, we also added a semi-automatic drawing algorithm that calculates shortest path between two molecular species.

One of our aims for developing MIM tool was to encourage other tools to support MIM notation and MIMML. We believe that with the development of MIMTool, MIM notation is closer to becoming a standard for protein-protein interaction visualization.

MIMTool is a state of-the-art drawing tool for MIMs. It introduces unique features such as object dragging and line drawing with automatic bending and crossover minimization, which are not available in most diagram editors. MIMTool helps fill the lack of software tools that was available for the MIM notation by contributing a much needed diagram editor. MIMTool should facilitate wider adoption of the MIM notation.

7.2. Future Work

In future version of MIMTool, some of the basic functionality can be improved; adding additional properties for molecular species and interactions, extra buttons for scene manipulation, etc. In addition, full SBML support can de added in order to increase data exchange between other tools and databases. Secondly, automatic orthogonal drawing algorithm can be implemented. This algorithm can be used to automate the map drawing process by producing aesthetic maps.

APPENDIX A: MIMTOOL MANUAL

A.1. How To Download MIMTool

MIMTool is free and available at http://code.google.com/p/mimtool/

A.2. How To Start MIMTool

Once downloaded successfully, MIMTool can be started by

- double clicking on the shotcut on the desktop
- double clicking on the MIMTool exe in the MIMTool directory
- from Start Menu, click on All Programs, then MIMTool, then MIMTool.

A.3. Overview of The Tool

MIMTool has three main working areas, one button toolbox on the left, and four toolbars at the top.

A.3.1. Working Areas

MIMTool has one drawing area; MIM; and two read-only text viewing areas; SBML and MIMML. When the program is started, drawing area is opened by default. Working area can be changed by switching between tabs.

A.3.2. Buttons

On the left of the working area, there is a toolbox. This toolbox is for drawing MIMs and can be used only in drawing area. Each button on this toolbox represents a symbol from MIM Notation.

A.3.3. Toolbars

On top of the working area, there are three toolbars; file toolbar, edit toolbar and view toolbar. Actions on the toolbar are: *New, Open, Save, Save as PDF, Save as PNG, Delete, Enter Text, Zoom* and *Grid*.

Figure A.1. Toolbars

A.4. Creating MIM

When MIMTool is started, MIM drawing area is opened by default. Buttons on the left toolbox is used to draw MIMs. Roughly, Entity Glyphs are nodes and Interactions are edges of MIMs.

A.4.1. Entity Glyphs

Entity Glyphs are molecular species with a physical structure or a concept; protein, pathways, etc.

A.4.2. Drawing Entity Glyphs

To place an entity glyph into the scene, click on the entity glyph button you want to draw, then click on a point in the drawing area. The entity glyph will be drawn at that point.

A.4.3. Features of Entity Glyphs

When an entity glyph is right clicked, a context menu appears. This menu differs according to the entity glyph type.

Still, there are common context menu actions: *Delete,Filter the interactions.*

Besides these, simple physical entity, entity feature and conceptual entity has *Properties* action.

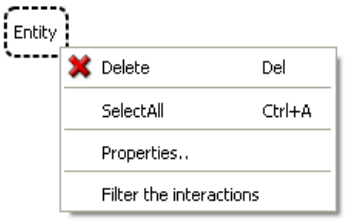

Figure A.2. Context Menu

A.4.3.1. Delete. Delete action deletes the selected item. It is also located on the edit toolbar.

A.4.3.2. Filter the Interactions. This action higlights the interactions of that item in the map.

A.4.3.3. Properties. When properties action is clicked on, a properties window appears where user can change the position and size of that graphics item.

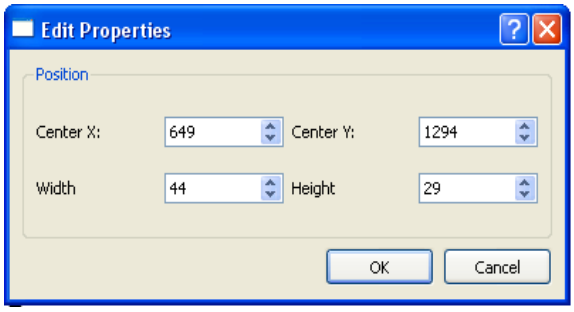

Figure A.3. Properties Window

A.4.4. Interactions

Interactions line represent relation between two entities or between an entity and an interaction.

A.4.5. Drawing Interactions

After clicking on an interaction button, a starting point should be clicked on the drawing scene. Since interactions must start from an entity or an interaction, that point should be on an entity or an interaction. Else, line cannot be drawn.

After selecting an initial point on an object, interaction line starts following the cursor orthogonally. When clicked on another point, that point becomes a bending point of that line. There is no limit on the number of bends an interaction line can have. Since an entity must be connected to other entities and other interactions, when an entity or another interaction line is clicked on, drawing ends.

A.4.6. Features of Interactions

When an interaction line is right clicked, a context menu appears. This menu differs according to the interaction type. There are also five context menu actions that are common to all interactions: *Delete, Properties, Add Intermolecular Glyph, Bring to Front* and *Send to back*. Interactions that connect two entities, also have another action called *Reverse the interaction direction*. An interaction starting from an interaction in addition has two other actions called *Add Right Branch* and *Add Left Branch*.

A.4.6.1. Delete. *Delete* action deletes the selected item. It is also located on the edit toolbar. Its shortcut is Ctrl+Del.

A.4.6.2. Properties. When properties action is clicked on, a properties window appears where the user can change the color of the interaction line.

A.4.6.3. Add Intermolecular Glyph. This action adds the intermolecular glyph symbol on the interaction line. The symbol is movable on the line and can be deleted by selecting an item and clicking on the delete button. When the symbol is added to the line, this action is removed from context menu.

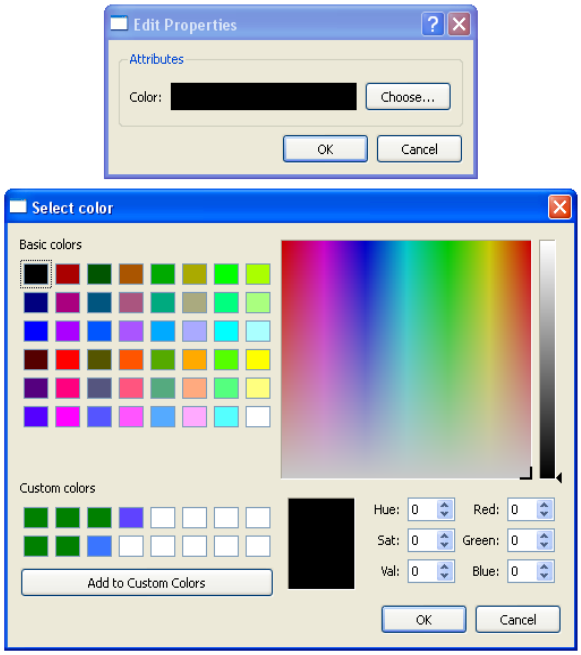

Figure A.4. Properties of Interactions

A.4.6.4. Bring to Front. By default entity glyphs are on top of interactions. By clicking on the *Bring to Front* action, the selected interaction line becomes in front of the entites.

A.4.6.5. Send to Back. This action changes back to default and places the selected interaction behind entity glyphs.

A.4.6.6. Reverse the interaction Direction. Since automatic drawing selects the direction of the interaction randomly, this action is created to reverse the direction if necessary. This action does not apply to interactions that connect entities with other ineractions.

A.4.6.7. Add Right Branch. This action only appears in context menu if interaction starts from another interaction. When this action is clicked, a right branching symbol is added at the beginning of interaction line. This implies starting interaction branches to this interaction.

A.4.6.8. Add Left Branch. This action only appears in context menu if interaction starts from another interaction. When this action is clicked, a left branching symbol is added at the beginning of interaction line. This implies starting interaction branches to this interaction.

A.5. Additional Features

A.5.1. Entering Text

Entering text feature can be used to enter text anywhere in MIMs, like numbering interactions. Text button is shown in Figure A.5.

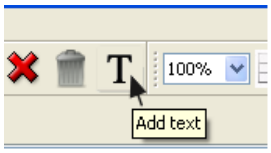

Figure A.5. Text button

A.5.2. Grid

By clicking on grid button, grid is added to the backrgound of the map. Grid is removed if the button is clicked again.

A.5.3. Zoom

On the view toolbar, there is a scale combo box. The scaling values on the combo box represents scaling of the scene. 100 being default scale value, increasing scale value zoomes into the scene while decreasing zoomes out of the scene as shown in Figure A.6.

A.6. Viewing SBML and MIMML

Besides the MIM tab, there are two other tabs; SBML and MIMML. These are text windows. When a file is opened, the MIM map is drawn in the drawing window,

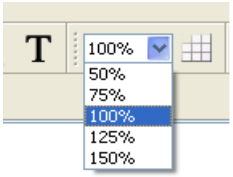

Figure A.6. Scale Combo Box for Zooming

XML based MIMML code is viewed in the MIMML tab and SBML code is generated and viewed in SBML tab. These text windows are read only. Therefore user cannot make any modifications. However if map is modified, saved and reopened, codes are updated in the text windows.

A.7. Saving MIM

There are a couple of different saving formats. MIMs can be saved as an image file or an XML based SBML or MIMML format. For now, it should be noted that MIMTool supports partial implementation of MIMML.

To save a file, click on save button after drawing the map or click on "File" menu and then "Save" or "Save As".

A.7.1. Opening and saving MIMML

MIMTool has partial implementation of MIMML [6]; an XML based format developed by Dr Kohn and his group. When MIM is drawn, to save as MIMML, click on save and enter a file name and choose a directory.

To open a MIMML file, click on open button from toolbar or click on Open from file menu. Open dialog appears to choose the directory of the file. Then click Open. Once clicked, the map is drawn in the MIM tab and the SBML and MIMML codes are viewed on the other tabs.

A.7.2. Save SBML

MIMTool can save MIMs as SBML files. To save them as SBML, go to "File" menu and click "Save as SBML Document". Program will ask for file name and directory to save the file.

A.7.3. Save Images

MIMs can be saved as PNG or PDF files by clicking on the PDF or PNG button on the edit toolbar. When the button is clicked, a File Dialog appears to get the name and directory of the file as shown in Figure A.7.

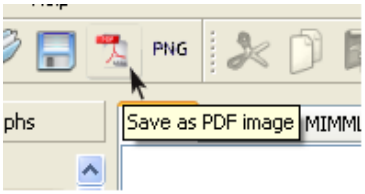

Figure A.7. PDF and PNG save buttons

APPENDIX B: DATA MEMBERS AND MEMBER FUNCTIONS OF CLASSES

B.1. MainWindow

Data Members:

- Scene *scene
- QMenu *fileMenu
- QMenu *editMenu
- QMenu *helpMenu
- QAction *newAction
- QAction *openAction
- QAction *saveAction
- QAction *saveAsAction
- QAction *aboutAction
- QAction *exitAction
- QAction *importAction
- QAction *exportAction
- QAction *deleteAction
- QAction *propertiesAction
- QAction *inTransAction
- QAction *printAction
- QAction *printPNGAction
- QAction *gridAction
- QAction *numberingAction
- QAction *filteringAction
- QAction *unfilteringAction
- QAction *reverseAction
- QAction *bringToFrontAction

- QAction *sendToBackAction
- QAction *sbmlAction;
- QAction *branchRightAction;
- QAction *branchLeftAction;
- QAction *selectAllAction;
- QToolBar *fileToolBar
- QToolBar *editToolBar
- QToolBar *viewToolbar
- QSpacerItem *space1
- QSpacerItem *space2
- QSpacerItem *space3
- QTabWidget *tab
- QTextEdit *SBMLText
- QTextEdit *MIMMLText
- QToolBox *toolBox
- QGraphicsView *view
- QButtonGroup *buttonGroup
- QComboBox *sceneScaleCombo
- QGraphicsTextItem* addNumber
- QString curFile
- QFileInfo mainFile
- QPrinter *printer
- QPainter *painter

Member Functions:

- MainWindow();
- void createActions();
- void createMenus();
- void createToolBar();
- void createToolBox();
- void loadFile(const QString &fileName, const int &index);

- void setCurrentFile(const QString &fileName);
- QString strippedName(const QString &fullFileName);
- bool maybeSave();
- bool saveFile(const QString &fileName);
- QWidget *createCellWidget(const QString &text, DiagramItem::DiagramType type);
- QWidget *createCellWidget(const QString &text, Edge::EdgeType type);
- void deleteFunc(QGraphicsItem *item1);
- void buttonGroupClicked(int id);
- void newFile();
- void open();
- bool save();
- bool saveAs();
- void about();
- void deleteItem();
- void itemInserted(DiagramItem *item);
- void EdgeInserted(Edge *item);
- void updateActions();
- void properties();
- void inTrans();
- DiagramItem *selectedDiagramItem() const;
- Edge *selectedEdge() const;
- void print();
- void printPNG();
- void grid();
- void numbering();
- void filtering();
- void unfiltering();
- void reverse();
- void bringToFront();
- void sendToBack();
- void documentWasModified();

- void sceneScaleChanged(const QString &scale);
- void saveSBML();
- void addRightBranching();
- void addLeftBranching();
- void selectAll();

B.2. Scene

Data Members:

- enum Mode Insert, Move, InsertEdge, InsertConEdge, Text
- Mode myMode
- QVector<QPointF>points
- QList<Edge*>edges
- QList<DiagramItem*>diagramItems
- QList<QString>ids
- bool grid
- QGraphicsLineItem *line
- DiagramItem::DiagramType myItemType
- Edge::EdgeType myArrowType
- QMenu *myItemMenu
- QMap<QString, int>Letter
- QGraphicsTextItem *addNumber
- QFont myFont
- QColor myTextColor
- QColor myItemColor
- QColor myLineColor
- int costArray[2000][2000]
- QPointF prevArray[2000][2000]
- QVector<QPointF>heap
- SBMLDocument* sbmlDoc

Member Functions:

- Scene(QObject *parent = 0);
- QFont font() const return myFont;
- QColor textColor() const return myTextColor;
- QColor itemColor() const return myItemColor;
- QColor lineColor() const return myLineColor;
- void updatePoints(bool t);
- DiagramItem* returnItem(QString id);
- Edge* returnPath(QString id);
- void filtering();
- void unfiltering();
- QString saveToXML();
- void loadFromXml(QIODevice *xmlData);
- void loadFromXmlInteraction(QIODevice *xmlData);
- void drawSemiAuto(QList<QGraphicsItem *>items);
- bool validateSBML (SBMLDocument* sbmlDoc);
- bool writeExampleSBML(const SBMLDocument* sbmlDoc, const QString & filename);
- SBMLDocument* createExampleEnzymaticReaction(const QString &modelName);
- void saveToSBML(const QString &fileName);
- char* writeExampleSBML2(const SBMLDocument* sbmlDoc);
- char* saveToSBML2();
- bool cross(const QPointF &p);
- void insertHeap(const QPointF &p);
- void decreaseKeyHeap(const QPointF &p);
- void deleteMinHeap();
- bool generateToAutoLayout();
- void addBranch(bool right);
- void setMode(Mode mode);
- void setItemType(DiagramItem::DiagramType type);
- void setArrowType(Edge::EdgeType type);

- void diagramItemChange(DiagramItem *item);
- void itemInserted(DiagramItem *item);
- void EdgeInserted(Edge *item);
- void itemSelected(QGraphicsItem *item);
- void mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseMoveEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseReleaseEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void drawBackground (QPainter * painter, const QRectF & rect) ;

B.3. DiagramItem

Data Members:

- enum DiagramType {protein, domain, Small, DNA, sourceSink, Dot, Complex, Trans, Invisible }
- Edge *host
- QVector <Edge*>edges
- double per
- QPolygonF myPolygon
- QString myId
- bool filtered
- int width, height
- QString myText
- DiagramType myDiagramType

Member Functions:

- DiagramItem(DiagramType diagramType, QGraphicsItem *parent = 0, QGraphicsScene *scene = 0);
- void addEdge(Edge *arrow);
- QRectF outlineRect() const;

- int roundness(double size) const;
- void setText(const QString &text);
- void setWidthHeight(const int &w, const int & h);
- QString text() const;
- QRectF boundingRect() const;
- DiagramType diagramType() const
- void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event);
- void mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseMoveEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseReleaseEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void DiagramItem::paint(QPainter *painter,const QStyleOptionGraphicsItem *option,QWidget * /* widget */);
- QVariant itemChange(GraphicsItemChange change,const QVariant &value);
- QPainterPath shape() const;
- void sizeChange(const qreal &r1, const qreal &r2);
- QString returnEnum(const int &num);

B.4. Edge

Data Members:

- enum EdgeType {nonCovalent, covalentMo, covalentBond, stoichiometric, loss-Reactant, transcription,combination, firstFeature, nextFeature, cleavage, stimulation, requirement, inhibition, catalysis }
- QVector <QPointF>points
- QList <Edge*>conEdges
- QList <DiagramItem*>invisibleItems
- DiagramItem *complex
- DiagramItem *trans
- bool isComplex
- bool intrans
- qreal startx, starty, endx, endy

- QString myId
- bool endUp
- bool startUp
- bool branchRight
- bool branchLeft
- DiagramItem *myEndItem
- DiagramItem *myStartItem
- EdgeType myEdgeType
- QColor myColor
- QColor myOldColor
- QPolygonF arrowHead, arrowHead2

Member Functions:

- Edge(DiagramItem *startItem, DiagramItem *endItem, EdgeType edgeType, QGraphicsItem *parent = 0, QGraphicsScene *scene = 0);
- void addEdge(Edge *arrow);
- int type() const return Type;
- QColor Color() const;
- EdgeType edgeType() const
- void setColor(QColor c);
- void setOldColor();
- QPainterPath shape() const;
- void mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseMoveEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void mouseReleaseEvent(QGraphicsSceneMouseEvent *mouseEvent);
- void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget = 0);
- QPointF intersectStartItem();
- QPointF intersect();
- QVector<QPointF>trackDiagramItem();
- bool containPoint(const QLineF &line,const QPointF &p);

- void change();
- int distance(const QPointF &p1,const QPointF &p2);
- void reverse();
- QString returnEnumFirst(const int &num);
- QString returnEnumSecond(const int &num);
- DiagramItem *startItem() const
- DiagramItem *endItem() const

B.5. propertiesDiagram

Data Member:

- DiagramItem *item

Member Functions:

- PropertiesDiagram2(DiagramItem *item, QWidget *parent = 0);
- void on_buttonBox_accepted();

B.6. propertiesDialog

Data Members:

- Edge *item
- QColor Color

Member Functions:

- PropertiesDialog(Edge *item, QWidget *parent = 0);
- void on_buttonBo_accepted();
- void on_colorButton_clicked();

- void updateColorLabel(QLabel *label, const QColor & color);
- void chooseColor(QLabel *label, QColor *color);

REFERENCES

1. Pirson, I., N. Fortemaison, C. Jacobs, S. Dremier, J. E. Dumont, and C. Maenhaut, "The visual display of regulatory information and networks", *Trends Cell Biology, Vol. 10, No. 10*, pp. 404-408, 2000.

2. Cook, D. L., J. F. Farley, and S. J. Tapscott, "A basis for a visual language for describing, archiving and analyzing functional models of complex biological systems", *Genome Biol. 2, RESEARCH0012*, 2001.

3. Kitano, H., "A graphical notation for biochemical networks", *Biosilico 1*, pp. 169-176, 2003.

4. Kurata, H., N. Matoba, and N. Shimizu, "CADLIVE for constructing a large-scale biochemical network based on a simulation-directed notation and its application to yeast cell cycle", *Nucleic Acids Res. 31*, pp. 4071-4084, 2003.

5. Kohn, K. W., M. I. Aladjem, J. N. Weinstein, and Y. Pommier, "Molecular interaction maps of bioregulatory networks: a general rubric for systems biology", *Molecular biology of the cell, Vol. 17, No. 1*, pp. 1-13, 2006.

6. "MIMML Schema", http://discover.nci.nih.gov/mim/, 2011.

7. Le Novre, N., M. Hucka, H. Mi, S. Moodie, F. Schreiber, A. Sorokin, E. Demir, K. Wegner, M. I. Aladjem, S. M. Wimalaratne, F. T. Bergman, R. Gauges, P. Ghazal, H. Kawaji, L. Li, Y. Matsuoka, A. Villger, S. E. Boyd, L. Calzone, M. Courtot, U. Dogrusoz, T. C. Freeman, A. Funahashi, S. Ghosh, A. Jouraku, S. Kim, F. Kolpakov, A. Luna, S. Sahle, E. Schmidt, S. Watterson, G. Wu, I. Goryanin, D. B. Kell, C. Sander, H. Sauro, J. L. Snoep, K. Kohn, H. Kitano, "The Systems Biology Graphical Notation", *Nat Biotechnol, Vol. 27, No. 8*, pp. 735-41, 2009.

8. Kohn, K. W., M. Aladjem, S. Kim, J. N. Weinstein, and Y. Pommier, "Depicting

combinatorial complexity with the molecular interaction map notation", *Mol Syst Biol, Vol. 2*, 2006.

9. Suderman, M., and M. Hallett, "Tools for visually exploring biological networks", *Bioinformatics (Oxford, England), Vol. 23, No. 20*, pp. 2651-2659, 2007.

10. Balaban, A. T., *Chemical Applications of Graph Theory*, Academic Press, 1976.

11. Di Battista, G., P. Eades, R. Tamassia, I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1998.

12. Ahuja, R. K., T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.

13. Tamassia, R., G. Di Battista, and C. Batini, "Automatic graph drawing and readability of diagrams", *Systems, Man and Cybernetics, IEEE Transactions on, Vol. 18, No. 1*, pp. 61-79, 1988.

14. Tamassia, R., "On embedding a graph in the grid with the minimum number of bends", *SIAM J. Comput., Vol. 16, No. 3*, pp. 421-444, 1987.

15. Bertolazzi, P., G. D. Battista, and W. Didimo, "Computing orthogonal drawings with the minimum number of bends", *Algorithms and Data Structures, Lecture Notes in Computer Science, Vol. 1272*, pp. 331-344, 1997.

16. Fößmeier, U., and M. Kaufmann, "Drawing high degree graphs with low bend numbers", *Graph Drawing, Lecture Notes in Computer Science, Vol. 1027*, pp. 254-266, 1996.

17. Weiss, M. A., *Data Structures and Algorithm Analysis in C (Second Edition)*, Pearson Edication, 1997.

18. Bang-Jensen J., and G. Gutin, *Diagraphs: Theory, Algorithm and Applications*, Springer, 2001.

19. Dijkstra, E.W., "A note on two problems in Connexion with graphs", *Numerische Mathematik 1*, pp. 269-271, 1959.

20. Schulz, F., D. Wagner, and K. Weihe, "Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport", *Journal of Experimental Algorithmics (JEA), Vol. 5*, 1999.

21. Holzer, M., F. Schuz, And T. Willhalm, "Combining speedup techniques for shortestpath computations", *In Experimental and Efficient Algorithms, Third International Workshop, C. C. Ribeiro and S. L. Martins, Eds. Lecture Notes in Computer Science, Vol. 3059, Springer, Heidelberg*, pp. 269284, 2004.

22. Möhring, R. H., "Partitioning graphs to speedup Dijkstra's algorithm", *Journal of Experimental Algorithmics (JEA), Vol. 11*, 2006.

23. Schulz, F., D. Wagner, and K. Weihe, "Dijkstra's algorithm on-line: An empirical case study from public railroad transport", *ACM Journal of Exp. Algorithmics, Vol. 5*, 2000.

24. "XML Technology", http://www.w3.org/standards/xml, 2010.

25. Hucka, M., A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novre, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang "The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models", *Bioinformatics, Vol. 19*, pp. 524-531, 2003.

26. "The System Biology Markup Language", http://www.sbml.org, 2010.

27. "Systems Biology Workbench", http://sbw.sourceforge.net, 2010.

28. Shannon, P., A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for integrated models of biomolecular interaction networks", *Genome Research, Vol. 13, No. 11*, pp. 2498-504, 2003.

29. "Cytoscape: An Open Source Platform for Complex-Network Analysis and Visualization", http://www.cytoscape.org, 2011.

30. Van Iersel, M.P., T. Kelder, A. R. Pico, K. Hanspers, S. Coort, B. R. Conklin, and C. Evelo, "Presenting and exploring biological pathways with PathVisio", *BMC Bioinformatics, Vol. 9, No. 1*, p. 399, 2008.

31. "Pathvisio/WikiPathways", http://www.pathvisio.org, 2010.

32. Funahashi, A., N. Tanimura, M. Morohashi, and H. Kitano, "CellDesigner: a process diagram editor for gene-regulatory and biochemical networks", *BIOSILICO, Vol. 1*, pp. 159-162, 2003.

33. "CellDesignerTM: A modeling tool of biochemical networks", http://www.celldesigner.org, 2011.

34. Bornstein, B. J., S. M. Keating, A. Jouraku. and M. Hucka "LibSBML: an API library for SBML", *Bioinformatics, Vol. 24*, pp. 880-881, 2008.

35. Sauro, H. M., M. Hucka, A. Finney, C. Wellock, H. Bolouri, J. Doyle, and H. Kitano, "Next generation simulation tools: the Systems Biology Workbench and BioSPICE integration", *OMICS, Vol. 7, No. 4*, pp. 355-72, 2003.

36. Ju, B., B. Park, J. Park, and K. Han, "Visualization and analysis of protein interactions", *Bioinformatics, Vol. 19, No. 2*, pp. 317-318, 2003.

37. Demir, E., O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and

M. Ozturk, "PATIKA: An Integrated Visual Environment for Collaborative Construction and Analysis of Cellular Pathways", *Bioinformatics, Vol. 18, No. 7*, pp. 996-1003, 2002.

38. Demir, E., M. P. Cary, S. Paley, K. Fukuda, C. Lemer, I. Vastrik, G. Wu, P. D'Eustachio, C. Schaefer, J. Luciano, F. Schacherer, I. Martinez-Flores, Z. Hu, V. Jimenez-Jacinto, G. Joshi-Tope, K. Kandasamy, A. C. Lopez-Fuentes, H. Mi, E. Pichler, I. Rodchenkov, A. Splendiani, S. Tkachev, J. Zucker, G. Gopinath, H. Rajasimha, R. Ramakrishnan, I. Shah, M. Syed, N. Anwar, O. Babur, M. Blinov, E. Brauner, D. Corwin, S. Donaldson, F. Gibbons, R. Goldberg, P. Hornbeck, A. Luna, P. Murray-Rust, E. Neumann, O. Reubenacker, M. Samwald, M. van Iersel, S. Wimalaratne, K. Allen, B. Braun, M. Whirl-Carrillo, K. Cheung, K. Dahlquist, A. Finney, M. Gillespie, E. Glass, L. Gong, R. Haw, M. Honig, O. Hubaut, D. Kane, S. Krupa, M. Kutmon, J. Leonard, D. Marks, D. Merberg, V. Petri, A. Pico, D. Ravenscroft, L. Ren, N. Shah, M. Sunshine, R. Tang, R. Whaley, S. Letovksy, K. H. Buetow, A. Rzhetsky, V. Schachter, B. S. Sobral, U. Dogrusoz, S. McWeeney, M. Aladjem, E. Birney, J. Collado-Vides, S. Goto, M. Hucka, N. Le Novre, N. Maltsev, A. Pandey, P. Thomas, E. Wingender, P. D. Karp, C. Sander, and G. D. Bader "The BioPAX community standard for pathway data sharing", *Nature Biotechnology, Vol. 28*, pp. 935942, 2010.

39. "The Systems Biology Institute", http://sbi.jp, 2011.

40. "MIMTool", http://code.google.com/p/mimtool, 2011.