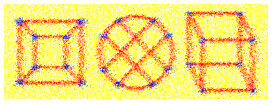
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 1, pp. 19–29 (2005)

Graph Pattern Analysis with *PatternGravisto*

Christian Klukas *Dirk Koschützki* *Falk Schreiber*

Institute of Plant Genetics and Crop Plant Research,
Gatersleben, Germany.
http://www.ipk-gatersleben.de/en/
{klukas,koschuet,schreibe}@ipk-gatersleben.de

Abstract

The analysis of patterns in graphs has applications in many fields of science. We propose a new method for analyzing graph patterns consisting of a user-friendly and flexible mechanism to specify patterns, an algorithm to recognize multiple appearances of patterns in a target graph, a pattern preserving layout algorithm, and a navigation technique to explore the underlying structure of the graph given by the patterns. This method has been implemented in a tool called *PatternGravisto*. We demonstrate the utility of our approach with the example graphs from the Graph Drawing Contest 2003 which cover problems from biology and sociology.

Article Type	Communicated by	Submitted	Revised
regular paper	G. Liotta	January 2004	July 2005

This work was supported by the German Ministry of Education and Research (BMBF) under grant 0312706A.

1 Introduction

Graphs are used to represent data and processes in many fields of science such as biology, engineering, and sociology. To analyze graph structured data and to uncover important properties *patterns* in graphs (also called *motifs* in networks) play an important role. Analyzing graph patterns has many applications, including:

- *Finding functional compounds in biological networks:* Patterns such as "feed-forward loops" may play functional roles for information processing in gene regulatory networks [12].
- *Determining the toxicology of substances:* The toxicology or carcinogenicity of chemical substances is often based on specific substructures [13].
- *Identifying substructures of circuits:* Using graph patterns sequential logic circuits can be separated into classes such that the classification is related to the circuit's functional description [11].

Motivated by the Graph Drawing Contest 2003 (`http://www.gd2003.org`) which asked entrants "to visualize distinguished graph structures that are contained in larger graphs in a distinct way" we present a method for analyzing and visualizing patterns in graphs. Our approach consists of (1) a powerful specification of patterns, (2) a pattern recognition algorithm to detect them in a given graph, (3) a pattern preserving layout algorithm to visualize the network such that patterns are easily recognizable, and (4) a navigation technique to explore the underlying structure of the graph.

This paper is organized as follows: in Section 2 we define the graph model on which we operate, describe the pattern specification and discuss our pattern recognition algorithm based on subgraph isomorphism. In Section 3 we present a graph layout algorithm which produces an overall force directed layout of the network with uniform visualization of all matches of a pattern. We furthermore address a navigation technique to explore the underlying structure of the graph using folding and unfolding of patterns and color-coding. Section 4 presents some implementation aspects. To demonstrate the utility of our method it is applied to the example graphs from the Graph Drawing Contest 2003 in Section 5. Finally, we discuss our approach in Section 6.

2 Pattern in Graphs

2.1 Graphs and Isomorphism in Graphs

We consider labeled directed graphs[1] $G = (V, E, L)$ where $V = \{v_1, \ldots, v_n\}$ is a finite set of vertices, $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V\}$ is a finite set of edges and

[1] A graph without labels can be interpreted as a graph with "empty" labels. The outlined method can also be adapted for undirected graphs, for example by using the transformation of undirected into directed graphs with antiparallel edges for both the target and the pattern graph.

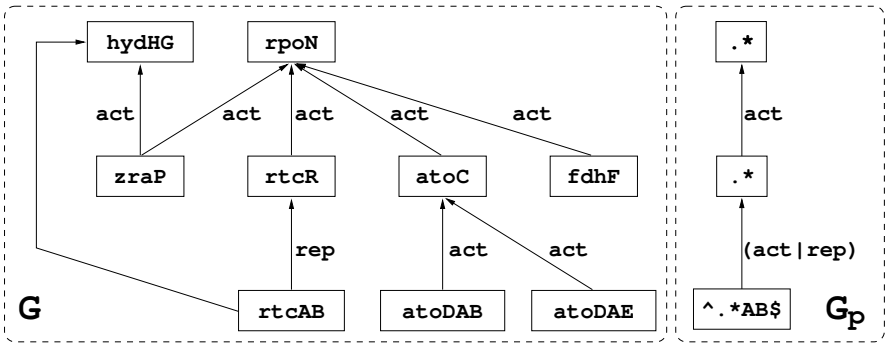

Figure 1: An example graph G and a pattern G_p with regular expressions at vertices and edges.

$L = \{l_1, \ldots, l_p\}$ is a finite set of vertex and edge labels. Each vertex or edge of the graph is required to have a not necessarily unique label. We denote the cardinality of the set of vertices by n and the cardinality of the set of edges by m.

A graph $G' = (V', E', L')$ is a subgraph of a graph $G = (V, E, L)$ if $V' \subseteq V$, $E' \subseteq E \cap (V' \times V')$, and L' is the restriction of L on $V' \cup E'$. Two graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$ are *isomorphic*, if (1) there exists a bijective mapping between the vertices in V_1 and V_2, (2) there is an edge between two vertices of one graph if and only if there is an edge between the two corresponding vertices in the other graph, and (3) the labels on the vertices and edges are preserved by the mapping. A graph G_1 is an *isomorphic subgraph* of G_2 if there exists a subgraph G_2' of G_2 which is isomorphic to G_1.

2.2 Pattern Specification

A simple form of graph pattern matching is the problem of finding a subgraph (the *match*) of an input graph (the *target*) such that the subgraph is isomorphic to another input graph (the *pattern*). This problem is known as the *subgraph isomorphism problem* [7, 14]. We are interested in a more general graph pattern matching approach which finds multiple or all matches of a pattern in the target, extends the scope defined by the patterns towards a more general definition of label equivalence, and restricts the matching of vertices further.

Patterns are labeled directed graphs. To define *families of patterns* we allow regular expressions as labels of vertices and edges. Every vertex or edge of a pattern may be labeled with a regular expression and therefore a single graph may specify either a single pattern, if the labels are interpreted as ordinary strings, or may specify a family of patterns, if the labels are interpreted as regular expressions.

Figure 1 shows a graph G and a pattern G_p with regular expressions at the vertex and edge labels of G_p. Without considering labels we find three

isomorphic subgraphs of the pattern G_p in the graph G which are shown in the following table:

vertex in pattern G_p	match 1	match 2	match 3
top vertex	rpoN	rpoN	rpoN
mid vertex	rtcR	atoC	atoC
bottom vertex	rtcAB	atoDAB	atoDAE

Taking labels into consideration we find two matches of the pattern G_p in the graph G, namely match 1 and match 2 from the previous table.

Regular expressions are not always sufficient to describe specific matches. In many application areas only some of the matches should be interpreted as valid depending on their topological environment. Think for example of a logical circuit where inner elements of a match are expected to have only as many incoming signals as defined by the pattern. Therefore, we add secondary information besides the labels to the pattern graph: every vertex may specify the number of "additional" incoming edges ($+_i : V \mapsto \mathbb{N}$) and outgoing edges ($+_o : V \mapsto \mathbb{N}$) which are not related to the pattern. This information restricts the available vertices for a match in the target graph. For example, if for the mid vertex v_m of the pattern graph G_p from Figure 1 an arbitrary number of additional incoming edges ($+_i(v_m) = \infty$) and outgoing edges ($+_o(v_m) = \infty$) are allowed we obtain the same matches as before. Other settings of $+_i(v_m)$ and $+_i(v_m)$ may restrict the matching of vertices. For example, the setting $+_i(v_m) = 0$ and $+_o(v_m) = 0$ does not allow any unrelated incoming and outgoing edges, therefore only the first solution given above (match 1) is valid.

2.3 Pattern Detection

Subgraph isomorphism, which is the basis of our matching problem, is an NP-complete problem [7]. A common approach to deal with the computationally difficult task of subgraph isomorphism is to restrict the classes of graphs. Examples are algorithms for trees [1], planar graphs [8] and graphs with bounded valence [10]. On the other hand, some graph class-independent algorithms do exist. Although a faster algorithm for the general problem of subgraph isomorphism exists [4, 5], the work of Ullmann from 1976 [14] seems to be in use in many situations.

Currently, we use Ullmann's algorithm [14] with an extension towards our pattern definition and therefore labeled, edge-restricted, directed graphs. The matching of labels with regular expressions from a pattern to the target graph is straightforward. The information concerning additional incoming and outgoing edges is handled during the creation of the matrix M^0. In Ullmann's algorithm M^0 represents pairs of vertices between graph and subgraph (pattern) and contains information about possible mappings between them. During initialization of M^0 the algorithm tests if the degree of a vertex of the target graph is equal to or greater than the degree of the corresponding vertex of the subgraph. We changed this initialization such that the in- and out-degrees of a vertex of the target graph are between the in- and out-degrees of the corresponding vertex

and these degrees plus the user-specified additional numbers of incoming and outgoing edges.

3 Pattern Visualization

3.1 Graph Drawing for Pattern Visualization

Research dealing with (sub)graph isomorphism or the detection of symmetries in graphs often considers visualization aspects. A typical visualization approach is to use force-directed layout algorithms and extend them to draw isomorphic (sub)graphs uniformly [2]. However, as far as we know no work considers the problem of visualizing a graph such that sets of matches of different patterns are easily readable.

In the following we present an approach that enhances the well known spring embedder method [3, 6] to provide a layout of graphs which contain pattern matches, where different patterns may have different layouts. In order to make the matches in the target graph easily recognizable every match will be uniformly laid out corresponding to the user defined layout of the pattern. The problem of what happens if a vertex is in the intersection of more than one match is discussed in Section 4. Our layout algorithm consists of two steps: (1) An initial placement of the vertices, and (2) the modified spring embedder loop.

The initial placement assigns random positions to all vertices that do not belong to a pattern. Afterwards all vertices of the matches in the target graph are placed with the given layout of the corresponding pattern. A random move vector is calculated and applied to all vertices that belong to a certain match such that at the end of the initial layout every match is placed on a different position without overlapping another match.

The second step is the spring embedder loop. The spring embedder method of calculating a force vector for a vertex v is used for vertices that do not belong to a match of a pattern. This force is the sum of the repulsive forces between v and all other vertices and the spring forces between v and the vertices that are connected with v. Vertices that belong to a match of a pattern are treated differently. For these vertices a uniform movement vector is calculated. This vector is the average of the forces which are computed for all vertices of the match. During the calculation of the force for such a vertex only forces to vertices that do not belong to the current match are considered. All vertices of the match are moved at once, therefore the relative layout of the vertices within the match does not change. Figure 3 shows a result of the layout algorithm.

3.2 Pattern Navigation Methods

Two methods for easy navigation through the target graph are considered: (1) color-coding and (2) folding and unfolding of matches.

Pattern matches can be color-coded, i.e., a color can be assigned to each pattern. This color is used for all matches of the pattern in the target graph

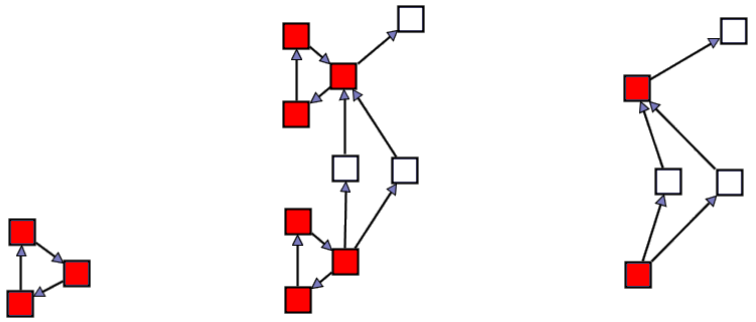

Figure 2: An example of a graph G which contains two matches of a pattern G_p. Graph G' is created by replacing the pattern matches in G with new vertices (folding operation).

and therefore allows the easy recognition of multiple matches of the pattern.

Matches of patterns in the target graph can be collapsed into single vertices (see Figure 2), which by default have the same color as the matches. The resulting graph G' is defined as follows: All vertices and edges of a match in the graph G are replaced by a new vertex, whereas edges which connect the vertices of the match to the remaining graph are preserved. It is possible to fold and unfold specific patterns or all patterns in order to easily analyze the structure of the original graph.

4 Implementation - *PatternGravisto*

PatternGravisto is based on *Gravisto*, a project from the University of Passau, Germany. We extended the functionality of the *Gravisto* graph editor via the plug-in mechanism towards pattern specification and pattern matching (see Section 2) as well as pattern layout and pattern navigation (see Section 3). We would like to emphasize some aspects of the implementation of pattern matching and pattern layout. Currently, the pattern matching algorithm allows a vertex to be a member of at most one match. The pattern layout and the pattern navigation components are restricted to deal with non-overlapping occurrences of matches. However, a pattern may match the same set of vertices in the target graph multiple times (e. g. for a clique with four vertices 24 matches to itself exist) and a vertex may belong to several matches. Furthermore, a vertex may belong to matches of different patterns. If a vertex is found multiple times during pattern matching, the first occurrence is used.

It is sometimes very difficult to find a good layout for a graph containing matched patterns with predefined layouts. Often several runs of the spring embedder algorithm with different parameters are required. To avoid this situation and improve the users' experience with the application, a thread-safe

implementation of the spring embedder algorithm has been developed. The algorithm computes the layout in the background and provides an interface for changing parameters such as optimal edge length and stiffness while the algorithm is still running. Therefore, every change of a parameter produces an immediately visible reaction and helps the user to find a good visualization in a very short time.

5 Application Examples

The usefulness of our approach is demonstrated on two examples given by the organizers of the Graph Drawing Contest 2003: A biological and a social network.

5.1 Biological Network - a Transcriptional Regulation Network

This network shows the transcriptional regulation of *Escherichia coli* (courtesy of Uri Alon, The Weizmann Institute). Vertices represent operons and transcription factors and edges represent typed interactions. Interestingly this network has frequent occurrences of small subgraphs (so-called *motifs*) described by Shen-Orr *et al.* [12] and the main request is to highlight such motifs using graph drawing methods.

A motif as described by Shen-Orr *et al.* [12] is a collection of related patterns. They define these collections by structural considerations such as patterns with similar structure but an arbitrary number of vertices (e.g. SIMs) or dense regions which fulfill additional constraints (e.g. DORs). Our approach of pattern matching is more related to subgraph isomorphism and therefore uses a fixed structure for one pattern. To deal with motifs we use sets of patterns to cover motifs.

Figure 3 shows a screenshot of *PatternGravisto* with the transcriptional regulation network consisting of 423 vertices and 578 edges. The window contains the target graph (left hand side) and shows the interface for pattern management (right hand side). One pattern which is part of the pattern set for the motif "single input module" (SIM) is shown. The target graph contains several matches of patterns which are laid out and color-coded as specified in the pattern management interface.

5.2 Social Network - a Drug Policy Making Network

The given social network represents informal communications among organizations involved in drug policy making (courtesy of Patrick Kenis, Tilburg University). A visualization of this network should depict the structure of the overall network, the structure of the subnetwork consisting only of confirmed relations, and the way in which the latter is embedded in the former.

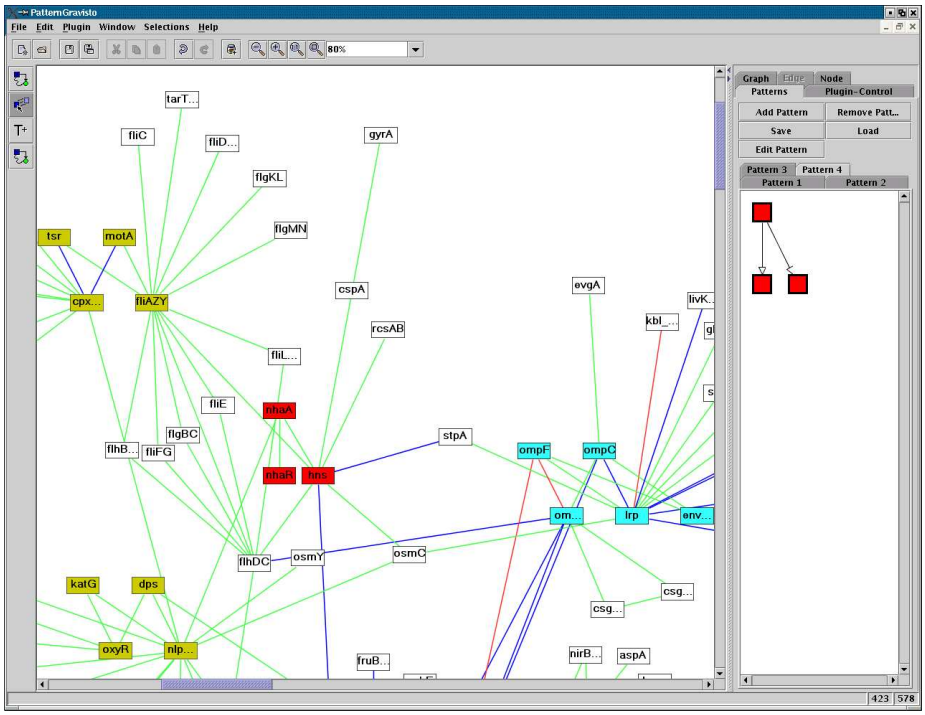

Figure 3: *PatternGravisto* showing a part of the transcriptional regulation network with some matches for motifs. For example, the highlighted subgraph with three vertices in the middle of the graph is a match of the pattern shown in the pattern management interface. To improve the readability of the drawing, edge labels have been replaced by edge colors, for instance most of the edges in this graph denote "activator" relations in the underlying transcriptional regulation network.

Our pattern matching method does not solve the given task automatically. Nevertheless, we believe that using our approach one can gain new insights into the structure of the network such as confirmed or partly confirmed relations in groups of organizations. We see our approach as an exploration technique which allows "network discovery by doing". For example, the screenshot in Figure 4 displays small groups of communication partners. We decided to focus on three organization groups where from all to none of the communication links are confirmed. The dark match shows a group where all communication links are confirmed, the grey matches where one link is confirmed, and the light grey match shows a group of three organizations where no link is formally confirmed[2].

[2]Note that every vertex can be a member of at most one match (see also Section 4), therefore groups of more than three organizations are not immediately visible.

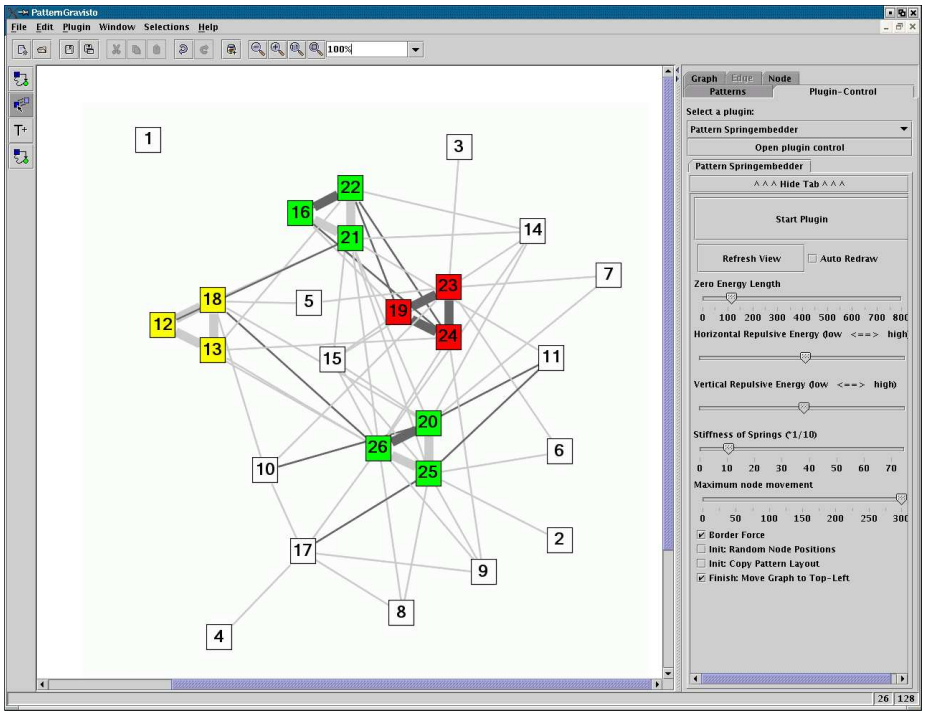

Figure 4: *PatternGravisto* showing the informal communication among organizations involved in drug policy making, with different groups of three organizations; dark edges relate to confirmed relations. The side panel shows the parameter settings for the interactive, pattern preserving spring embedder algorithm.

6 Discussion

We have presented an approach to analyze patterns in graphs. In this paper we discussed a flexible way to define patterns, an algorithm based on subgraph isomorphism to recognize patterns in a given target graph, a pattern preserving layout algorithm extending the spring embedder approach, and a navigation technique to explore the underlying structure of the graph given by the patterns. The proposed algorithms have been implemented in a tool called *PatternGravisto*.

The extensions of pattern detection and layout do not change the overall complexity of both algorithms. Running time experiments with the example graphs showed that the analysis and layout took a few seconds (the social network with 26 vertices and 128 edges and a pattern size of 3 vertices and 3 edges) and approximately five minutes (the biological network with 423 vertices and 578 edges and a pattern size of 7 vertices and 12 edges) on a normal personal computer. Nearly all this time was inevitably spent on the pattern detection

algorithm.

Our pattern matching method is based on subgraph isomorphism. The use of regular expressions for vertex and edge labels allows us to define a family of patterns by one pattern. By the restriction of additional incoming and outgoing edges we are able to be more strict than subgraph isomorphism and therefore to recognize specific matches, an extension which was motivated by discussions with users. A limitation of our approach is that there is no way to express a set of structurally similar patterns using one single pattern. One way to define such sets of patterns would be the use of graph grammars. However, graph grammars are only known within the computer science community and training and explanation hurdles for users not familiar with this concept seem to be too high, and therefore we decided to use the more straightforward solution of direct pattern specification.

We plan to extend the functionality towards a user interface which allows the user to assign a specific match of a pattern to a set of vertices. Currently, *PatternGravisto* only works with a user-provided set of patterns. We therefore want to investigate the automatic discovery of patterns in a given graph. A further possible extension is an even more flexible and faster matching algorithm. We are convinced that our method is of great value to find and visualize patterns in graphs in order to obtain new insights into underlying data. Currently, we are investigating the usability of our method in cooperation with scientists from the University of Bielefeld, Germany, for pattern analysis in hormone networks.

Acknowledgements

We would like to thank Franz J. Brandenburg, Michael Forster, Andreas Pick, Marcus Raitner and Paul Holleis (all University of Passau, Germany) for the excellent cooperation and for granting usage of *Gravisto*; and the reviewers who provided their thoughtful comments and suggestions for this paper.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[2] S. Bachl. *Isomorphe Subgraphen und deren Anwendung beim Zeichnen von Graphen*. PhD thesis, Universität Passau, 2001.

[3] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[4] P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In Jolion et al. [9], pages 176–187.

[5] P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In Jolion et al. [9], pages 149–159.

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

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[8] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proc. 6th ACM Symposium on Theory of Computing*, pages 172–184. ACM Press, 1974.

[9] J. M. Jolion, W. Kropatsch, and M. Vento, editors. *Proc. 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, 2001.

[10] E. M. Luks. Isomorphism of graphs of bounded valance can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

[11] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, 2002.

[12] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31(1):64–68, 2002.

[13] A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. E. Sternberg. The predictive toxicology evaluation challenge. In *Proc. 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1–6, 1997.

[14] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal ACM*, 23(1):31–42, 1976.