Frequent Subgraph Discovery in Dynamic Networks

Bianca Wackersreuther
Institute for Informatics
University of Munich
Munich, Germany
wackersb@dbs.ifi.lmu.de

Peter Wackersreuther
Institute for Informatics
University of Munich
Munich, Germany
wackersr@dbs.ifi.lmu.de

Annahita Oswald
Institute for Informatics
University of Munich
Munich, Germany
oswald@dbs.ifi.lmu.de

Christian Böhm
Institute for Informatics
University of Munich
Munich, Germany
boehm@dbs.ifi.lmu.de

Karsten M. Borgwardt
MPIs for Developmental Biology and Biological Cybernetics
Tübingen, Germany
karsten.borgwardt@tuebingen.mpg.de

ABSTRACT

In many application domains, graphs are utilized to model entities and their relationships, and graph mining is important to detect patterns within these relationships. While the majority of recent data mining techniques deal with static graphs that do not change over time, recent years have witnessed the advent of an increasing number of time series of graphs. In this paper, we define a novel framework to perform frequent subgraph discovery in dynamic networks. In particular, we are considering dynamic graphs with edge insertions and edge deletions over time. Existing subgraph mining algorithms can be easily integrated into our framework to make them handle dynamic graphs. Finally, an extensive experimental evaluation on a large real-world case study confirms the practical feasibility of our approach.

1. INTRODUCTION

Graphs are the universal data structure to model entities and their relationships. All common data structures, vectors, strings and time series, can be represented as graphs. Consequently, it is not surprising that the amount of graph-structured data is ever increasing in a wide range of application domains ranging from bioinformatics and medicine to large database management, culminating in web log analysis. Hence efficient graph mining algorithms are of utmost importance for increasing our understanding of the information represented by these large datasets of graphs. One central question in graph mining is finding frequent subgraphs within these datasets.

The majority of recent subgraph mining approaches have focused on characterizing the topology of static networks. But to model real-world systems, often a temporal component has to be taken into account, as interactions between objects here usually occur for a certain period of time only. Therefore, a realistic model has to consider that edges will be inserted and/or deleted over time. The resulting data structure is called a *dynamic graph*.

These dynamicn graphs occur in many real-world applications. In Biology, a wide-spread approach is to model interacting proteins as networks, where each vertex corresponds to a protein and two vertices are connected by an edge if the corresponding proteins can bind. In addition, further technologies allow biologists to measure the distribution of protein interactions at different time points. Hence, associating a time series for each protein provides interesting insights into the dynamically changing system. In social networks like Facebook, people contact each other at specific time points and form various complex relationships.

Our new approach for frequent subgraph discovery performs data mining on such dynamic graphs in an on-top fashion, i.e. we search a set of frequent *static* subgraphs for frequent *dynamic* patterns. Existing subgraph mining algorithms on static graphs can be easily integrated into our framework to make them handle dynamic graphs. The search for dynamic patterns is based on the idea of suffix trees.

The remainder of the paper is organized as follows: Section 2 gives a brief survey of the large previous work on mining static networks and summerizes recent papers that address data mining on dynamic graphs. Essential definitions from classic and dynamic graph theory are provided in Section 3. Our efficient algorithmic solution for finding frequent subgraphs in dynamic graphs is presented in Section 4. Section 5 provides an extensive experimental evaluation of our algorithm on large real-world biological data. Finally, Section 6 summarizes the paper and lists promising directions of future research.

MLG '10 Washington, DC USA

2. RELATED WORK

Here we review the previous work on finding patterns in graph data and focus on three problems. First, frequent subgraphs across a dataset of graphs. Second, frequent subgraphs within one single large graph. Third, frequent subgraphs in dynamic graph data.

Graph Dataset Mining. The graph dataset mining approaches can be broadly divided into two classes, apriori-based and pattern-growth based. AGM (Apriori-based Graph Mining) [10] determines subgraphs S in a dataset D of labeled graphs that occur in at least t percentage of all graphs (also called *transactions*) in D. AGM uses a canonical representation of subgraphs in order to reduces runtime costs for subgraph isomorphism checking. Similar to AGM, FSG (Frequent SubGraph Discovery) [12] uses a canonical labeling based on the adjacency matrix. Canonical labeling, and candidate generation and evaluation are sped up in FSG using graph invariants and the Transaction ID principle, which stores the ID of transactions a subgraph appeared in. This speed-up is paid for by reducing the class of subgraphs discovered to connected subgraphs, i.e. subgraphs where a path exists between all pairs of nodes.
The most well known member of the class of pattern-growth algorithms, gSpan (graph-based Substructure pattern mining) [22], discovers frequent substructures efficiently without candidate generation. Tree representations of graphs are encoded using a Depth First Search (DFS) code, amongst which a minimum DFS code is chosen according to some lexicographic order. Pre-order DFS-tree search is then conducted to find the complete set of frequent subgraphs in a set of graphs. gSpan is efficient, both w.r.t. runtime and memory requirements, making it one of the best state-of-the-art algorithms for graph dataset mining.

Large Graph Mining. Unlike graph dataset mining, large graph mining intends to find subgraphs that have at least t embeddings in one large graph. Note that any large graph mining algorithm can be applied to the graph dataset mining problem as well, simply by concatenation of all single graphs from a dataset into one large graph. Of course, subgraphs that include nodes from distinct single graphs must be pruned during pattern search then. The reverse, applying graph dataset mining algorithms to large graphs, is not directly possible. GREW [14] and SUBDUE [5] are greedy heuristic approaches for frequent graph mining that deal speed for completeness of the solution. GREW iteratively joins frequent pairs of nodes into one supernode and determines disjoint embeddings of connected subgraphs by a maximal independent set algorithm. Similarly, vSIGRAM and hSIGRAM [13] find subgraphs that are frequently embedded within a large sparse graph, using "**h**orizontal" breadth-first search and "**v**ertical" depth-first search, respectively. They employ efficient algorithms for candidate generation and candidate evaluation that exploit the sparseness of the graph. SUBDUE tries to minimize the minimum description length (MDL) of a graph by compressing frequent subgraphs. Frequent subgraphs are replaced by one single node and the MDL of the remaining graph is then determined. Those subgraphs whose compression minimizes the MDL are considered frequent patterns in the input graph. The candidate graphs are generated starting from single nodes to subgraphs with several nodes, using a computationally constrained beam search. While most of the techniques mentioned before, often use some sort of heuristic search strategy that repeatedly compresses the graph to find frequent subgraphs, methods based on sampling subgraphs to estimate their frequency are predominant in application domains, like bioinformatics [11, 19]. Another strategy is to exhaustively enumerate all subgraphs. This has the advantage that one can then compute exact rather than approximate frequencies, but for large graphs, it is only feasible for subgraphs with a limited number k of nodes, typically $k \in \{3, 4\}$.

Dynamic Graph Mining. While the evolution of graphs over time has been addressed before, the corresponding studies predominantly dealt with topics such as densification and shrinking diameters of real-world graphs over time [16]. Only a few papers [3, 6, 15] define terminology for mining dynamic networks, but to the best of our knowledge no paper presents an efficient algorithm for detecting frequent subgraphs within dynamic graphs so far.

3. FORMAL BACKGROUND

Now we provide the essential definitions from classic and dynamic graph theory necessary to follow our argumentation.

3.1 Classic Graph Theory

A **labeled graph** G is a set of vertices V, in which pairs of vertices can be linked by edges E, and in which both vertices and edges may bear labels L. A graph $G_s = (V_s, E_s)$ is a **subgraph** of G if $V_S \subseteq V$ and $E_S \subseteq E$, denoted by $G_S \subseteq G$.
The **graph isomorphism** problem is the question whether there exists a bijection f between the nodes of two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ such that $(v_{1_a}, v_{1_b}) \in E_1$ if and only if $(v_{2_a}, v_{2_b}) \in E_2$ where $v_{2_a} = f(v_{1_a})$ and $v_{2_b} = f(v_{1_b})$. If G_1 is isomorphic to G_2, we refer to (v_{1_a}, v_{1_b}) and (v_{2_a}, v_{2_b}) as **corresponding edges** in the remainder of this paper. It is unclear whether this problem is in NP or in P, and all attempts to classify it have failed so far.
Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the **subgraph isomorphism problem** consists in finding a subgraph of G_1 that is isomorphic to G_2. This problem is known to be NPcomplete [7].
If a graph G_1 is a subgraph of graph G and isomorphic to another graph G_3, then G_1 is often referred to as an **embedding** of G_3 in G. G_1 is also a **frequent subgraph** if it contains at least t embeddings of G_3, where t is an user-defined frequency threshold parameter. In application domains like bioinformatics, **motif** is often used as synonym for frequent subgraph.

3.2 Dynamic Graph Theory

Now we give formal definitions for the class of graphs we want to study, namely dynamic graphs.

DEFINITION 1 (TIME SERIES OF GRAPHS). *Given a sequence G_{ts} of n graphs $\{G_1, \ldots, G_n\}$ with $G_i = (V_i, E_i)$ for $1 \leq i \leq n$. We define G_{ts} to be a **time series of graphs** if $V_1 = V_i$ for all $1 \leq i \leq n$. G_i is the i-th state of G_{ts} and A_i is the adjacency matrix of the i-th state. In time step i, labels l_i are assigned to nodes and edges.*

Note that l_i is not necessary the same for all $1 \leq i \leq n$. This means that an edge (or even a node) might change its label in consecutive time steps of the time series. Such a time series of graphs can be transformed into a dynamic graph as follows:

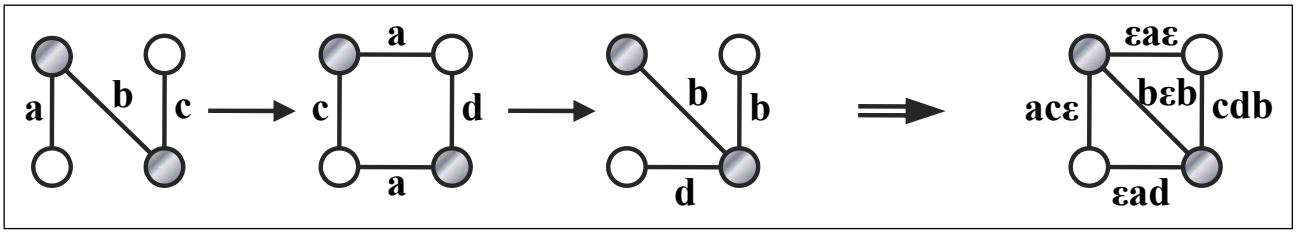


Figure 1: Transformation of a time series of graphs into a dynamic graph. The three graphs on the left represent a time series of graphs with edge insertions and edge deletions over time. The graph on the right is a dynamic graph that summarizes all information represented by the time series.

Definition 2 (Dynamic Graph). *Given a time series of graphs G_{ts} with n states. Then the* ***dynamic graph*** *$DG(G_{ts})$ of G_{ts} is defined as $DG(G_{ts}) = (V_{DG}, E_{DG}, \ell)$, where $V_{DG} = V_i$ for all $1 \leq i \leq n$ and $E_{DG} = \cup_{i=1}^{n} E_i$. The mapping $\ell : E_{DG} \rightarrow L \cup \varepsilon$ maps each edge e in E_{DG} to a string $\ell(e)$ of length n, referred to as the* **edge existence string** *of e. Let us denote the i-th character of $\ell(e)$ as $\ell(e(i))$. $\ell(e(i)) = \varepsilon$ if e does not exist in state i of G_{ts}. If e does exist in state i of G_{ts}, then $\ell(e(i)) = l_i(e)$.*

An example for such an transformation is depicted in Figure 1. Note that the union of all edges of the time series is the set of edges of the dynamic graph; therefore it is also referred to as the **union-** or **summary graph**.
To extend frequent subgraph discovery to dynamic networks, we have to define two types of frequent subgraphs:

Definition 3 (Static Frequent Subgraph). *A subgraph $G_{S_{ts}}$ that has more than t embeddings in a dynamic graph $DG(G_{ts})$ is called a* ***static frequent subgraph****.*

In other terms, static frequent subgraphs in a dynamic graph are subgraphs that are topologically frequent in the summary graph. Hence, edge existence strings play no role when looking for static frequent subgraphs. The static frequent subgraphs in a dynamic graph are defined in exactly the same manner as frequent subgraphs on one single network.
A dynamic pattern D is a static frequent subgraph S such that in more than u embeddings of S, edge insertions, deletions and label changes occur in the same temporal order. As we want to study cases where these insertions and deletions happen in a subsection of the complete time series only, we now have to make use of the existence strings of the edges.

Definition 4 (Common Substring). *For two strings s_1 and s_2 of length $|s_1|$ and $|s_2|$, $sub(s_1, i, j) = sub(s_2, i, j)$ means that the substrings from the i-th to the j-th character in s_1 and s_2 are identical, where $1 \leq i \leq j \leq \min(|s_1|, |s_2|)$. This identical substring is called a* ***common substring*** *of s_1 and s_2.*

Definition 5 (Common Dynamic Pattern). *Let S be a static frequent subgraph in dynamic graph DG, and let $S_1 = (V_{S_1}, E_{S_1})$ and $S_2 = (V_{S_2}, E_{S_2})$ be two embeddings of S in DG. As S_1 and S_2 are isomorphic, there must be a bijection f between their set of edges E_{S_1} and E_{S_2}. Then S_1 and S_2 share a* ***common dynamic pattern*** *from position i to j if for all pairs of edges (e_1, e_2) from $E_{S_1} \times E_{S_2}$ where $e_2 = f(e_1)$ the following equality holds: $sub(\ell(e_1), i, j) = sub(\ell(e_2), i, j)$, i.e. the existence strings of corresponding edges are identical from position i to position j.*

If enough embeddings of the same static frequent subgraph share the same common dynamic pattern, then this static frequent subgraph contains a dynamic frequent subgraph.

Definition 6 (Dynamic Frequent Subgraph). *Let S be a static frequent subgraph with t embeddings $\{S_1, \ldots, S_t\}$ in a dynamic graph DG. Let u be a user-defined frequency threshold. If at least u embeddings of S share the same common dynamic pattern, then the topology of S and the common dynamic pattern represent a* ***dynamic frequent subgraph****.*

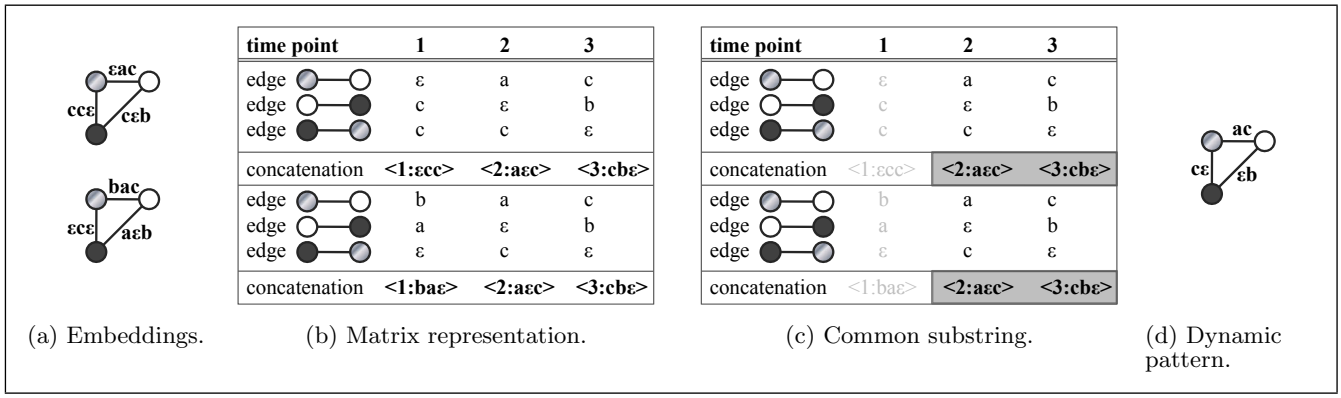
For instance, in our real-world case study (cf. Section 5), a dynamic frequent subgraph describes a set of groups of proteins that show similar patterns of co-expression during a certain interval of a time series.

4. EFFICIENT DYNAMIC FREQUENT SUBGRAPH DISCOVERY

After defining frequent subgraphs in dynamic networks, we now present an efficient algorithmic solution on how to compute them. Our framework is based on the idea of suffix trees. Dynamic frequent subgraph discovery can be performed in an on-top fashion in two steps. First we employ one of the state-of-the-art algorithms for finding frequent subgraphs in the union graph of a time series of graphs (cf. Section 2). Second we search the resulting static frequent subgraphs for frequent dynamic patterns. Figure 2 visualizes an example for frequent subgraph discovery in a dynamic network.

Matrix Representation of Embeddings. To discover dynamic patterns we process the results of a static frequent subgraph mining algorithm (see Figure 2(a)). Key to our efficient scheme is to represent each embedding of a frequent static subgraph by the set of existence strings of its edges. Hence, for each of the s embeddings of a static frequent subgraph S with ℓ edges, we obtain a set of ℓ existence strings. We represent these sets of strings for one embedding S_i of S as an $\ell \times n$ matrix, $M(S_i)$, where each row corresponds to one edge, and each column to one time point in the time series (see Figure 2(b)).

Canonical Edge Order. To compare the matrices $M(S_1)$ and $M(S_2)$ of two different embeddings S_1 and S_2 of the same subgraph S efficiently, we have to define a canonical ordering of edges for all embeddings of S. gSpan [22] orders



(a) Embeddings. (b) Matrix representation. (c) Common substring. (d) Dynamic pattern.

Figure 2: Two embeddings (2(a)) of the same static frequent subgraph and their common dynamic pattern depicted as dynamic graph (2(d)). The two embeddings are shown with their matrix and string representation each (2(b)). In time step 2 and 3, edge insertions and deletions occur simultaneously in both embeddings. This is the common substring (2(c)) that refers to the dynamic pattern.

the edges in each embedding of a frequent subgraph according to a Depth First Search on the vertices. In this manner, we can sort the rows of all matrices representing embeddings of the same subgraph such that corresponding edges are represented by the same row in each of these matrices.

String Representation of Embeddings. Using this canonical form of the matrices dynamic frequent subgraph discovery is equivalent to detecting identical blocks of columns. Therefore we transform each matrix into a string representation by treating each column in the matrix as one character which is just the concatenation of all entries in this column. We obtain s strings, each describing one embedding, which have a common substring if the embeddings show identical dynamic behaviour over a certain period of time (see Figure 2(c)).

Frequent Common Substring Discovery. A classic result from the field of string algorithms [9] helps us to discover these common substrings extremely efficiently: detecting all longest substrings in a set of s strings is possible in linear runtime, i.e. in time proportional to reading all strings exactly once by employing a suffix tree to store all substrings from a set of strings. A simple breadth-first search in the tree then provides all frequent common substrings. Obviously, each of these frequent common substrings corresponds to one dynamic pattern (see Figure 2(d)).

To guarantee that all embeddings of a dynamic pattern start at the same time point of the time series, we include a time stamp into our matrix representation of each embedding, by adding an extra row to the matrix, which numbers the time steps from 1 to n, where n is the number of time steps in the time series. We then include these time stamps into the string representation of our matrix. Our framework is summarized in Algorithm 1.

Algorithm 1 Dynamic frequent subgraph discovery

Input: All embeddings of one static frequent subgraph S
for each embedding S_i of S **do**
1) Order the edges of S_i according to canonical labeling
2) Store their existence strings in this order as a matrix
3) Translate each column of the matrix $M(S_i)$ into one character
4) Concatenate these characters into one string
end for
Perform frequent substring discovery on all the resulting strings
Output: Dynamic patterns within embeddings of S

5. EXPERIMENTS

In this section, we confirm the practical feasibility of our framework for finding frequent subgraphs in dynamic networks on the basis of a large case study on real-world biological data.

5.1 Dynamic Network Construction

We create a dynamic network by integrating protein-protein-interaction (PPI) data from yeast and a time series of yeast gene expression levels. We obtain the yeast PPI network data from the database DIP (Database of Interacting Proteins) [21, released on January 6, 2008], containing all pairs of interacting proteins identified in the yeast organism *S. cerevisiae*. This dataset consists of 4,923 proteins, which are the original set of nodes in our dynamic graph and 18,324 interactions representing edges between the nodes.

In order to assign labels to all nodes of the network, we are mapping the proteins to their corresponding protein-coding open reading frames (ORFs), via information provided by the SGD (Saccharomyces Genome Database) [8] from DIP. Finally, we determine the 'molecular function' associated with this ORF in the Gene Ontology (GO) hierarchy at depth two [2]. As GO terms are organized in directed acyclic graphs, and each term can be traced to different depths, it can occur that multiple GO annotations can be assigned to one single ORF at a given depth. As current methods for frequent subgraph mining and enumeration require a unique

label, we retaine all ORFs with a non-ambiguous GO labeling at depth two. The 12 remaining GO functional classes are then used as node labels for the remaining proteins.
Next we use a cell cycle time series of yeast gene expression levels by [4], available from the ExpressDB collection of yeast RNA Expression Datasets [1]. The dataset consists of 6,601 time series of 17 measurements for 6,259 ORFs, of which 6,045 ORFs can be mapped to one unique time series. 3,607 ORFS are both present in our GO-labeled yeast PPI network and in the gene expression datatset. The corresponding proteins are the final set of nodes in our dynamic graph, other nodes and their adjacent edges from the original graph were skipped. The final network comprises 3,607 nodes, 12 different classes of node labels, 7,395 edges per time step, and 17 time steps.

Edge Existence Strings. Each edge in our dynamic network is assigned an edge existence string (cf. Definition 2) of length 17 according to the following rules: We distinguish three basic categories of gene expression levels for each gene g at each time point i. We refer to its expression level at time point i as $g(i)$. Categories are assigned by comparing the median expression level of g across the time series with the $z - score_{median}$ of $g(i)$:

$$z - score_{median}(g(i)) = \frac{g(i) - median_{j\in\{1,..,n\}}g(j)}{\sqrt{\frac{1}{n}\sum (g(k) - median_{j\in\{1,..,n\}}g(j))^2}} \quad (1)$$

We use a median-based $z - score$ rather than a mean-based as we want to detect unusually high and unusually low gene expression levels during the yeast cell cycle. In contrast to the $z - score_{mean}$ a $z - score_{median}$ is more robust w.r.t. these extremes and better suited for detecting them, as validated in initial experiments (not shown here).

- **H**igh: $g(i)$ is significantly higher than the median expression level of g ($z - score_{median}(g(i)) \geq 2$).
- **M**edium: $g(i)$ does not significantly differ from the median expression level of g (-2 $< z - score_{median}(g(i)) <$ 2).
- **L**ow: $g(i)$ is significantly lower than the median expression level of g ($z - score_{median}(g(i)) \leq$ -2).

For each edge e in our dynamic graph, we compare the expression profiles of its adjacent genes g_1 and g_2 to generate the edge existence string according to the following rules:

- $\ell(e(i))$ = 'H' if $g_1(i)$ and $g_2(i)$ both show a high gene expression level.
- $\ell(e(i))$ = 'M' if $g_1(i)$ and $g_2(i)$ both show a medium gene expression level.
- $\ell(e(i))$ = 'L' if $g_1(i)$ and $g_2(i)$ both show a low gene expression level.
- $\ell(e(i))$ = 'N' if $g_1(i)$ and $g_2(i)$ show different gene expression levels.

In this manner, we generate edge existence strings from the alphabet $\Sigma = \{H, L, M, N\}$ for pairs of interacting genes in the yeast PPI network.

5.2 Enumerating all Static Frequent Subgraphs

In order to compute exact frequencies for static frequent sugraphs, we exhaustively enumerate all frequent subgraphs of our dynamic network. This step is performed using the FANMOD tool which implements the algorithm by [19]. As our dynamic network is a large graph, we have to limit ourselves to subgraphs of size 3 and 4 as previous studies [20], but this allows us to guarantee that we are not missing out on dynamic frequent subgraphs.

5.3 Significance of Dynamic Patterns

To assess the significance of a static frequent subgraph with frequency t, we use p-values for each static frequent subgraph according to [19]. The p-value represents the probability of this static frequent subgraph occurring at least t times in a random graph with identical degree distribution. We only retain those static frequent subgraphs whose p-value is below the significance level of $\alpha = 0.025$. We also assess the significance of the common dynamic string pattern σ of a dynamic frequent subgraph D which occurs in u out of t embeddings of the static frequent subgraph S in terms of a p-value. For this purpose, we compute the probability p of σ to occur in an embedding of S by chance. Let $M = M(i,j)_{l'\times n'}$ be the matrix representation of σ, where each row corresponds to one edge (existence string) and each column to one time step. We then define the probability of σ occurring by chance as

$$p_\sigma = \prod_{i=1}^{l'}\prod_{j=1}^{n'} p_i(M(i,j)) \quad (2)$$

where p_i is the background probability of the character represented by $M(i,j)$ occurring in the i-th edge of S. In other terms, the random model assumes that the existence strings of the embeddings of S were randomly generated. Under this model, the probability of σ appearing u times (denoted $|\sigma| = u$) in the t embeddings of S then follows a binomial distribution:

$$P(|\sigma| = u) = \binom{t}{u} p_\sigma^u (1 - p_\sigma)^{t-u}, \quad (3)$$

and the p-value of the common dynamic pattern σ can be computed straightforwardly:

$$p\text{-value}(|\sigma| = u) = P(|\sigma| \geq u) \quad (4)$$

We deem common dynamic patterns significant, if their p-value is below 0.025. Dynamic frequent subgraphs are considered significant if their associated static frequent subgraph is significant and if its common dynamic pattern is significant as well. We only retained those dynamic frequent subgraphs that occur in at least 50% of the embeddings of the corresponding static frequent subgraph.

5.4 Evolutionary Conservation Rate: A Quality Criterion for Dynamic Frequent Subgraphs

Following previous biological analysis [18], we deem dynamic frequent subgraphs the more *conserved*, the more proteins that participate in embeddings of this subgraph have identifiable orthologs in other organisms. We analyse the

	$k=3$			$k=4$		
	Static	**Dynamic**	**Static only**	**Static**	**Dynamic**	**Static only**
# frequent subgraphs	367	515		3,293	2,111	
# proteins	2,302	1,787	515	2,403	1,770	633
# conserved proteins	577	477	100	591	470	121
conservation rate	0.2507	0.2669	0.1942	0.2459	0.2655	0.1912

Table 1: Evaluation w.r.t. conservation rate of frequent subgraphs with three and four nodes.

conservation rate of yeast proteins by searching for orthologs in each of five other higher eukaryotic organisms for which information is available in the InParanoid database [17]: 644 of the 3,607 proteins in our dynamic network turned out to be conserved across all five species. Hence the probability of observing a conserved protein by uniform random sampling from the PPI network is 0.1785, of observing a set of three conserved proteins is 0.0057, and of observing a set of four conserved proteins is 0.0010.

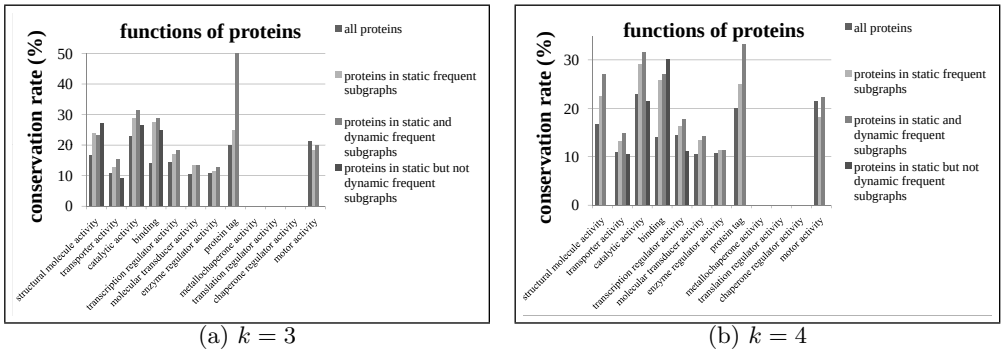
5.5 Conservation Rate among Static and Dynamic Frequent Subgraphs

Our static frequent subgraph discovery results in 367 subgraphs with three nodes and within these 515 dynamic patterns that were both frequent and significant. We find 2,302 proteins to be members of at least one static frequent subgraph, out of which 577 are conserved across species (conservation rate 0.2507). Hence the rate of conservation among proteins that are part of static frequent subgraphs is significantly higher than among proteins that are sampled randomly from our dynamic network (Binomial distribution $B(t = 2,302; p = 0.1785)$; p-value < 0.0001). 1,787 proteins are members of at least one dynamic frequent subgraph, including 477 conserved proteins (conservation rate 0.2669). Hence the rate of conservation among proteins that are part of dynamic patterns is significantly higher than among proteins that are sampled randomly from our dynamic network (Binomial distribution $B(t = 1,787; p = 0.1785)$; p-value < 0.0001). 515 proteins are members of at least one static frequent subgraph, but of none of the dynamic patterns (100 of these are conserved, conservation rate 0.1942). While the rates of conservation among proteins from static and dynamic frequent subgraphs are not significantly different (two-sample t-test for unequal variances, $p = 0.2392$), we observed that proteins that participate in static *and* dynamic frequent subgraphs are significantly stronger conserved than those that are members of static, *but not* of dynamic static subgraphs (two-sample t-test for unequal variances, $p = 0.004$).

We discover 3,293 static frequent subgraphs of size $k = 4$. Within these subgraphs we find 2,111 dynamic patterns that were both frequent and significant. 2,403 proteins are members of at least one static frequent subgraph, out of which 591 are conserved (conservation rate 0.2459). Consistent with our results on frequent subgraphs with three nodes the rate of conservation among proteins that are part of static frequent subgraphs is significantly higher than among proteins that are sampled randomly from our dynamic network (Binomial distribution $B(t = 2,403; p = 0.1785)$; p-value < 0.0001). 1,770 proteins are members of at least one dynamic pattern, including 470 conserved proteins (conservation rate 0.2655). Analogously to the frequent subgraphs with three nodes the rate of conservation among proteins that are part of dynamic patterns is significantly higher than among proteins that are sampled randomly from our dynamic network (Binomial distribution $B(t = 1,770; p = 0.1785)$; p-value < 0.0001). 633 proteins are members of at least one static frequent subgraph, but of none of the dynamic patterns (121 of these are conserved, conservation rate 0.1912). While the rates of conservation among proteins from static and dynamic frequent subgraphs with four nodes are not significantly different (two-sample t-test for unequal variances, $p = 0.1525$), we observed that proteins that participate in static *and* dynamic frequent subgraphs are significantly stronger conserved than those that are members of static, *but not* of dynamic ones (two-sample t-test for unequal variances, $p = 0.001$). Therefore the significance of the conservation rate of frequent subgraphs with four nodes is even higher than for subgraphs with three nodes. Table 1 summarizes the results of these experiments.

5.6 Conservation of Dynamic Frequent Subgraphs across GO Classes

We also check whether proteins that are part of static and dynamic frequent subgraphs show different levels of conservation depending on their molecular function as defined by GO. As can be seen from Figure 3, conservation rates vary widely among different functional classes, both for background conservation rate, and among dynamic and static frequent subgraphs. The figure shows the conservation rates for proteins that are participating in frequent subgraphs with three and four nodes w.r.t. 12 different classes of protein functions. For each function the general conservation rate of proteins among this class is depicted. The second, third and fourth bars of the histogram stand for the conservation rate of proteins involved in static frequent subgraphs, in dynamic patterns (therefore also in static frequent subgraphs) and the conservation rate of proteins that play a role in static frequent subgraphs only. It can be seen that frequent subgraphs with three and four nodes both show differences in their conservation rate dependig on the protein functions. For example, proteins with structural molecule activity or proteins regulating transcription participating only in static frequent subgraphs are highly conserved in subgraphs with three nodes only. Proteins that have metallochaperone activity or regulate translation or chaperones are not conserved, neither in static nor in dynamic patterns.



(a) $k = 3$ (b) $k = 4$

Figure 3: Evaluation w.r.t. protein function of frequent subgraphs with three and four nodes.

5.7 Selected Frequent Subgraphs

The most frequent and significant ($p < 0.0001$) subgraph with three nodes in our dynamic network has 4,060 embeddings and is depicted in Figure 4(a). Two proteins share the same biological function *catalytic activity*, whereas the third is a transporter protein. This is not surprising as transporter proteins catalyze the transfer of a substance from one side of a membrane to the other. Within these 4,060 embeddings we detect 1,728 dynamic patterns comprising two time points. Figure 4(b) shows a significant ($p < 0.0001$) static frequent subgraph with three embeddings, but in two of them we detect a dynamic pattern that has identical behaviour of co-expression over a period of 16 out of 17 time points. The corresponding interacting proteins are organized under *binding*, *enzyme regulator activity* and *motor activity* term of GO, respectively. This static frequent subgraph has quite few embeddings, but includes the longest dynamic pattern within static frequent subgraphs of size 3.

The most frequent subgraph of size 4 has 18,056 embeddings (cf. Figure 4(c)). Two proteins show catalysis properties whereas the other two are binding proteins. 8,511 out of these embeddings have dynamic patterns comprising two time points. In Figure 4(d) the most significant ($p < 0.0001$) frequent subgraph has three embeddings, whereof two have a dynamic pattern over a time period of 15 time steps. This indicates that this subgraph shows identical temporal behaviour over almost the whole time course. In the first time step both the binding protein and the transportation protein show very low expression levels.

6. CONCLUSION

In this paper, we have presented a framework for frequent dynamic subgraph discovery in dynamic graphs. We have shown how to efficiently compute these patterns using suffix trees. Furthermore, we have described how to combine frequent subgraph mining algorithms with our dynamic framework and we have applied our framework on a large real-world case study to handle dynamic graphs. Dynamic graph mining is a technique that can be used in many fields of applications. Finding dynamic patterns in PPI maps promises to reveal interesting insights into biological processes. Similarly, dynamics in social networks could contain interesting patterns, which could be discovered by dynamic graph mining. Furthermore, telecommunication logs could be examined to try to uncover special dynamic patterns. We intend to explore these applications in future research.

7. REFERENCES

[1] J. Aach, W. Rindone, and G. M. Church. Systematic Management and Analysis of Yeast Gene Expression Data. *Genome Res.*, 10:431–445, Feb 2000.

[2] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet.*, 25:25–29, 2000.

[3] K. M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther. Pattern Mining in Frequent Dynamic Subgraphs. In *ICDM*, pages 818–822, 2006.

[4] R. Cho, M. Campbell, E. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T. Wolfsberg, A. Gabrielian, D. Landsman, D. Lockhart, and R. Davis. A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol. Cell*, 2:65–73, Jul 1998.

[5] D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *JAIR*, 1:231–255, 1994.

[6] P. Desikan and J. Srivastava. Mining Temporally Evolving Graphs. In *WebKDD Workshop*, 2004.

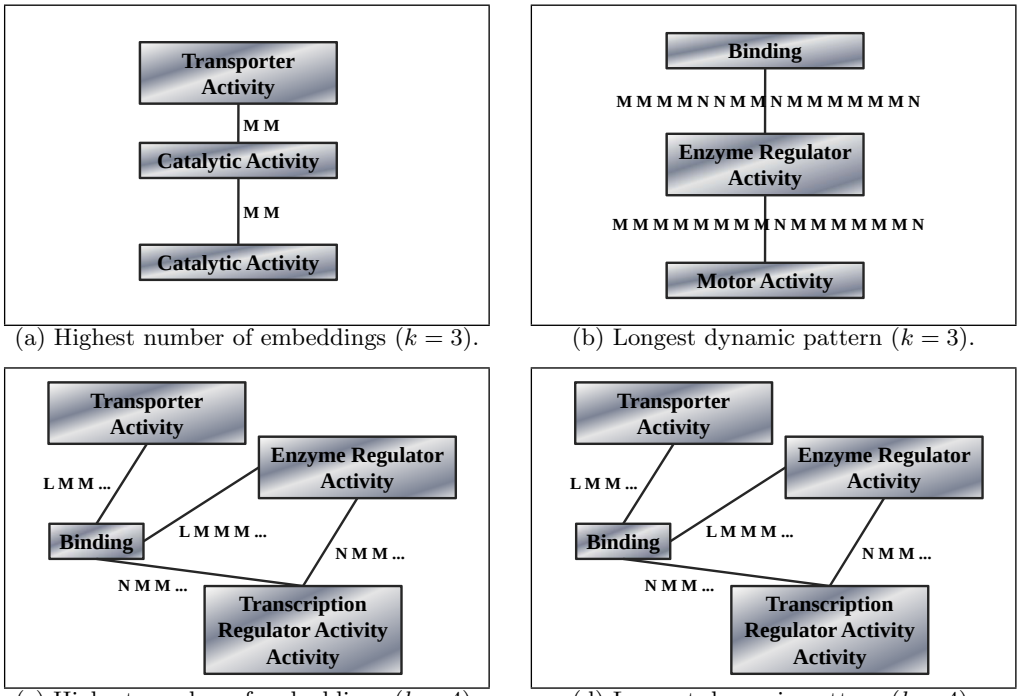
[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[8] E. L. Hong, R. Balakrishnan, Q. Dong, K. R. Christie, J. Park, G. Binkley, M. C. Costanzo, S. S. Dwight, S. R. Engel, D. G. Fisk, J. E. Hirschman, B. C. Hitz, C. J. Krieger, M. S. Livstone, S. R. Miyasato, R. S. Nash, R. Oughtred, M. S. Skrzypek, S. Weng, E. D. Wong, K. K. Zhu, K. Dolinski, D. Botstein, and J. M. Cherry. Gene Ontology annotations at SGD: new data sources and annotation methods. *Nucleic Acids Res.*, 36:D577–81, 2008. PMID: 17982175.

[9] L. C. K. Hui. Color set size problem with application to string matching. In *CPM*, pages 230–243, 1992.

[10] A. Inokuchi, T. Washio, and H. Motoda. Complete Mining of Frequent Patterns from Graphs: Mining Graph Data. *Machine Learning*, 50(3):321–354, 2003.

[11] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.



(a) Highest number of embeddings ($k = 3$).
(b) Longest dynamic pattern ($k = 3$).
(c) Highest number of embeddings ($k = 4$).
(d) Longest dynamic pattern ($k = 4$).

Figure 4: Selected frequent subgraphs that consist of three or four nodes.

[12] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *ICDM*, pages 313–320, 2001.

[13] M. Kuramochi and G. Karypis. Finding Frequent Patterns in a Large Sparse Graph. In *SDM*, 2004.

[14] M. Kuramochi and G. Karypis. GREW-A Scalable Frequent Subgraph Discovery Algorithm. In *ICDM*, pages 439–442, 2004.

[15] M. Lahiri and T. Y. Berger-Wolf. Structure Prediction in Temporal Networks using Frequent Subgraphs. In *CIDM*, pages 35–42, 2007.

[16] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*, pages 177–187, 2005.

[17] M. Remm, C. E. V. Storm, and E. L. L. Sonnhammer. Automatic Clustering of Orthologs and In-paralogs from Pairwise Species Comparisons. *J. Mol. Biol.*, 314:1041–1052, Oct 2001.

[18] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of Escherichia coli. *Nat Genet.*, 31(1):64–68, May 2002.

[19] S. Wernicke. Efficient Detection of Network Motifs. *TCBB*, 3(4):347–359, 2006.

[20] S. Wuchty, Z. N. Oltvai, and A. L. Barabasi. Evolutionary conservation of motif constituents in the yeast protein interaction network. *Nat Genet.*, 35(2):176–179, Oct 2003.

[21] I. Xenarios, L. Salwinski, X. J. Duan, P. Higney, S. M. Kim, and D. Eisenberg. DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions. *Nucleic Acids Res.*, 30(1):303–305, Jan 2002.

[22] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*, pages 721–724, 2002.