MOVE

Bosman, D. W.; Blom, E. J.; Ogao, P. J.; Kuipers, O.P.; Roerdink, J. B. T. M.

Published in:
In Silico Biology

In Silico Biology 7 (2007) 35–59
IOS Press

MOVE: A Multi-Level Ontology-Based Visualization and Exploration Framework for Genomic Networks

Diederik W.J. Bosman[a], Evert-Jan Blom[b], Patrick J. Ogao[a], Oscar P. Kuipers[b] and Jos B.T.M. Roerdink[a,*]

[a]*Institute for Mathematics and Computing Science, University of Groningen, Netherlands*
[b]*Department of Molecular Genetics, University of Groningen, Netherlands*

Edited by E. Wingender; received 3 August 2006; revised and accepted 23 November 2006; published 26 December 2006

ABSTRACT: Among the various research areas that comprise bioinformatics, systems biology is gaining increasing attention. An important goal of systems biology is the unraveling of dynamic interactions between components of living cells (e.g., proteins, genes). These interactions exist among others on genomic, transcriptomic, proteomic and metabolomic levels. The levels themselves are heavily interconnected, resulting in complex networks of different interacting biological entities. Currently, various bioinformatics tools exist which are able to perform a particular analysis on a particular type of network. Unfortunately, each tool has its own disadvantages hampering it to be used consistently for different types of networks or analytical methods. This paper describes the conceptual development of an open source extensible software framework that supports visualization and exploration of highly complex genomic networks, like metabolic or gene regulatory networks. The focus is on the conceptual foundations, starting from requirements, a description of the state of the art of network visualization systems, and an analysis of their shortcomings. We describe the implementation of some initial modules of the framework and apply them to a biological test case in bacterial regulation, which shows the relevance and feasibility of the proposed approach.

KEYWORDS: Gene regulatory networks, prokaryotes, visualization framework, information exploration

INTRODUCTION

The development of biological high-throughput experimental methods has led to the availability of large amounts of raw data. These novel experimental techniques allow biologists to study the spatio-temporal behavior of specific components (e.g., metabolites, proteins, genes) of biological organisms in different environments. The resulting data sets can be analyzed to extract information on the dynamics of a biological system. Analysis and construction of network models that describe the dynamics is a central topic in molecular and cell biology.

Currently, it is not possible to derive large network models fully automatically from experimental data. Existing approaches are semi-automatic in nature and consist of a chain of sequential analytical steps. In each step the data are manually prepared for input in an (automatic) analysis algorithm, after which the researcher evaluates the results, which are then used as input for the next step. For instance,

*Corresponding author. E-mail: j.b.t.m.roerdink@rug.nl.

Electronic publication can be found in *In Silico Biol.* **7**, 0004 <http://www.bioinfo.de/isb/2007/07/0004/>, 26 December 2006.

there is a wide interest in reconstructing gene regulatory networks. Usually, the researcher starts with all available information: known networks, gene expression data and possibly other information (metabolic networks, operon maps, etc.). These data have to be preprocessed manually and combined in one data set. Subsequently, the reconstructed network has to be visualized and evaluated. Finally, the network can be used in a following analysis step to analyze the behavior of regulons.

There are several issues which hamper the outlined analysis strategy. Preparation and integration of the needed data sources is difficult and time consuming. Tools for visualization of the processed networks are not flexible, which causes the researcher to use a different visualization tool for each analysis step. For example, when new input data become available or when parameters of the analysis/reconstruction algorithm are changed, part of the chain of processing steps has to be redone. Moreover, it is difficult to relate parts of the final output network with the input data. For example, when a metabolic network is combined with other data to reconstruct a genetic network it is interesting to see which parts of the generated network relate to certain parts of the metabolic network.

In this paper we discuss the development of a software framework which allows flexible processing and interactive visualization of genomic (regulatory, metabolic, protein-interaction) networks. The framework employs a modular design to support easy addition of new input data, algorithms for analysis or visualization techniques. Our framework incorporates several new ideas to alleviate the problems described in the previous paragraph. First, all biological data and networks are described by well-defined data models so that different data sources can be used throughout the system. Second, an extensive network view is provided to explore generated networks. Third, the framework ensures that an up-to-date network is presented. This means that when the input data are modified the required analysis steps are applied automatically to keep the output up-to-date. Finally, because information from different data models is coupled it is possible to relate parts of one processed network with any other related network in the system.

We emphasize that the focus of this paper is on the conceptual foundations, starting from requirements, a description of the state of the art, and an analysis of the shortcomings of the current network visualization systems. Although we do describe the implementation of some initial modules of the framework (partly based on existing software components), a complete system is not yet available. However, we believe that an outline of a visualization framework which can not only accommodate current requirements but also future ones is of sufficient value to be presented independently of the implementation of a complete software package. Hopefully this will lead to a less fragmented approach in software design in genomic network visualization, where now manifold systems keep appearing which have a large overlap in functionality, but which still do not address some fundamental requirements as perceived by many researchers in the field.

The remainder of this paper is organized as follows. Section “Requirements” presents the system requirements in more detail. Several existing tool components will be examined in section “State of the art in gene network visualization” with respect to the specified requirements. This results in the design of the MOVE (Multi-level Ontology-based Visualization and Exploration) framework which is described in detail in section “The MOVE framework”. A partial implementation of the framework has been carried out and a biological case study is presented in section “Bacillus subtilis: a case study” to demonstrate its usefulness. Finally, the current status and possible extensions of the framework are discussed in section “Conclusions and future work”.

REQUIREMENTS

The first step in the development of an application framework is determining the software requirements. These are derived from requests by domain experts (genomic scientists and molecular geneticists) and from features of existing software packages. The requirements are split up in three categories.

Flexibility and openness

High-throughput genomics and bioinformatics are fast changing research fields. Ideally, software to be used by bioinformaticians has to be easy to adapt to new ideas. An important restriction is that we are only considering open source initiatives. This avoids complicated license issues when distributing derived applications. Furthermore, the new software framework should use a modular design. Existing modules can then be modified or replaced and new processing modules can be added to the system without having to redesign the software structure. Another important design principle is the separation of user input, the modeling of the external world, and the visual feedback to the user. The model-view-control (MVC) design pattern well known in the software engineering field provides such a separation [Buschmann *et al.*, 1996]. MVC is very important for data mining applications as it allows the original biological data to be displayed in different graphical representations which are all in correspondence with the original data source.

Biological research requirements

The framework is required to handle different types of biological networks, and support different network ontologies. The mathematical representation of the networks should be adapted to these requirements.

Network representation

Biological networks can be represented by mathematical structures called *graphs*, consisting of nodes connected by edges (also called arcs or links). The biological entities are represented by nodes and relations between these biological entities by edges. For example, in a simplified gene regulatory network each data model entity ‘Gene’ is represented by a node and each data model entity ‘regulation’ is represented by an edge. There are different types of graphs, each corresponding to different constraints on the topology of the networks. There are ‘directed’ and ‘undirected’ graphs, depending on whether there is a specific direction associated to the edges or not. Biological networks are best represented by *directed multigraphs*. A multigraph is a graph in which the same pair of nodes can be connected by multiple edges. This is a useful feature for biological networks as one biological entity can interact with another entity in different ways (for example a gene can regulate another gene by different means). This implies that the framework should be able to handle different types of (multi)graph structures.

Graphs and ontologies

A drawback of using a pure graph data structure is that information concerning the semantics (the biological ‘meaning’) of the network is lacking. Entities in a biological network have attributes like gene title (for a gene node) or gene start position (for a transcription edge). These attributes need to be stored in the graph efficiently. A biological network has several classes of nodes (e.g., genes, proteins) and different classes of edges (e.g., regulation, transcription). The scheme of attributes, meta-relations between classes and semantic constraints is called an *ontology*.

When a graph and ontology are combined it becomes possible to apply algorithms which can use both the structural information contained in the graph as well as domain specific knowledge (semantics) which is obtained from the ontology. To combine a graph with an ontology each graph element must be associated to class information stored in the ontology.

Although the ultimate aim is to visualize networks on all levels (genomic, transcriptomic, proteomic and metabolomic) in an integrated way, one often limits oneself to one or two levels. In such cases, we do not use a full ontology, but a restricted one, consisting of only those ontological classes which are necessary to describe the restricted system (say a gene interaction network).

Hence a requirement of the framework is that each processing module has as input and output a biological network which consists of a graph combined with an annotation which references the appropriate part of an ontology. We will refer to such an annotation as an *ontology map*. The ontology should preferably be one that resembles ontologies used by other network analysis software, as this will help exporting and importing data from such applications. Of course, there currently is no single ontology covering all biological information, and also ontologies evolve over time. So we may even allow for several ontologies to be used in the framework. The essential idea is that a graph transformation involves a new ontology map or annotation of graph elements, i.e., a reference to a different part in the same ontology, or to one or more different ontologies.

Supporting multiple ontology maps

A processing module is defined as having a graph with an ontology map at both input and output. This definition allows a processing module to remodel a graph in several ways which go beyond basic filtering of graph elements. First, it is possible to use the same ontology for the input and output graph. In this case graph elements are mapped from one type to another type in the same ontology. For example, the BioPax ontology supports both Gene and Protein classes, so a gene network can theoretically be re-mapped to a protein network without changing ontology. However, often cases arise (especially during network simplification) in which the processing module will add or remove information from a graph which causes its structure to conflict with the input ontology. As an example one could have an input ontology X which specifies genes, proteins and interactions between proteins and genes. A network simplification might result in a graph with genes and gene-gene interactions only. These gene-gene interactions are mediated by the protein-gene interactions. Thus although a gene network reconstruction algorithm could use prior knowledge in the form of a graph modeled in ontology X it could not output a graph modeled in this ontology as gene-gene interactions are lacking in the ontology. When this problem arises an annotation in a different ontology has to be used for the output graph.

Furthermore, the framework will have to ensure that two networks with different semantic interpretations are kept consistent when either the output or input network is modified. The mechanism that provides consistency across semantic types should be used also to provide contextual selections. When network structures in the output network have been selected, this selection corresponds to a certain selection of input network elements. Thus, it should be possible to relate a selection in one ontology map to a selection in another.

Visualization

The final processing step consists of visualizing the network. Again the MVC concept should be used here to separate the visual representation from the data structures. A major concern is that each network visualization should be a component in itself. This ensures that it will be possible to create several network visualizations with different ontologies (e.g., metabolic pathways, gene networks, protein networks).

The network visualization software should be very flexible, provide multiple views of different network representations, permit user interaction and be efficient enough to generate interactive network pictures in real-time. The network will have to be visualized using different kinds of graphical symbols as well as visual attributes (e.g., color, size, position) for these symbols. Positioning the graph components is a problem that will have to be handled by specialized automatic graph layout algorithms [Di Battista *et al.*, 1999]. Layout algorithms can already reduce the visual complexity of displayed graphs by removing screen clutter. However, forms of *graph complexity management* [Ju and Han, 2003] are required to further simplify the visualized network topology (see section "Network Complexity Management"). Finally, the visualization must support a number of interactive capabilities like symbol selection, zooming and panning.

Computational efficiency

A last concern on the software design is that the used network data structures and algorithms have to be quite efficient, both in terms of memory usage as well as computation time because several genome-scale networks will have to be handled simultaneously. In the case of large networks, these efficiency constraints are already hard to meet for the visualization itself [Herman *et al.*, 2000].

Most existing efficient graph visualization techniques will not meet all the requirements outlined in the previous paragraphs. Probably some of the outlined requirements directly conflict with applying some of the existing efficient techniques. Many existing visualization tools, although they provide spectacular results, do not meet important requirements and, even worse, are not adaptable to these requirements.

STATE OF THE ART IN GENE NETWORK VISUALIZATION

A large number of software packages exist which can be used to visualize certain aspects of biological networks. In this paragraph we analyze a few of these applications with respect to the previously formulated requirements. We also discuss some general frameworks which are used for graph visualization in other application domains.

Software packages

The software packages which are currently available can be classified in six groups, see Table 1, which can be divided in data mining/exploration applications and data editing applications. An application is further assigned to one of three classes based on whether it uses an ontology with different network semantics, uses a specific (static) ontology or does not make use of ontology information at all. We now discuss the groups one by one, mentioning some representative examples from each group, without aiming to be exhaustive.

1. The first group consists of diagram editors like JGraph, Visio and Xfig, which are often used to prepare presentation slides or detailed biological schematics in textbooks. The design process is completely done by hand. In theory one could write tools to convert network information into a format which these applications could read. However, diagram editors do not expect manually designed graphs of thousands of nodes, since this results in performance problems. Furthermore the detailed editing capabilities are not very useful to explore or get insight into complex networks.

Table 1
The six groups of applications for biological network visualization, with some representatives in each group. The ideal application would be placed in group 6, unfortunately no applications exist yet which can be assigned to this group

Ontology	Data editing	Data mining/exploration
None	1: JGraph	2: TouchGraph, KEGG, H3Viewer, GVF
Specific	3: NetBuilder, BioUML	4: Osprey, Cytoscape, VisANT
General	5: Protege, IsaViz	6:

2. This group consists of relatively simple tools that do not (or only partly) make use of ontology information. Examples are network layout techniques such as TouchGraph or visual representation techniques like H3Viewer [Munzner, 1998]. These techniques can be used on any input network, irrespective of its ontology. This group also includes tools that do use an ontology in certain parts of the visualization. KEGG [Kanehisa, 1997] for example uses a certain ontology for metabolic pathways to display (static) metabolic maps. Via a web-based script it is possible to color enzymes based on values entered by a script. These values can have any meaning and take no ontology into account. Although the tools are highly flexible, as they can process any data in the correct input format, they lack data mining functionality. Without a mapping of network elements to specified semantic types it is not possible to display abstractions of the data. For example, if a complex protein-gene network which contains gene, transcription, translation and protein nodes would be visualized with H3Viewer the visualization would also have to display these nodes. Probably it would be more interesting to have multiple visualizations: a network consisting of genes only, a network consisting of gene-protein interactions, and a network containing only protein interactions. This requires the user to manually convert the input network to one of the simplified networks.
3. This group contains applications to handle biological diagrams. NetBuilder and BioUML [BioUML, 2002] use the designed diagrams to perform simulations of a given network. Again these applications are not very usable to explore large complex networks as they lack data exploration techniques.
4. The next group consists of data mining, exploration and visualization applications. Applications like Cytoscape [Shannon *et al.*, 2003], VisANT [Hu *et al.*, 2004] and Osprey [Breitkreutz *et al.*, 2003] handle graph visualization based on both graph and ontology information. These applications were specially designed to visualize biological networks and have integrated several tools (e.g., filters, layout algorithms, graph metrics) which help biologists in solving specific problem cases. These applications each support one or more visualizations and one or more ontological classes (e.g., in VisANT nodes can either be proteins, genes, chemical compounds or KEGG pathways). Unfortunately, it is not possible in any of the current applications in this group to display the same network in multiple visualizations across multiple semantic types, e.g., as both a gene network and a metabolic network simultaneously. The semantic types that are supported by the applications in this group are integrated in the applications itself. As a result it will be quite difficult to implement other types.
5. This category consists of data modeling and visualization software. When an ontology is defined it can be used to model network information. Several tools like Protege [Noy *et al.*, 2001] and IsaViz exist for curating information stored in the model. Because the semantics of the data are precisely defined, the tools can automatically determine if relations are allowed between certain kinds of nodes. General ontology authoring tools are not designed with a specific biological goal in mind. The highly detailed editing capabilities offered may cause the visualization to contain too

much detail for analysis and exploration of large amounts of biological network data, such as result from highthroughput experiments.

From this overview we conclude that groups 1–5 lack certain required features to various degrees. From Table 1 group 6 is of special interest as it is currently empty: this means there is a need for applications that can both deal with several semantic levels and contain mining/visualization techniques to explore networks. This is exactly the focus of the MOVE framework.

Software frameworks

Instead of redesigning an existing biological network application to meet the requirements, one can consider software frameworks that can provide solutions to general classes of network visualization. A software framework is not a complete application, but it rather provides an extensible model which can be adapted to deal with the problem at hand. In our case there are two main problem domains: general graph processing and user interface design. There are a number of existing frameworks covering these areas that can be used as a basis for a full-fledged application.

The Eclipse Rich Client Platform

The Eclipse Platform was designed for building integrated development environments (IDEs). While Eclipse is well known as a Java software development environment it can be used to create many types of applications.

Eclipse integrates so-called 'plugins' which each implement a certain well-defined function. The backbone consists of a number of graphical user interface plugins that together form a rich client platform. By creating specific new application plugins the basic client can be customized. With respect to graph processing we can use existing plugins to automatically create an extensive GUI for managing graph data. To visualize graphs new plugins have to be added.

JGraph framework

The JGraph framework [JGraph, 2001–2006] is especially designed to ease problems associated with diagram presentation and editing. The *JGraphPad* application which was built using JGraph is a quite capable diagram editor demonstrating the capabilities of the framework. However, there are some drawbacks that make JGraph less suitable: JGraph assumes Java's 2D user-interface Swing toolkit, while we do not want to constrain the visualization method beforehand. Second, JGraph is a presentation framework, and does not address processing aspects. Third, visualization of large graphs is memory inefficient, as each graph component is represented by a heavy-weight component in the view.

GVF framework

The Graph Visualization Framework (GVF) [Marshall *et al.*, 2001] focuses on general aspects of graph processing and visualization. The graph visualization application Royere was built using this framework. GVF introduces several interesting concepts which we will examine in more detail.

The graphs implemented in GVF are *attributed graphs*, that is, graphs which support node/edge attributes that may be complex, i.e., not just numbers or labels. Primitive graph complexity management is also supported by permitting subgraphs to be inserted as nodes in a parent graph.

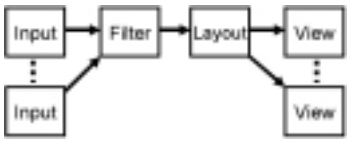


Fig. 1. GVF Pipeline building blocks. Several inputs or views can be used.

GVF pipeline and modules

Central to GVF is a graph processing pipeline consisting of connected building blocks or modules. Each module takes an input graph, processes it and generates an output graph. Modules can operate either synchronously or asynchronously. The asynchronous operation mode allows multiple modules to process graphs simultaneously, while in synchronous mode a module must wait for other modules to finish processing before starting processing itself. GVF defines five types of modules:

- *Input module*. This imports graphs in different file formats. Royere's input module supports several graph exchange formats (GML, GraphXML and CNS).
- *Filter module*. This simplifies a graph by removing unwanted nodes and edges. The filtering is based upon certain (topological) properties, such as number of edges of a node.
- *Metrics module*. This calculates values and uses these as attributes of the graph. Royere supports the Strahler metric which can be used for graph clustering.
- *Layout module*. This carries out automatic graph layout which positions the nodes and edges. The Royere layout module implements several layout algorithms (Reingold-Tilford, Fruchterman – Reingold, Radial, Ring, Barycentric and Random).
- *View module*. This displays the graph on the screen. Royere provides a Java2D visualization module. The visualization is accomplished by drawing simple arrows for the (directed) edges, and circles for the nodes.

Because of the modular scheme, it is easy to replace or add modules, e.g., new layout algorithms. Different graph problems can be solved flexibly by setting up different configurations of connected modules. For example several view modules can be coupled to one source module, thus allowing different parts of the same graph to be viewed simultaneously, see Fig. 1.

The network of processing modules is also used as an event handling system. When for example a graph attribute is modified within the source module, all connected modules are eventually notified. As a result the view will always display the most recent network.

GVF issues

The current implementation of the Royere/GVF combination has several problems:

- The filter module only allows removal, not addition, of nodes and edges, as would be required to allow generation of multiple levels of detail. Filtering is expressed by using logical constructs ('and', 'or', 'not', etc.) on the attributes of a graph element. A shortcoming of this method is that operations based on graph topology constraints (e.g., 'filter all cycles') cannot be expressed.
- Attributes in GVF consist of an identifier (e.g., 'gene title') and a value (e.g., 'genX'). Both simple values (text, numbers, etc.) are supported as well as complex Java objects (e.g., pictures and lists). Unfortunately, the attributes are stored at node/edge level, thus presenting a large overhead. For example, when a gene title is assigned to each of 4000 or more gene nodes, memory usage of the graph data structure will become prohibitive.

– Royere's view module is not very sophisticated. Displayed symbols have too few visual attributes. Furthermore symbols are either simple dots or arrows. The visualization engine uses Java2D which is not too efficient when displaying large graphs.
– With respect to the formulated requirements in section "Requirements", GVF lacks inclusion of ontologies in the graph data structure. This is an important issue because it does not permit exploration of different but related networks (e.g., gene and protein networks).

Unfortunately, GVF is not available as such but is integrated in Royere. As a result it is very hard to separate GVF from Royere and then to use GVF for development of new application. Therefore, we chose to develop our new MOVE graph processing framework geared towards biological networks. Nevertheless, our approach is heavily indebted to concepts which underlie GVF.

Ontology

To interpret any information one needs to have knowledge of the rules which define the structure and form of the information (syntax) and the rules that define the meaning (semantics). When biological networks are stored in graphs the mathematical graph concept defines the syntax and the types of permitted nodes and relations define the semantics. The syntactic and semantic rules are combined to define data models. A formal data model definition is mostly used for three purposes: (1) To consistently define the meaning (semantics) of the data, (2) To define of the internal application data structures, (3) To define of storage formats like file exchange formats and database schemas.

Semantics

Usually ontologies are used to formally specify the semantics of information. An ontology often consists of a class hierarchy in which data items are categorized. For example in a genetic network we can identify "DNA", "RNA" and "Protein" classes. These classes are hierarchically grouped under the class "Biochemical entity". In turn the class "DNA" can be split up in the classes "Gene" and "Operon".

Ontologies can be used to annotate data with labels (metadata) which indicate their meaning. In this way, the meaning of Web content becomes machine accessible and processable, which is the central idea of the *Semantic We*b initiative [Antoniou and van Harmelen, 2004; Geroimenko and Chen, 2003].

A well designed ontology will ensure that a data model is both flexible and extensible. Especially in the bioinformatics domain this is very important as no 'final' data model exists for many biological problem descriptions and there will thus always be need for ongoing improvements and additions. To identify a suitable class hierarchy it is important to study the semantics of object attributes and relations between objects. The problem is that for a given problem statement many hierarchies exist that model the data. Using these models for data processing however can be harder or easier depending on the chosen class hierarchy.

BioPax

A good example of a system using a well designed data model is the aMaze system [van Helden *et al.*, 2000]. Knowledge model standardization efforts are conducted by BioPax (Biological Pathway Exchange format), see BioPAX [2004, 2005a,b]. BioPax is an effort conducted by several key players (BioCyc, Kegg, aMaze, WIT, etc.) which aims to provide a consistent pathway exchange format, which could enable tool developers to integrate different pathway data sources.

Currently Biopax provides 'level 1' (biochemical reactions, enzyme catalysis, transport catalysis and complex assembly), and level 2 (metabolic pathways, molecular interactions and protein post-translational modifications). Future levels will expand support for signaling pathways, gene regulatory networks and genetic interactions.

OWL

It is quite interesting that BioPax has chosen to define the model using Web Ontology Language [Grau *et al.*, 2005; OWL, 2004]. OWL language definitions can be used by many data modeling tools. e.g., BioPax recommends Protege [Noy *et al.*, 2001] to allow managing and manipulation of the model definition itself as well as the data the model contains.

An application that can handle any OWL model can be used to process a larger variety of data than an application that handles only BioPax models. By supporting OWL instead of only BioPax one can modify BioPax to include regulatory interactions. Of course it is not always feasible to completely generalize the application w.r.t. to an ontology. This is especially the case when data processing (e.g., transforming one biological network to an abstraction of that network) is required, where the processing depends on the semantics of the data.

File formats incorporating ontology information

Although flat files (human readable text files) are popular among bioinformatics tools, because they can be easily viewed and edited through a text editor, machine processing of these files without human interaction is not always possible. General graph formats like GML [GML, 1997], GraphML [Brandes *et al.*, 2002] or GXL [Winter *et al.*, 2002] are better suited to store graph information. Of these formats GraphML and GXL are both based on the well established XML standard. Still the ontology itself is not stored in XML itself, and as a result the semantics of the stored information is not readily exchangeable.

The OWL standard (built on top of XML) can be used to describe the graph data as well as the mathematical concept of a graph itself. Whereas OWL approaches the definition of graphs from a more abstract view than the special graph XML formats, other formats such as SBML [Hucka *et al.*, 2003] and CellML [Lloyd *et al.*, 2004] take an opposite direction by allowing the user to directly specify biological compounds and reactions which involve these compounds.

Most of the described formats support the inclusion of custom information in which specific information can be stored which is not covered by the ontology, such as layout information. Although this can enhance compatibility between applications it also allows misuse of this feature in the sense that some applications store most of their data in the custom fields.

THE MOVE FRAMEWORK

In this section we outline the core of our new framework called MOVE (Multi-level Ontology-based Visualization and Exploration) for visualizing genomic networks. Figure 2 gives an overview of the system components. The components themselves are described in the following paragraphs.

Networks

Figure 3 (left) displays the network component in detail. A network is defined by a multigraph and semantic graph complexity management (see section “Network Complexity Management”). This can be used to group nodes in the graph based on network information (e.g., genes are grouped into operons). Each node or edge can have attributes which are provided by an attribute storage component. A selection contains a subset of nodes and edges contained in the network.

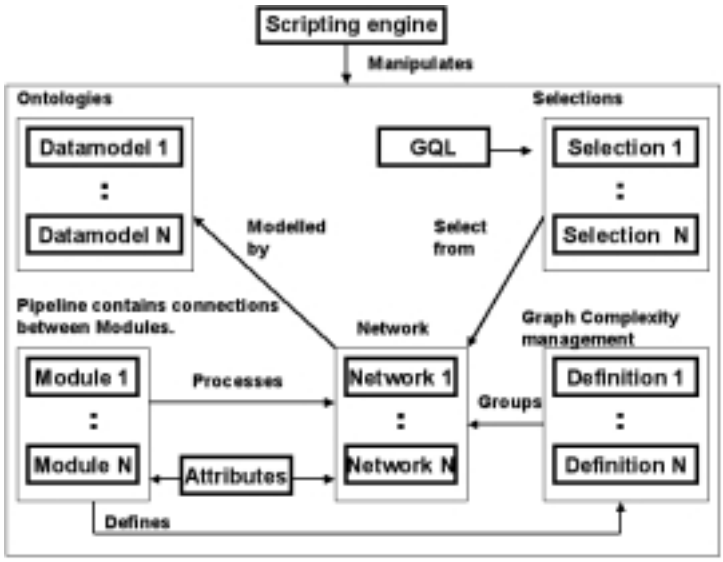


Fig. 2. System overview, with Network, Data model, Pipeline and Graph Complexity Management components.

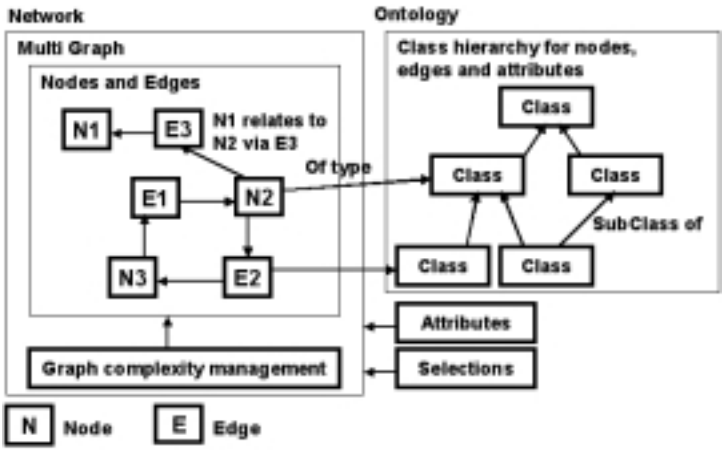


Fig. 3. Network and ontology component overview.

Ontology

The ontology (see Fig. 3, right) defines the relations between the classes of network elements. It also defines possible node/edge attributes and places constraints on the network connectivity. The Jena library was used to model the complete ontology. We are only supporting OWL-Lite models and not OWL-DL and OWL-Full ontologies. In this way, we can support BioPax, Modified BioPax (to model gene networks), and a format for simple gene networks (modeling only relations between genes).

As we keep the original OWL structure and do not use a simplified OO hierarchy, our input and output modules can export a graph as two files: one consisting of OWL Classes and another file consisting of the topology of the network (in the form of OWL Individuals).

Ontology, OWL and Biopax are described in more detail in section “Ontology”.

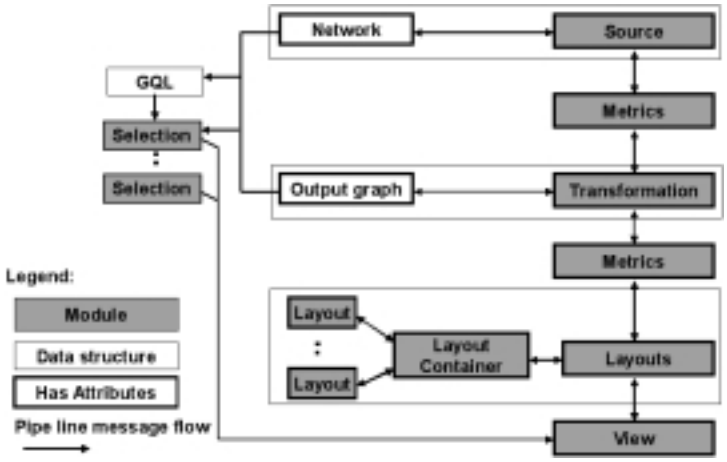


Fig. 4. Pipeline architecture overview. A standard linear pipeline is represented by the gray boxes. An actual pipeline can have many branching configurations (see Fig. 7 for different configurations). Boxes with bold frames depict data structures that support dynamic attributes.

Pipeline

The pipeline defines the data flow of processed networks and consists of modules. There are five types of modules: source, transformation, metric, layout and view. Each module has a number of input and output ports. Each output port can be connected to an input port. For example a module that combines two networks which are generated by other modules will have two input ports. This scheme of connecting modules is presented in Fig. 4. The system which provides attributed networks is used to assign attributes to modules as well. Module attributes can be used to modify parameters of module functions.

In addition to processing networks, the module pipeline is used to transport events. For example, when a node is removed from a source network a message is sent through the pipeline along connected modules. These modules can then act accordingly. In the case of a view module it will remove the symbol representing the removed node. Just as events follow a path forward through the pipeline, request messages follow the path backwards. Such messages are used to prevent the user interface from directly dealing with the internal data structures. For example, if a user wants to create a new gene in a view, the gene is not created directly. Rather a request for the creation of a gene is sent back through the pipeline. When the gene is created somewhere in one of the pipeline modules an event will be sent forward through the pipeline signaling that the gene was created. This mechanism allows the MVC concept to be easily enforced and ensures that the view naturally displays an up-to-date view of the data.

Graph attributes

The attribute storage component allows to associate attributes of any type to components (e.g., nodes, edges, modules, etc.). An attribute consists of an attribute *definition* (which may derive from an ontology) and a *value*. Attributes are stored efficiently so that gene labels for all the gene nodes in a network representing a complete genome can be stored. For example, when a gene title is retrieved for a certain gene X from a simple output gene network, the gene X is first transformed to a network structure Y in the original model. The attribute is then retrieved from the network structure Y (see the paragraph on the transformation module for details).

Some modules like layouts and views have their own attributes as well. In this case, properties control certain layout parameters and presentation options.

Source module

After a network has been imported, it is assigned to a source module which then passes on the network to connected modules. The module also handles requests and sends events when the network is modified.

Transformation module

Instead of a filtering module, as in the original GVF framework (see Fig. 1), a more general *transformation* module was conceived. This module processes an input network, containing nodes and edges modeled according to a certain ontology. The output is another network, but now containing graph elements which have new ontology maps reflecting the transformation, that is, which refer to a new (expanded, reduced) set of ontological classes. E.g., one could design a transformation which takes a complex biological network with genes, proteins, mRNA and their interactions, and produces a network containing only genes and interactions between genes.

A transformation module must define actions that should be taken when events and requests concerning certain network elements are received from other modules. When an event is received the network elements concerning this message are transformed so that they fit semantically to the target graph. In simple cases this means that the semantic types of the graph elements are just mapped to other types. In more complex situations the creation of a single input graph element could lead to the creation of several graph nodes and edges in the output graph, which requires a more complex ontology map. When a request is received an inverse transformation is applied. The message flow can even be more complex when several transformation modules are cascaded. In essence the transformation module ensures that graphs with different ontology maps are kept consistent.

Currently it is still quite hard to design and efficiently implement the specific actions a transformation module will have to take. One cause is that current general graph matching algorithms do not have real-time performance or have constraints on type of graph (or type of attributes). The Attributed Grammar System (AGG) [Taentzer, 2004] provides a graph transformation system based on rules.

Metrics module

In contrast to the transformation module which modifies the topological structure of a graph, a metrics module only assigns information to the network elements. The out-degree (number of outgoing edges) of a gene node in a regulatory network can be an interesting metric to determine which genes are important regulators. A metrics module performs the out-degree calculation for each gene node and then stores an attribute containing the value. Modules that lie downstream with respect to the metrics module, such as the view module, can use the attributes generated by the module.

Layout module

The layout module manages automatic layout algorithms. User-defined selections of the graph can be assigned to an automatic layout algorithm, which will position nodes and edges on the screen. The layout system permits layouts to be grouped hierarchically. A main layout can distribute its selection of network elements over sublayouts. The main layout then just lays out the sublayouts, the sublayouts themselves will perform a layout as normal. This feature can be used for example to lay out each connected component of a graph separately. It is also possible to assign the nodes and edges to a manual layout, in which graph elements can be repositioned as needed.

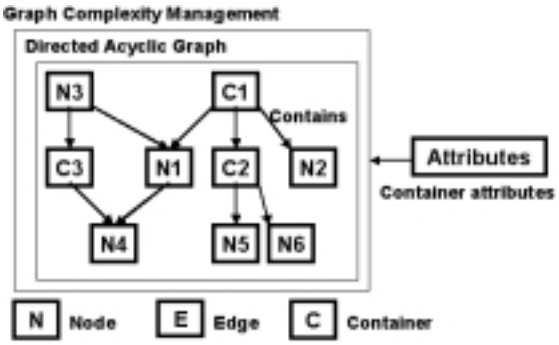


Fig. 5. Example of graph complexity management: Nodes N1,C2,N2,N5 and N6 are grouped under container C1. Nodes C3,N1 and N4 are folded onto node N3. Group C2 contains nodes N5 and N6.

Network Complexity Management

Network Complexity Management is a technique for simplifying large networks. The complexity of the network is reduced by either replacing certain groups of nodes by a single new 'container' node or by folding groups of nodes on an existing node, with an appropriate modification of the connecting edges. This process of grouping is also known as graph condensation in graph theory.

Standard approaches remove nodes from a graph after they have been condensed because the removed nodes are assumed not to be of use anymore. In the networks which are processed by modules this is not the case. Some modules may need to process condensed graphs while others will use the full graph. Therefore, rather than removing nodes from the network the complexity management component of the system administers which nodes are contained in which groups. This approach has as another advantage that condensed nodes can be part of multiple groups, see Fig. 5 for an example.

When we take into account that networks consist of both an ontology map and a graph there are two types of graph condensation.

- *Semantic grouping*: the network topology as well as the ontology is used to determine which nodes are condensed. The condensing nodes are part of the network. For example, this type of condensation is used when grouping clusters of genes.
- *Visual grouping*: only the network topology is used for the condensation. This type of condensation is used to visualize simplified graphs where uninteresting parts of the network are temporarily hidden.

Semantic grouping is achieved by defining a network complexity component for each network. Visual grouping is achieved by defining the component for each view.

Algorithms can take complexity management to their advantage. When the computational complexity of an algorithm depends on the number of nodes in the network (such as a layout algorithm) the performance increases rapidly if the algorithm is performed on a small graph. Ju and Han, 2003, use graph condensation to perform an efficient force directed layout on networks. The difference of our condensation method with their approach is that they actually replace the nodes that are condensed with so-called 'dummy' nodes while in our approach the original network is not modified.

View module

In the final step of the processing pipeline the view module visualizes the processed network by displaying symbols. Each view can have its own set of symbols. A symbol is defined as a graphical

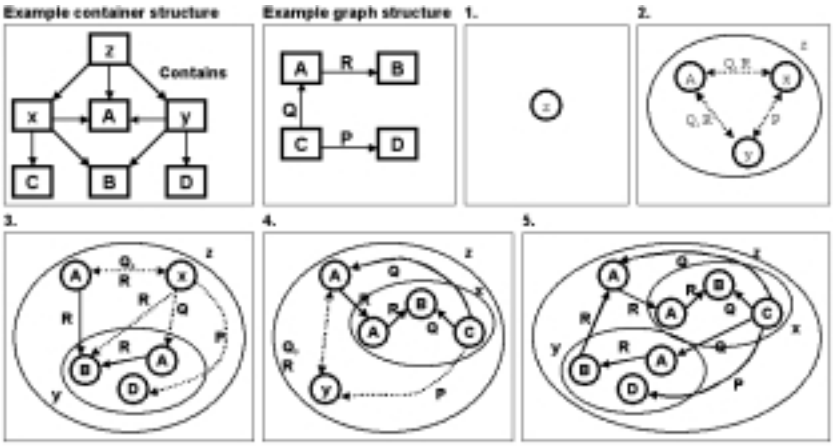


Fig. 6. Example of graph visualization using complexity management. 1. Visualized graph when all containers have been closed. 2. Visualized graph when container z is opened. The dashed arrows represent meta-edges that contain multiple edges in the original. 3. Containers z and y opened. 4. Containers z and x opened. 5. All containers opened. Note that the edge R from node A in group x to B in group y is not visualized.

representation of a class from the ontology. Choosing perceptually effective symbols to display data is not a trivial task. Kitano, 2003, has evaluated and designed a number of usable symbols. For example a gene node can be visualized by drawing a circle with a fill color, a radius and a border thickness. The view defines how attributes from the network are mapped to visual attributes of a corresponding symbol. When an attribute is modified in a network, events will cause all views visualizing this attribute to apply their mappings to visualize the updated value of the attribute.

The network complexity management component can be used to provide a simplified view of the network. Each folded or grouped node can be expanded which displays the contents of the group. Both semantic and visual group definitions can be used to hide parts of the network. Because a node can be grouped under several groups it is possible that an edge connected to such a node will be visualized by multiple edge symbols. In such a case special rules are taken into account to determine which edges have to be displayed, see Fig. 6 for an example. More specifically, the rules will ensure that if two groups are opened which contain exactly the same set of nodes (which causes each node in the set to be visualized twice) only edges are drawn between nodes in the same group.

Non-graph network visualization

The input network does not necessarily have to be visualized as a graph. An example is a linear genome ap view, which extracts gene nodes from the network and plots associated transcribed and translatedregions. In this case an arrow symbol represents positioned annotations on the genome. The concept of symbols and attribute mapping is of course still applicable.

Visualization engine

The framework architecture does not make any assumptions as to which method is used to display the graphics. We implemented a Scalable Vector Graphics (SVG) view by using Adobe's SVG engine. Unfortunately, while SVG is very versatile for static images or simple dynamic images, it has performance problems when dealing with large networks (especially those containing text labels), just as Java2D has.

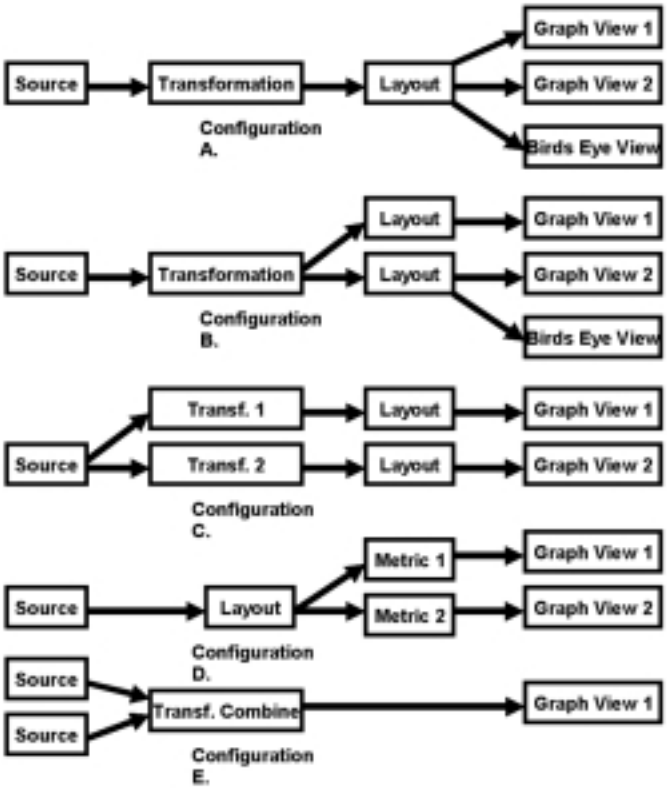


Fig. 7. Examples of pipeline configurations. Configuration A: three views of the same data. Each view can have a separate focus or zoom factor. The birds-eye view is used to provide an overview of large graphs. Layout is the same for all views. Configuration B: three views of the same data. 'Graph View 1' displays the same graph as 'Graph View 2' but now with a different layout. Birds-eye view displays an overview of the graph presented in 'Graph View 2'. Configuration C: each view displays a network modeled with different ontology maps, but derived from the same source graph. Configuration D: 'Graph View 1' displays the same graph in the same layout as 'Graph View 2'. The displayed symbols will have different visual attributes which are mapped from 'Metric 1' and 'Metric 2'. Configuration E: two source graphs are merged by a special transformation module.

Currently, we are considering alternatives, such as Open Scene Graph (OSG) [Burns and Osfield, 2003] or the Prefuse library [Heer *et al.*, 2005] for interactive graph visualization.

Pipeline configuration

By displaying several views, each with their own attribute mappings and all connected to the same source network via different transformation modules, it is possible to create many synchronized visualizations of the same data source (see section "Linked views") for more information on the synchronization mechanism). By connecting modules in different ways a number of interesting use cases can be distinguished; see Fig. 7 for examples of pipeline configurations.

Linked views

When several views are opened simultaneously, it is necessary to keep information synchronized between the views. Such synchronized selections (also known as linking and brushing) are widely used

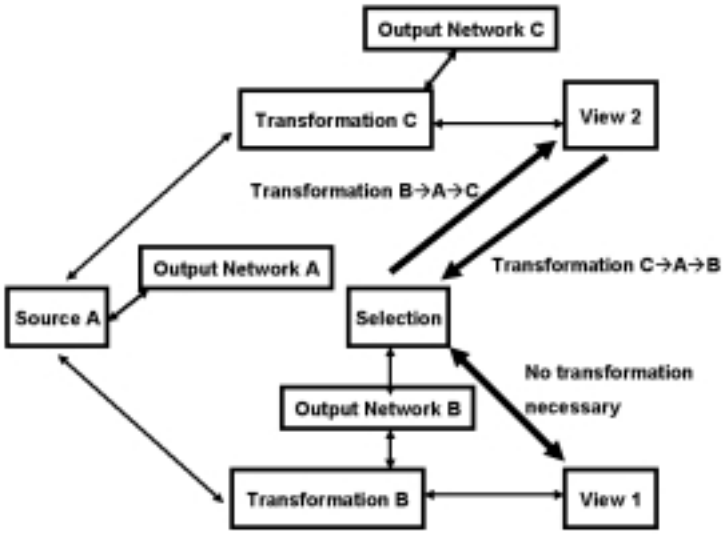


Fig. 8. Synchronizing a selection in two views. If nodes, modeled in data model B, are added to the selection in 'View 1', a notification message is sent through the pipeline. When 'View 2', which displays data model C, updates its highlighted symbol selection it will query the selection for its contents. The contents of the selection is B-1C, which means that the selected graph elements in model B are first transformed inversely via 'Transformation B' into elements in model A and then transformed forward via 'Transformation C'.

in information visualization [Keim, 2002]. The framework provides *universal selections* for this purpose. A universal selection stores a collection of selected network elements and a reference to their semantic type. The selection can be displayed in any view. Selected visualized symbols are automatically updated in all views when the selection is modified. The selection of network elements can either be created by the user (who just selects network elements in a particular view) or by an automated process. This approach can also be used in network complexity management when a user expands or collapses container nodes. In this case, a selection in one view A causes collapsed nodes in another view B to be highlighted if they contain the selected node of view A. However, this has not been implemented yet.

A particular powerful feature of universal selections is that the selection can highlight symbols in views that display a network with another ontology map than that of the selection itself. The selection cannot be directly used to highlight symbols in these views because of the difference in type. The framework solves this by tracking the path through the pipeline from the module that originally defined the selection to the target view. If this path contains transformation modules the contents of the selection will be transformed to the correct ontological classes, cf. Fig. 8.

Graph querying

In many network applications it is possible to execute search queries. In most of these applications the query comprises a set of attribute constraints which network elements must satisfy. These queries are limited by the fact that constraints on the network topology cannot be expressed. A graph query tool was implemented to support querying of graphs for certain graph structures [Dijkshoorn, 2004]. As an example the tool can be used to state queries like: "Give all cycles", "Give all paths between A and B", "Give all paths from A in which nodes must be validated by a rule X". Rules can be used to add biological semantics to queries. In a rule associated with a node a small amount of script code is used to find out if the node satisfies the scripted constraints. For example a rule can be expressed as: "Check if the node

is a gene". Because scripts can also access the annotated attribute information of network elements it is possible to formulate rules like "Check if the title property of this node is equal to X", "Check that the node has less than N edges". Graph querying can be quite computationally expensive. It is easily possible to state 'wrong' queries which will not be solved quickly even on the fastest supercomputers.

BACILLUS SUBTILIS: A CASE STUDY

We have worked on a biological case-study of the proposed framework by using *Bacillus subtilis* as a model organism. The prokaryote *B. subtilis* was chosen because 1) a reasonable amount of literature data is available; 2) these data are available in organized repositories which in turn are relatively easy to import in automated processes; 3) our team, i.e., the Molecular Genetics group at the University of Groningen, has an established expertise concerning this organism.

A source graph was constructed based on gene regulatory network information from the publicly available DBTBS (database of transcriptional regulation in *Bacillus subtili*s) database [Makita *et al.*, 2004], which contains information on transcription factors and regulated target genes.

Information abstraction

The source graph was modeled in a specific data model, which is an extension of the aMaze data model [van Helden *et al.*, 2000]. This means that the graph also contains protein, transcription, translation and RNA nodes. Most of these entities do not contain imported information, but are required by the data model. A transformation is needed to remove nodes from the network which do not contain useful information. In this case we transformed the source graph into a graph containing only genes and regulation interactions between genes.

Automatic layout

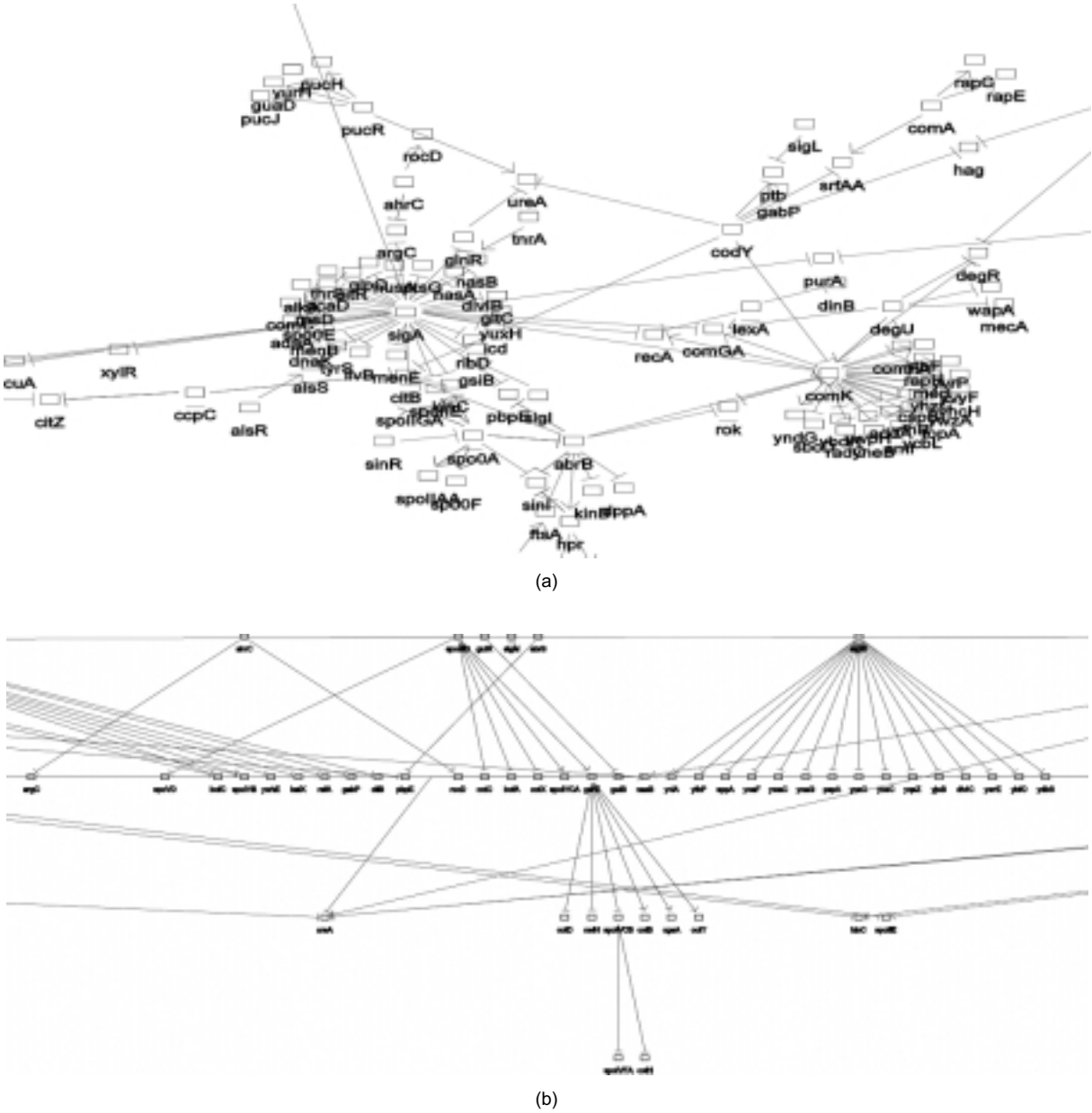
A force directed layout algorithm as well as a hierarchical layout algorithm were used to layout the DBTBS graph. The transformed network graph is presented in a network view where genes are visualized by small circle symbols and interaction nodes by arcs between genes Fig. 9. The position attribute of each gene node is used to position the circle symbol. The interaction type attribute (activation or inhibition) of each interaction node is used to color or grey-shade the arcs. Gene title attributes are displayed as text labels.

Scripting: Importing attributes on the fly

A simple Jython script was written which imports an attribute for each gene. The imported attributes consist of a simple value or a vector containing multiple values. Various types of attributes were imported using this script:

- COG (cluster of orthologous groups) attributes, which identify a gene's functional classification. The total set of genes is divided in twenty COG groups.
- Data from a time series DNA-microarray experiment were imported. Each gene is assigned a vector containing its expression level at four time points. These data were obtained from a *ccpA* mutant analysis in *B. subtilis* by Lulko et al. (manuscript in preparation).

The imported attributes were mapped to visual properties. In this case the grey value of the gene nodes represents the attribute type Fig. 10. We also developed a simple *common motif* script which determines the largest group of common regulators of selected group of genes.



(a)

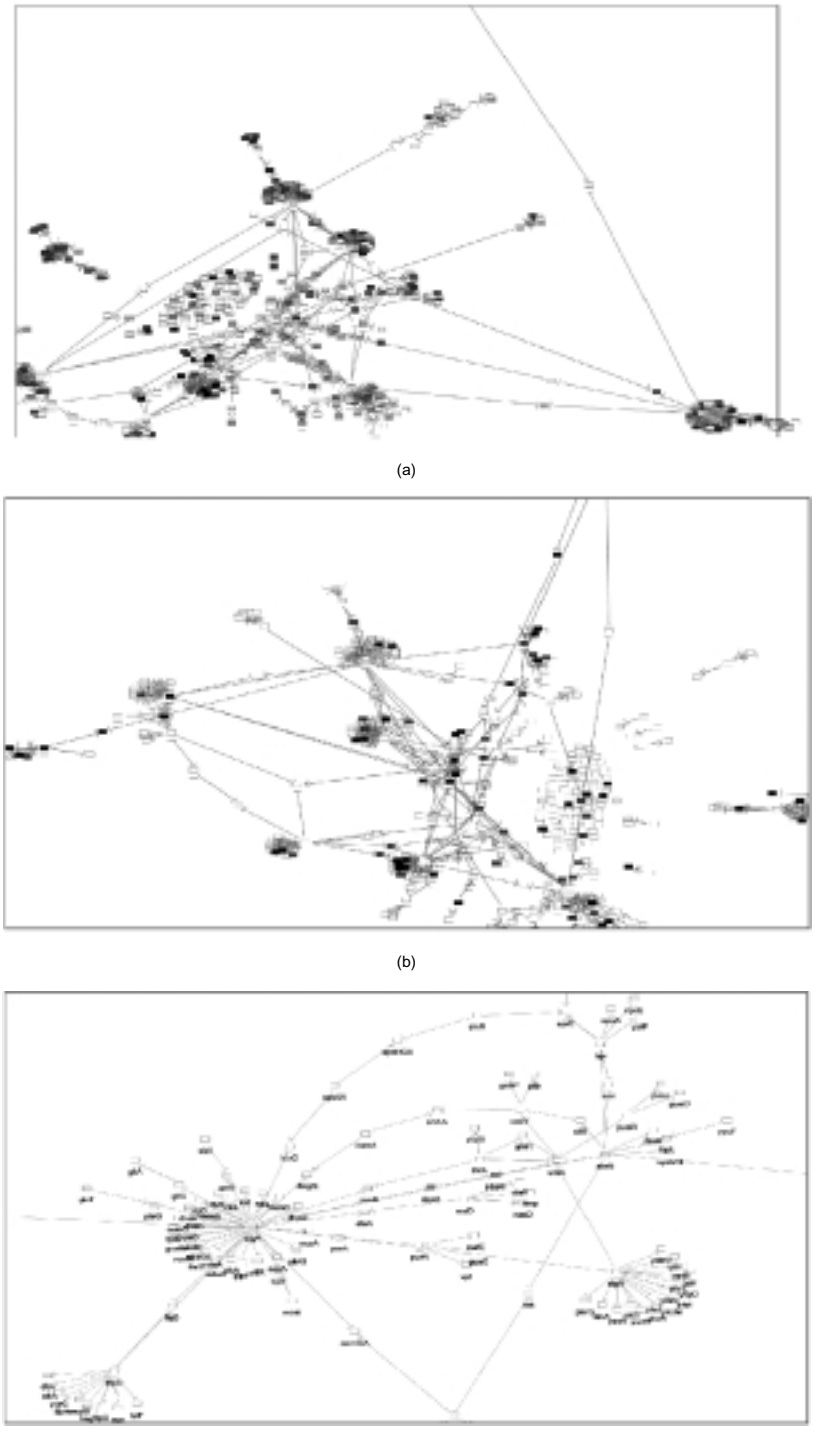
(b)

Fig. 9. DBTBS graph after performing the force directed layout (A) and hierarchical layout (B).

Graph complexity management

To demonstrate graph complexity management an algorithm was designed which collapses leaf nodes (i.e., nodes with only incoming edges) into their parent, see Fig. 11. After such an algorithm is performed on a complex graph, the graph is simplified and the essential graph structure is emphasized.

The input network does not necessarily have to be visualized as a graph. We have implemented a genome explorer, which extracts gene nodes from the network and plots associated transcribed and

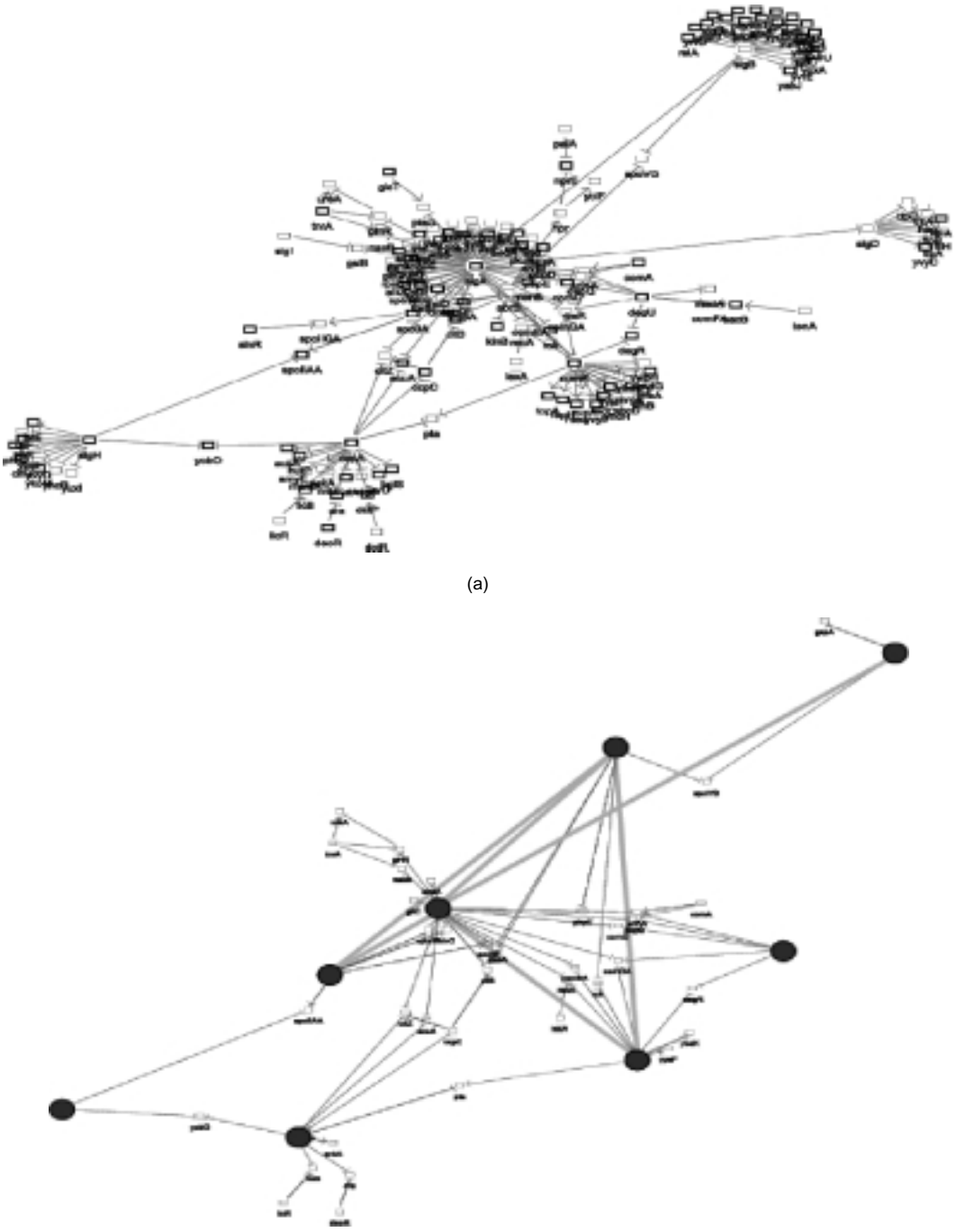


(a)

(b)

(c)

Fig. 10. DBTBS graph with grey-shaded nodes according to COG category (A), expression level at time point two of the microarray time series experiment (B), and result of the common motif analysis script (C).



(a)

(b)

Fig. 11. Collapsing leaf nodes in the DBTBS graph. (A): before collapse; (B): after collapse.

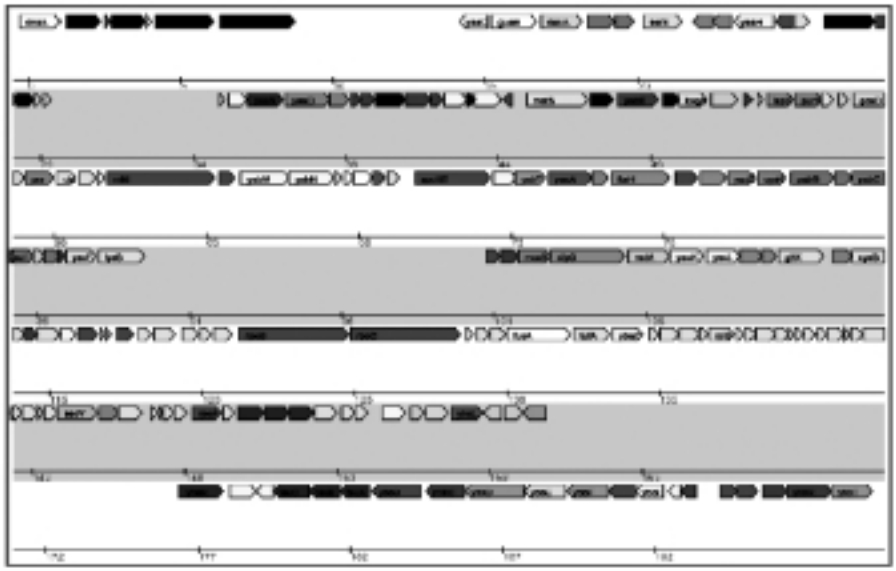


Fig. 12. Genome explorer example of part of the DBTBS network. Arrow-shaped blocks denote genes, with arrow direction indicating gene orientation on the genome. Grey values denote different COG classes.

translated regions as a linear map. The genome explorer directly uses the source graph which contains genomic information. The gene title attribute as well as genome position graph attribute are used for each gene to create the picture; see Fig. 12.

Linked view exploration

The network views as well as the genome explorer are linked. Whenever the user selects genes in the genome explorer the selection is transformed to the gene network model and the corresponding genes are selected in the network view. This also works the other way around when genes are selected in the network view.

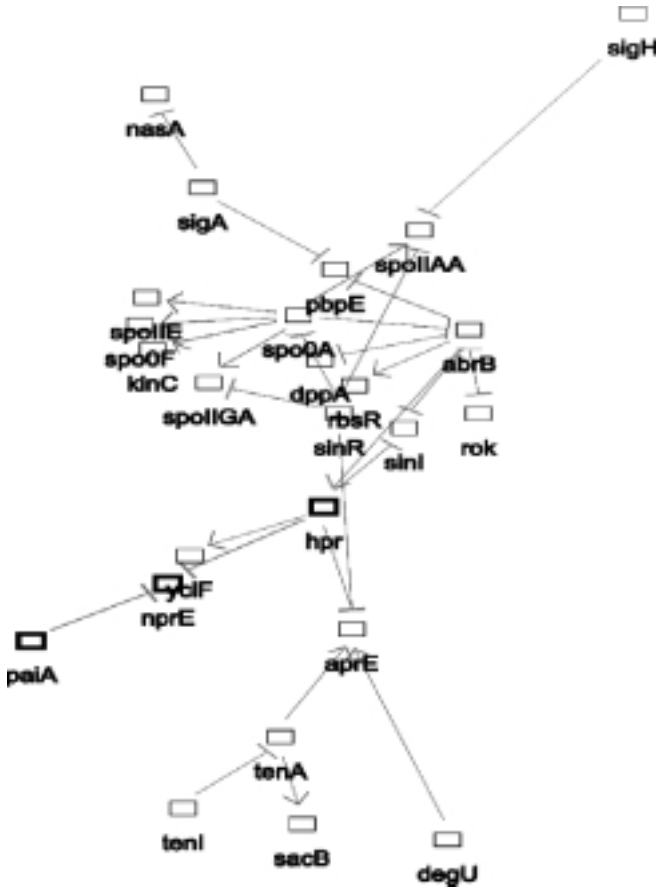
Searching

The query tool can be used to find certain graph structures. For example we can look for special cycles or graph branches, see Fig. 13. The output of a query is assigned to a selection. After assigning this selection to the network view, the found gene can be located by examining the birds-eye view or the network view itself.

CONCLUSIONS AND FUTURE WORK

We have reviewed and discussed a number of requirements and design issues which have to be addressed when developing a framework for data mining and visualization of genomic networks. The proposed framework is able to visualize biological data by displaying several views. Although many networks with different ontologies can be displayed simultaneously we have shown that it is still possible to link the visualizations in such a way that interactive exploration is possible.

(a)



(b)

Fig. 13. Graph query examples of the DBTBS network. (a): Cycles of 3 nodes, constrained to those in which one of the nodes has exactly 2 edges. (b): branches of length 3.

Future development will focus on implementation of the framework architecture by using and developing open source components, by refinements (e.g., simplification of the transformation module) and by the design of relevant modules which aid in biological network research. We are especially interested in reconstructing regulatory gene networks, which requires new source modules to import relevant data sources, transformation modules that perform network reconstruction and metrics to compare reconstructed networks.

ACKNOWLEDGEMENTS

This research is part of the project “Computational genomics of prokaryotes”, funded by the Dutch National Science Foundation (NWO), project no. 050.50.206.

REFERENCES

- Antoniou, G. and van Harmelen, F. (2004). A semantic web primer. The MIT Press.
- BioPAX (2004). BioPAX–biological pathways exchange language. level 1, version 1.0 documentation.
- BioPAX (2005a). BioPAX Working Group, level 2 documentation. http:///www.biopax.org.
- BioPAX (2005b). BioPAX Working Group, level 2 ontology. http://www.biopax.org/release/biopax-level2.owl.
- BioUML (2002). BioUML – framework for systems biology. http://www.biouml.org.
- Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M. and Marshall, M. (2002). GraphML progress report: Structural layer proposal. *In*: Proc. 9th Intl. Symp. Graph Drawing (GD ‘01), LNCS **2265**, 501–512.
- Breitkreutz, B. J., Stark, C. and Tyers, M. (2003). Osprey: a network visualization system. Genome Biology **4**, R22.
- Burns, D. and Osfield, R. (2003). Open scene graph – an open source solution for real time image generation. Image. http://www.openscenegraph.org/documentation/IMAGE2003/IMAGE_2003.pdf.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. (1996). Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley & Sons, New York, NY.
- Di Battista, G., Tollis, I. G., Eades, P. and Tamassia, R. (1999). Graph drawing : algorithms for the visualization of graphs. Prentice Hall, Englewood Cliffs, NJ.
- Dijkshoorn, E. (2004). A Graph Query Language. Master’s thesis, Institute for Mathematics and Computing Science, University of Groningen.
- Geroimenko, V. and Chen, C., editors (2003). Visualizing the semantic web : XML-based internet and information visualization. Springer, Wien, New York.
- GML (1997). Graph modelling language. http://infosun.fmi.uni-passau.de/Graphlet/GML.
- Heer, J., Card, S. K. and Landay, J. A. (2005). Prefuse: a toolkit for interactive information visualization. *In*: CHI’05: Proc. SIGCHI Conf. Human Factors in Computing Systems, ACM Press, New York, NY, USA, pp. 421–430.
- Herman, I., Melancçon, G. and Marshall, M. S. (2000). Graph visualisation and navigation in information visualisation. IEEE Trans. Visualization and Computer Graphics **6**, 24–43.
- Hu, Z., Mellor, J., Wu, J. and DeLisi, C. (2004). VisANT: an online visualization and analysis tool for biological interaction data. BMC Bioinformatics **5**, 17.
- Hucka, M., *et al.*; SBML Forum (2003). The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. Bioinformatics **19**, 524–531.
- IsaViz (2001). IsaViz 2.1: A visual authoring tool for RDF. http://www.w3c.org/2001/11/IsaViz.
- JGraph (2001-2006). http://www.jgraph.com.
- Ju, B.-H. and Han, K. (2003). Complexity management in visualizing protein interaction networks. Bioinformatics **19 Suppl. 1**, i177–i179.
- Kanehisa, M. (1997). A database for post-genome analysis. Trends Genet. **13**, 375–376.
- Keim, D. A. (2002). Information visualization and visual data mining. IEEE Transactions on Visualization and Computer Graphics **8**, 1–8.
- Kitano, H. (2003). A graphical notation for biological networks. BioSilico **1**, 169–176.
- Lloyd, C. M., Halstead, M. D. B. and Nielsen, P. F. (2004). CellML: its future, present and past. Prog. Biophys. Mol. Biol. **85**, 433–450.
- Makita, Y., Nakao, M., Ogasawara, N. and Nakai, K. (2004). DBTBS: database of transcriptional regulation in Bacillus subtilis and its contribution to comparative genomics. Nucleic Acids Res. **32**, D75–77.
- Marshall, M. S., Herman, I. and Melançon, G. (2001). An object-oriented design for graph visualization. Software: Practice and Experience **31**, 739–756.
- Munzner, T. (1998). Drawing large graphs with H3Viewer and Site Manager. *In*: Proc. Symp. Graph Drawing, Montreal, Canada, Springer-Verlag, pp. 384-393.
- NetBuilder (2002-2006). NetBuilder, a graphical tool for building logical representations of genetic regulatory networks.
- Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W. and Musen, M. A. (2001). Creating semantic web contents with Protégé-2000. IEEE Intelligent Systems **16**(**2**), 60–71.
- OWL (2004). OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features.
- Ruttenberg, A., Rees, J. A. and Luciano, J. S. (2005). Experience Using OWL DL for the Exchange of Biological Pathway Information. *In*: OWL: Experiences and Directions, Workshop Proceedings, Grau, B. C., Horrocks, I., Parsia, B. and

Patel-Schneider, P. (eds.), Galway, Ireland, http://www.mindswap.org/2005/OWLWorkshop/sub37.pdf.
- Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J., Ramage, D., Amin, N., Schwikowski, B. and Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res. **13**, 2498–2504.
- Taentzer, G. (2004). AGG: A graph transformation environment for system modeling and validation of software. *In*: Application of Graph Transformations with Industrial Relevance (AGTIVE'03), Pfaltz, J., Nagl, M. and Boehlen, B. (eds.), LNCS **3062**, 446-453.
- TouchGraph (2003-2006). http://www.touchgraph.com
- van Helden, J., Naim, A., Mancuso, R., Eldridge, M., Wernisch, L., Gilbert, D. and Wodak, S. J. (2000). Representing and analysing molecular and cellular function using the computer. Biol. Chem. **381**, 921–935.
- Visio (2000-2006). http://office.microsoft.com/visio.
- Winter, A., Kullbach, B., Riediger, V. (2002). An Overview of the GXL Graph Exchange Language. *In*: Software Visualization, Diehl, S. (ed.), LNCS **2269**, 324-336.