In E. Merelli et al. (Eds.), Proc. Intl. Workshop Network Tools and Applications in Biology (NETTAB'04), 91-102, 2004.

Towards Motif Detection in Networks: Frequency Concepts and Flexible Search*

Falk Schreiber Henning Schwöbbermeyer

Bioinformatics Center Gatersleben-Halle, Institute of Plant Genetics and Crop Plant Research Gatersleben, Corrensstraße 3, D-06466 Gatersleben, Germany.
{schreibe,schwoebb}@ipk-gatersleben.de

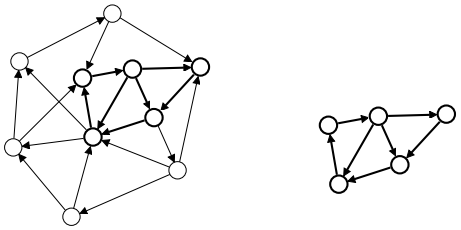
Abstract. Network motifs, patterns of local interconnections with potential functional properties, are important for the analysis of biological networks. To analyse motifs in networks the first step is finding patterns of interest. This paper presents 1) three different concepts for the determination of pattern frequency and 2) a flexible algorithm to compute these frequencies. The different concepts of pattern frequency depend on the reuse of network elements. The presented algorithm finds patterns with highest frequency and can be used to determine pattern frequency in directed graphs under consideration of these concepts. The utility of this method is demonstrated by applying it to real-world data.

1 Introduction

Biological processes form large and complex networks. Network analysis methods may help to uncover important properties of these networks and therefore may assist biologists in understanding processes in organisms. *Network motifs* can be seen as the basic building blocks of complex networks [7] and are important for the functional analysis of biological networks [8, 11]. There is no commonly used definition of a network motif. Some authors use this term to represent a set of related networks [8], for others it is simply a single small network [11]. Often motifs are described as patterns of local interconnections which occur in networks at numbers significantly higher than those in randomised networks [7, 11]. However, as it has been already noted in [7], motifs that are functionally important but not statistically significant could exist and would be missed by this approach.

The term *motif* is not precisely defined [7, 8, 11], but often refers to some functional properties of a (set of related) substructure(s) within a network. Usually the functional properties of substructures cannot be derived from their frequency alone, their frequency can only suggest potential functional properties. To distinguish between functional motifs and the substructures where the function is currently unknown we use the term pattern for such substructures of a network.

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



(a) G_T (b) G_P

Fig. 1. (a) A graph with a randomly selected subgraph (highlighted with thick lines). This subgraph is isomorphic to the graph G_P shown in (b). The highlighted subgraph in G_T is also a match of G_P in G_T.

A pattern itself is defined as a single network. We allow the user to investigate all possible patterns in the network and do not restrict our search to statistically significant substructures. This approach is similar to mining frequent patterns in networks as described in [2].

For the problem of pattern finding in collections of independent networks several algorithms have been presented [2, 3, 12]. However, there are only a few approaches to find patterns in large connected networks [5, 10]. This paper deals with two topics: 1) we introduce three different concepts for the determination of pattern frequency (the number of occurrences of a pattern in the target network) and 2) we present a flexible algorithm to find frequent patterns. This algorithm is based on [5]. It is extended to deal with the different concepts for frequency counting and to find patterns in directed graphs.

The remainder of the paper is structured as follows: in Sect. 2 we define the graph model on which we operate and describe the pattern specification used. In the following Sect. 3 we address the problem of determination of pattern frequency and present three different concepts. Sect. 4 introduces the *frequent pattern finder* algorithm and in Sect. 5 the details of the algorithm are illustrated. To demonstrate the utility of our method it is applied to typical real-world data in Sect. 6.

2 Definitions

A *directed graph* $G = (V, E)$ consists of a finite set V of vertices and a finite set $E \subseteq V \times V$ of edges. An edge $(u, v) \in E$ goes from vertex u, the source, to another vertex v, the target. The vertices u and v are said to be *incident* with the edge e and *adjacent* to each other. A subgraph of the graph $G = (V, E)$ is a graph $G_s = (V_s, E_s)$ where $V_s \subseteq V$ and $E_s \subseteq (V_s \times V_s) \cap E$. For a graph G and a subgraph G_s an edge $e = (u, v) \in E$ is called *incident* to G_s if exactly one of the vertices u, v is element of the set V_s.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a one-to-one correspondence between their vertices, and there is an edge directed

from one vertex to another vertex of one graph if and only if there is an edge with the same direction between the corresponding vertices in the other graph. The *in-degree* of a vertex is defined as the number of edges coming into the vertex, the *out-degree* as the number of edges going out of it. The *degree* of a vertex is the number of all edges connected to it.

Let G_T be a graph representing the biological network to be analysed (the *target network*) and G_P be a graph representing the *pattern* of interest. A *match* G_M is a subgraph of G_T which is isomorphic to G_P (see Fig. 1). The *pattern size* is defined in this paper as the number of edges that the pattern comprises.

3 Pattern Frequency

3.1 Different concepts for determination of pattern frequency

The *frequency of a pattern* in a target graph is the maximum number of different matches of this pattern. There are different concepts for determining the frequency of a pattern depending on which elements of the graph can be shared by two matches. Tab. 1 lists the four possibilities, from which only three can be reasonably applied.

Frequency concept $\mathcal{F}_1$ counts every match of this pattern. This concept gives a complete overview of all possible occurrences of a pattern even if elements of the target graph have to be used several times. It does not exclude possible matches (as the other concepts often do) and therefore shows the full 'potential' of the target graph with regard to the pattern.

Frequency concepts $\mathcal{F}_2$ and $\mathcal{F}_3$ restrict the reuse of graph elements shared by different matches of a pattern. If different matches share graph elements not allowed by the particular concept, not all matches can be counted for the frequency. In this case the maximum set of non-overlapping matches selected for frequency counting has to be calculated. This is known as the maximum independent set problem, and we approximate the exact result using a heuristic for performance reasons. Concept $\mathcal{F}_2$ is a common concept [10, 5] and only allows

	Graph elements shared by different matches		Values for the example in Fig. 2	
Concept	Vertices	Edges	frequency	selected matches
$\mathcal{F}_1$	yes	yes	5	$\{M_1, M_2, M_3, M_4, M5\}$
$\mathcal{F}_2$	yes	no	2	$\{M_1, M_4\}$ *or* $\{M_3, M_4\}$
$\mathcal{F}^*$	no	yes	–	–
$\mathcal{F}_3$	no	no	1	*one of* $\{M_1, M_2, M_3, M_4, M_5\}$

Table 1. Concepts for sharing of graph elements by the matches counted for the frequency of a pattern. Note that concept $\mathcal{F}^*$ is not applicable, since edges always connect vertices. In concept $\mathcal{F}_1$ where all elements can be shared, separate matches have to differ at least by one element. The concepts are applied to the graph and pattern in Fig. 2 and the results are shown at the right side of the table.

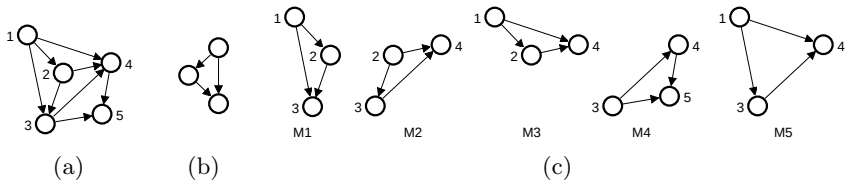


Fig. 2. An example graph (a), a pattern (b) and all different matches of the pattern (c, $M_1 - M_5$). The vertices of the graph and of the matches are numbered consecutively for identification purposes.

edge disjoint matches. Concept $\mathcal{F}_3$ is even more restrictive concerning sharing graph elements of the target graph for different matches. All matches have to be vertex and edge disjoint. The advantage of this concept is that the matches in the target can be seen as non-overlapping clusters. This clustering of the target graph allows specific analysis and navigation methods such as folding and unfolding of clusters.

The results of the application for the different concepts are illustrated by the example in Fig. 2. Applying concept $\mathcal{F}_1$, the frequency is five, for $\mathcal{F}_2$ the frequency is two counting the matches M_1 and M_4 or, alternatively M_3 and M_4. By applying concept $\mathcal{F}_3$ only one match out of the five can be selected.

An important property, which allows reduction of the search space, holds for concepts $\mathcal{F}_2$ and $\mathcal{F}_3$. For them the frequency of descending patterns (i.e. patterns on paths of the traversal tree) is monotonically decreasing with increasing size of the patterns, allowing a pruning of the search space [5].

3.2 Downward closure property of the frequency

An important feature for the presented search algorithm is the downward closure property of the frequency of the patterns, which is described in [5]. The search algorithm traverses the patterns like a tree (see Sect. 4.2), where the downward closure property ensures that the frequency of descending patterns (i.e. patterns on paths of the traversal tree) is monotonically decreasing with increasing size of the pattern.

Based on the downward closure property, the search space of patterns can be reduced by pruning of infrequent patterns in the traversal tree. Intermediately discovered patterns which fall below a particular frequency threshold can be discarded together with all patterns descending from it, since no frequent patterns can be found in this branch of the traversal tree. The downward closure property holds for frequency concepts $\mathcal{F}_2$ and $\mathcal{F}_3$, but not for $\mathcal{F}_1$. This is illustrated in Fig. 3 showing (a) a graph of size 9 and two patterns of (b) size 3 and (c) size 4. The pattern in (c) is a one-edge extension of the pattern in (b). The frequency of the pattern in (b) within the target graph in (a) is one for the three concepts. For the pattern in (c) the number of matches within the target graph in (a) is one for concept $\mathcal{F}_2$ and $\mathcal{F}_3$ and six for concept $\mathcal{F}_1$. In the latter case the number

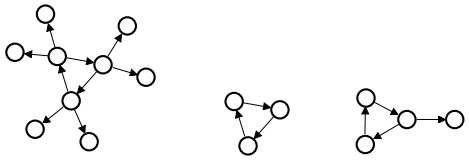


Fig. 3. A graph of size 9 (a) and two patterns of size 3 (b) and size 4 (c) illustrating that frequency concept $\mathcal{F}_1$ is not downward closed.

of matches increases for a pattern which is an extension of another pattern and hence is not downward closed.

4 Frequent pattern finder algorithm

4.1 Overview

In its basic form, the *frequent pattern finder* (FPF) algorithm searches for patterns of a given size (the target size) which occur with maximum frequency under a given frequency concept. As the number of different patterns grows very fast with increasing size of the patterns, a systematic search of all patterns of a particular size can become very time consuming even for medium-size patterns. In order to avoid the generation of a high number of patterns FPF uses a method that builds a tree of only the patterns which are supported by the target graph and traverses this tree such that only promising branches are examined. To build this pattern tree, a particular pattern of size i is assigned to a parent pattern of size $i-1$, the *generating parent*, from which it can be exclusively derived. Fig. 4 illustrates the concept.

For the frequency concepts $\mathcal{F}_2$ and $\mathcal{F}_3$ this pattern tree together with the downward closure property allows the pruning of the tree. As soon as the frequency of a pattern of intermediate size falls below the frequency of a pattern of target size discovered so far this branch of the tree can be discarded since the frequency of descending patterns is monotonically decreasing. If a (nearly) maximum frequent pattern of target size is discovered early in the search process, the frequency threshold for discarding intermediate size patterns becomes relatively high very early. Consequently, the number of patterns to be searched is reduced significantly. To find a frequent pattern early, the FPF algorithm selects the pattern with the highest frequency from the set of intermediate size patterns for expansion.

In Sect. 4.2 the frequent pattern finder algorithm is described in detail and in Sect. 4.3 points for further enhancements are outlined.

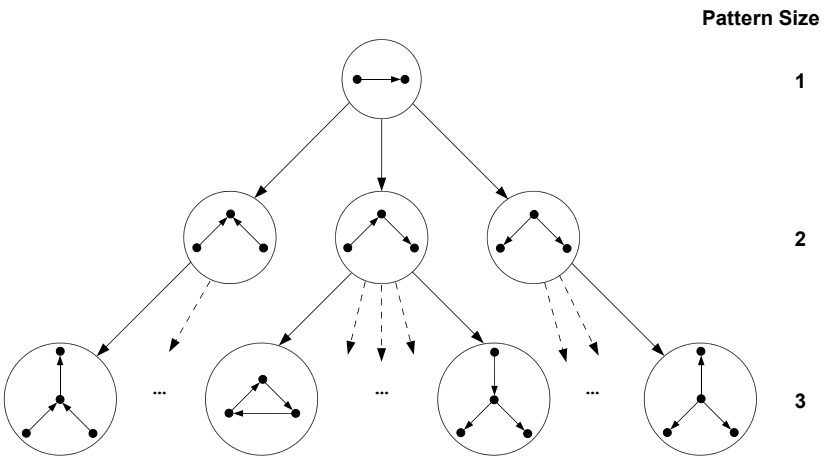


Fig. 4. The pattern tree. Note that there are more patterns of size 3 indicated by dashed lines.

4.2 Pattern traversal

In general the patterns of intermediate size are extended as long as they reach the target size and then are logged as a result pattern or are discarded if they are infrequent. The search process terminates if no patterns of intermediate size are left for extension.

The search starts with the smallest pattern p of size 1 which consists of one edge and two vertices. All matches $\mathcal{M}_{p_1}$ of this pattern are generated using every edge once. On the basis of the matches $m \in \mathcal{M}_p$ of a pattern p of size i all one-edge extension patterns p' of size $i+1$, which have p as generating parent, are created. This is done in the following way: for every match m all incident edges within the graph are identified and are used successively to create a new match m' of size $i+1$. The resulting extension pattern p' of m' is discarded if p is not the generating parent. If it is a valid extension, it is checked whether this pattern was discovered before using a canonical labelling for isomorphism testing, and it is recorded as an extension pattern if not already present. The matches $\mathcal{M}_{p'}$ for all newly created extension patterns are identified during the process of pattern generation. These newly created extension matches are recorded and are assigned to the matches of the corresponding extension pattern. The complete algorithm is shown in Alg. 1 and 2.

4.3 Enhancements of the search process

The presented algorithm searches for patterns of a given size with maximum frequency. It is straightforward to extend the algorithm, with additional features controlling the search or process tasks in parallel, to achieve better performance on multiprocessor machines:

Algorithm 1: Frequent pattern finder

```
Data   : Graph 𝒢, target pattern size t, frequency concept ℱ
Result : Set ℛ of patterns of size t with maximum frequency

ℛ ⟵ ∅ ;    f_max ⟵ 0
/* 𝒫: temporary data structure for patterns of intermediate size */
𝒫 ⟵ start pattern p_1 of size 1
ℳ_{p_1} ⟵ all matches of p_1 in 𝒢
while 𝒫 ≠ ∅ do
    𝒫_max ⟵ select all patterns from 𝒫 with maximum size
    p ⟵ select pattern with maximum frequency from 𝒫_max
    ℰ = ExtensionLoop(𝒢, p, ℳ_p)
    foreach pattern p ∈ ℰ do
        if ℱ = ℱ_1 then
            f ⟵ size(ℳ_p)
        else
            f ⟵ MaximumIndependentSet (ℱ, ℳ_p)          (see Sect. 5.4)
        end
        if size(p) = t then
            if f = f_max then
                ℛ ⟵ ℛ ∪ {p}
            else
                if f > f_max then
                    ℛ ⟵ {p} ;    f_max ⟵ f
                end
            end
        else
            if ℱ = ℱ_1 or f ≥ f_max then
                𝒫 ⟵ 𝒫 ∪ {p}
            end
        end
    end
end
```

- Search for a set of patterns with highest frequency
  Instead of searching only for the pattern with maximum frequency, the set of n frequent patterns with highest frequency to be recorded can be specified.
- Specification of a threshold for minimum frequency
  If all patterns with a frequency above a certain threshold are of interest, this threshold can be used as a global value for the minimum frequency for discarding patterns. All patterns with a frequency above this threshold are recorded.
- Parallel processing of the pattern extension
  The patterns are extended independently from each other within the *Extension Loop* of the algorithm (see Alg. 2), thus allowing parallel execution of extension loops.
- Parallel processing of different branches of the search tree

Algorithm 2: Extension Loop

Data : Graph $\mathcal{G}$, pattern p of size i, set of all matches $\mathcal{M}_p$
Result : Set of extension patterns $\mathcal{E}$ of size $i+1$ together with all matches

$\mathcal{E} \longleftarrow \emptyset$
foreach *match* $m \in \mathcal{M}_p$ **do**
  foreach *incident edge* e *of* m **do**
    $m' \longleftarrow m \cup e$
    $p' \longleftarrow$ corresponding pattern of m'
    if p *is generating parent of* p' **then**
      $\mathcal{E} \longleftarrow \mathcal{E} \cup \{p'\}$; $\mathcal{M}_{p'} \longleftarrow \mathcal{M}_{p'} \cup \{m'\}$
    end
  end
end

In the search process the pattern space is traversed like a tree. Patterns on different branches are processed independently, except that infrequent patterns are discarded during the search process and with them complete branches of the tree. If a global threshold for the frequency exists, the processing of patterns on different branches is completely independent of each other. Therefore different branches of the search tree can be processed in parallel.

Although the search space is narrowed down efficiently by the frequent pattern finder algorithm, the search process can be accelerated by executing tasks in parallel. These enhancements have been implemented but are not shown in the algorithms Alg. 1 and 2.

5 Details of the frequent pattern finder algorithm

5.1 Generating parent

A pattern of size i can be derived from up to i different patterns of size $i-1$ by adding one edge. In order to avoid the redundant generation of a pattern only one defined pattern of size $i-1$ is allowed to generate this pattern, the *generating parent*. It is defined on the basis of the canonical ordering (see Sect. 5.2) of the adjacency matrix of the graph. The last edge defined by the top-to-bottom and left-to-right ordering of the adjacency matrix which does not disconnect the remaining edges of the pattern is removed. The remaining size $i-1$ pattern is the generating parent of the given size i pattern.

5.2 Canonical label

The canonical label is a unique identifier of a graph or pattern invariant to the ordering of the vertices and edges. By comparing the canonical labels graphs can be checked for isomorphism. The principle of the algorithm for the generation of

a canonical label is described in [4]. Some modifications are made with respect to different properties of the input data. The method used for the generation of a canonical label is the transformation of the adjacency matrix into a string by concatenating it row-by-row. Since this string is dependent on the ordering of the vertices, one needs to find an ordering which is invariant with respect to isomorphism. A possible solution would be the generation of all possible permutations of the ordering of the vertices and the use of the lexicographically largest or smallest code as the canonical label. Because the number of permutations is $|V|!$ this approach is prohibitive since this number is extremely high even for medium-size graphs. A way to reduce the number of permutations needed for generating a canonical label is the division of the vertices into partitions with a definite order. Now only the vertices within one partition have to be permuted, so for the partitions containing $p_1, p_2, ..., p_m$ vertices the number of permutations reduces to $\Pi_{i=1}^{m}(p_i!)$. If it is possible to compute a fine-grain partitioning of the vertices this number is substantially smaller than that for the permutations of the entire set of vertices.

5.3 Iterative Partitioning

The iterative partitioning algorithm is a method to calculate a fine-grain partitioning of the vertices on the basis of vertex invariants. The algorithm starts with the computation of a initial partition identifier (partition-ID) for the vertices incorporating their degree. The sequence of overall degree, the out-degree and the in-degree is used to calculate a unique identifier for every unique combination. All different identifiers are sorted and the sorting index of a particular identifier is used as the partition-ID. Vertices with identical partition-IDs fall into the same partition. Subsequently the vertices are rearranged ascending according to the numerical order of their partition-IDs, thus the vertices of one partition follow directly on each other. The order of the vertices within one partition is not defined.

Now the iterative process starts with the computation of a new partition-ID for each vertex on the basis of the partition-IDs of its neighboring vertices. According to the current order of the vertices an identifier is calculated for each vertex by concatenating the partition-IDs of all neighboring vertices. First the partition-IDs of neighboring vertices on outgoing edges are concatenated, subsequently the partition-IDs of neighboring vertices on incoming edges are used. In both cases the neighboring vertices are processed in ascending order. The identifier is unique for a particular combination of this attributes. All different identifiers are sorted and the sorting index of a particular identifier is used again as the partition-ID. If new partitions have been created, the vertices are sorted ascending according to the numerical order of their partition-ID and the cycle iterates with the computation of new partition-IDs. Otherwise the partitioning process is finished.

When the iterative partitioning process has finished and all vertices fall into a unique partition nothing further has to be done. If not, the vertices of the partitions containing more than one vertex have to be permuted within their

(a)

Pattern	Frequency		
	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
P_1	3175	6	3
P_2	2785	10	5
P_3	2544	9	4

(b)

Pattern	Frequency		
	$\mathcal{F}_2$	$\mathcal{F}_1$	$\mathcal{F}_3$
P_4	12	1627	5
P_5	11	1194	5
P_6	11	996	5

(c)

Pattern	Frequency		
	$\mathcal{F}_3$	$\mathcal{F}_1$	$\mathcal{F}_2$
P_2: max($\mathcal{F}_1$)	5	2785	10
P_4: max($\mathcal{F}_2$)	5	1627	12
P_7: min($\mathcal{F}_1$,$\mathcal{F}_2$)	5	303	5

Table 2. Patterns with highest frequency for the different concepts for frequency counting, in (a) for concept $\mathcal{F}_1$, in (b) for $\mathcal{F}_2$ and in in (c) for $\mathcal{F}_3$. Since in (c) 30 patterns occur with the maximum frequency of five for concept $\mathcal{F}_3$, the three patterns which have maximum respective minimum frequency for the other two concepts were chosen for illustration. In all cases the frequencies of the patterns for the other two concepts are also presented. The patterns are shown in Fig. 5.

partition to find the permutation which leads to the lexicographically smallest code, which is used as the canonical label for the graph.

5.4 Maximum independent set

Whereas the frequency of a pattern in concept $\mathcal{F}_1$ is given by all matches of this pattern in the target graph, for the concepts $\mathcal{F}_2$ and $\mathcal{F}_3$ the pattern frequency is calculated after the determination of all matches by computing the maximum independent set on the basis of their overlap graph using a fast heuristic. The overlap graph is constructed by inserting a vertex for each match of the pattern. An edge is inserted between a pair of vertices if their corresponding pattern matches overlap, i.e. they share graph elements which the particular concept does not allow for different matches counted for the frequency.

The maximum independent set S of a graph $G = (V, E)$ is defined as the largest subset of vertices of V such that no pair of vertices of S defines an edge of E. Since the problem of finding the maximum independent set is NP-complete [1], a heuristic is used to approximate the exact result for performance reasons. A widely used heuristic is a greedy algorithm which selects a vertex of minimum degree, adds it to the independent set, deletes this vertex and all of its neighbors. This process is repeated until the graph becomes empty.

6 Application

For the application of our algorithm we took a data set of interactions of transcription factors in *Saccharomyces cerevisiae*. It was published in [6] and is available under `http://jura.wi.mit.edu/young_public/regulatory_network/`. It comprises 106 transcriptional regulators with 108 interactions, where an interaction stands for the binding of one factor to the promoter of the gene of a regulated factor and is therefore directed. Autoregulation of a factor by binding to its own promotor exists.

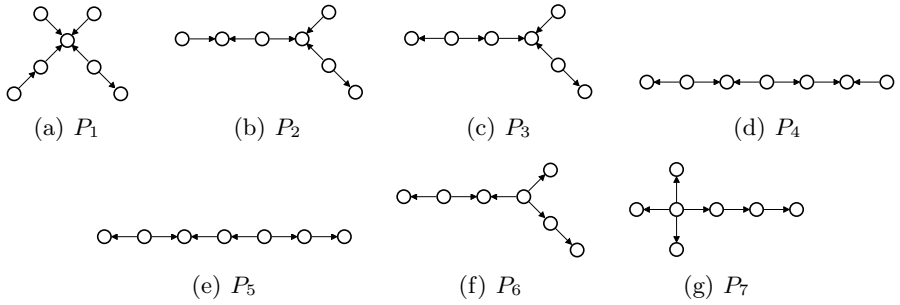


Fig. 5. The frequent patterns of Table 2.

Searching for all patterns of size 6 resulted in 1811 non-isomorphic patterns covering a broad range of frequency. Tab. 2 lists the patterns with highest frequency for the different concepts of frequency counting. Remarkably, the values for $\mathcal{F}_1$ in comparison to $\mathcal{F}_2$ and $\mathcal{F}_3$ are up to two orders of magnitude higher for frequent patterns. Comparing the values for the different concepts shows that a high frequency of a specific pattern for one concept does not necessarily imply a high frequency for another concept.

If the analysis of a particular network is focused on revealing all possible occurrences of a specific pattern, frequency concept $\mathcal{F}_1$ provides the desired information. Frequency concepts $\mathcal{F}_2$ and $\mathcal{F}_3$ are appropriate for example if the maximum number of instances of a particular pattern which can be "active" at the same time is demanded or if non-overlapping matches for visualisation purposes are of interest. The frequency of a pattern alone is not sufficient for identifying functional motifs. If a frequent pattern has a functional role in the network has to be analysed further. Therefore this step is only a starting point which generates interesting hypotheses for further studies.

7 Discussion

This paper discussed two topics related to frequent pattern mining: 1) three different concepts for the determination of pattern frequency and 2) a flexible algorithm to find frequent patterns in directed graphs. This algorithm is easily extendible. Analysing networks and finding interesting patterns is not only important in biology as shown, but has applications in other fields of science such as the toxicology or carcinogenicity of chemical substances [9] and the classification of sequential logic circuits [7].

The algorithm considers only the topology of the network, but does not take additional information about the involved elements and relations into account. This is an abstraction of the functional aspects of the biological network, however it keeps the major relationships of the elements forming the network. The incorporation of more detailed information into the search is planned for

further developments. As the parallel version of the algorithm has just been implemented, the analysis of its scalability and improvements are the next steps. Finally we plan to extend the visual analysis of the patterns in the target network by developing specific graph layout methods for this task.

References

1. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
2. A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.
3. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *IEEE International Conference on Data Mining (ICDM)*, pages 313–320, 2001.
4. M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. Technical report, Dep. of Computer Science, Univ. of Minnesota, 2002.
5. M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *SIAM International Conference on Data Mining (SDM-04)*, 2004.
6. T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, Z. Bar-Joseph, G. K. Gerber, N. M. Hannett, C. T. Harbison, C. M. Thompson, I. Simon, J. Zeitlinger, E. G. Jennings, H. L. Murray, D. B. Gordon, B. Ren, J. J. Wyrick, J. B. Tagne, T. L. Volkert, E. Fraenkel, D. K. Gifford, and R. A. Young. Transcriptional regulatory networks in Saccharomyces cerevisiae. *Science*, 298(5594):799–804, 2002.
7. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
8. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of Escherichia coli. *Nature Genetices*, 31(1):64–68, 2002.
9. A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. E. Sternberg. The predictive toxicology evaluation challenge. In *15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1–6, 1997.
10. N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *IEEE International Conference on Data Mining (ICDM)*, pages 458–465, 2002.
11. S. Wuchty, Z. N. Oltvai, and A. L. Barabási. Evolutionary conservation of motif constituents in the yeast protein interaction network. *Nature Genetics*, 35(1):176–179, 2003.
12. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *IEEE International Conference on Data Mining (ICDM)*, pages 721–724, 2002.