In Proc. of ICDM 2006, Hong Kong

Pattern Mining in Frequent Dynamic Subgraphs

Karsten M. Borgwardt, Hans-Peter Kriegel, Peter Wackersreuther
Institute of Computer Science
Ludwig-Maximilians-Universität
Munich, Germany
kb|kriegel|wackersr@dbs.ifi.lmu.de

Abstract

Graph-structured data is becoming increasingly abundant in many application domains. Graph mining aims at finding interesting patterns within this data that represent novel knowledge. While current data mining deals with static graphs that do not change over time, coming years will see the advent of an increasing number of time series of graphs. In this article, we investigate how pattern mining on static graphs can be extended to time series of graphs. In particular, we are considering dynamic graphs with edge insertions and edge deletions over time. We define frequency in this setting and provide algorithmic solutions for finding frequent dynamic subgraph patterns. Existing subgraph mining algorithms can be easily integrated into our framework to make them handle dynamic graphs. Experimental results on real-world data confirm the practical feasibility of our approach.

1 Introduction

Graphs are the universal data structure to model entities and their relationships. All common data types, such as vectors, strings and time series, can be modeled as graphs. Consequently, it is not surprising that the amount of graph data is ever increasing in domains of large database management, from bioinformatics to social network analysis. Hence pattern recognition algorithms on graphs are of utmost importance for increasing our understanding of the information represented by these large datasets of graphs. One of the central questions in graph data mining is finding frequent patterns, i.e. subgraphs that occur frequently in graph data. Previous research in frequent subgraph mining has focused on two problems: first, finding frequent subgraphs across a dataset of graphs [5, 7, 10]; second, finding frequent subgraphs within one single large graph [1, 9, 8].

From an application-oriented view, both have in common that they try to find frequent patterns within static relations between objects. However, to study interactions in real-world systems it is more adequate to look at temporal interactions, as relations between objects in many real-world systems usually occur for a certain period of time only. For example, proteins interact temporally, emails are sent at a certain point of time, people communicate at certain times. To make one of these examples explicit: If you want to understand the dynamics of email communication in a group of people, it is much more interesting to study who wrote whom at which point of time, than just looking at who wrote whom at all, ignoring the temporal information.

For this reason, we want to define and tackle a novel third problem of frequent graph mining in this paper: frequent pattern mining on dynamic graphs. We are interested in subgraphs that are both a) (topologically) frequent within a large graph and b) that show an identical dynamic behavior over time. 'Dynamic behavior' refers to the fact that insertions and deletions of edges between their nodes occur in the same order over time.

While the evolution of graphs over time has been addressed before, these studies dealt with topics such as densification and shrinking diameters of real-world graphs over time and do not define pattern mining algorithms on these dynamic graphs. Only in the domain of web mining, the importance of dynamic graph pattern mining has been mentioned before [2], yet a theoretical framework has not been defined so far.

2 Frequency in dynamic graphs

In this section, we will define dynamic graphs and the type of patterns that we are interested in within these graphs.

2.1 Graph theory and graph mining

Some basic terminology from graph theory is required to follow our description, which we define in utmost brevity here. 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 S is a **subgraph** of G if its vertices and edges are subsets of those of G. These subsets are then referred to as an **embedding** of S in G. S is a **frequent subgraph** of G if G contains more than t embeddings of S, where t is a predefined threshold. Two graphs G_1 and G_2 are **isomorphic** if there exists a bijection between their nodes and edges. Edges that are mapped to each other by this bijection will be referred to as *corresponding edges* in this article.

2.2 Dynamic Graphs

We are now in a position to define 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.*

Such a time series of graphs can be transformed into a dynamic graph as follows:

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 $DG(G_{ts}) = (V_{DG}, E_{DG}, es)$, where $V_{DG} = V_i$ for all $1 \leq i \leq n$ and $E_{DG} = \cup_{i=1}^{n} E_i$. The mapping $es : E_{DG} \rightarrow \{0 \mid 1\}^n$ maps each edge e in E_{DG} to a binary string $es(e)$ of length n. The i-th character of $es(e)$ is 1 if e exists in state i of G_{ts}, and 0 if e does not exist in state i of G_{ts}. $es(e)$ is referred to as the* **existence string** *of edge e.*

We are interested in subgraphs of such a graph, namely topological subgraphs and dynamic subgraphs, which we will define next. We consider graphs with node labels L, but our results can easily be extended to edge-labeled graphs as well.

Definition 3 (Topological subgraph) *Let $DG_1 = (V_1, E_1, L_1, es_1)$ and $DG_2 = (V_2, E_2, L_2, es_2)$ be node-labeled dynamic graphs. DG_1 is a topological subgraph of DG_2 ($DG_1 \subseteq DG_2$) if the following conditions hold: $V_1 \subseteq V_2$, $E_1 \subseteq E_2$, $L_1 = L_2$, $es_1(e_1) = es_2(e_2)$ for all corresponding edges e_1 in E_1 and e_2 in E_2. If DG_1 is a* **topological subgraph** *of a dynamic graph DG_2, then DG_2 contains DG_1 topologically, or in other terms, DG_2 is a topological supergraph of DG_1.*

Hence this definition of topological subgraph is very close to the classic definition of subgraph for static graphs, except for the fact that existence strings are now considered as well. Dynamic subgraphs — which we will define next — require some terminology on the similarity of strings.

Definition 4 (Substring) *Given two strings s_1 and s_2 of length k and n, respectively. For $1 \leq i \leq n - (k-1)$, we then denote by $substr(s_2, i, i + (k-1)) = s_1$ the fact that s_1 is a* **substring** *of s_2, starting at the i-th character in s_2 and ending at the i+(k-1)-th character in s_2. In short, s_1 is a k-length substring of s_2, starting at character i.*

We can now formalize our notion of a dynamic subgraph:

Definition 5 (Dynamic subgraph of length k) *A graph $DSG_1 = (V_1, E_1, L_1, es_1, start_1)$ is a* **dynamic subgraph of length k** *of a dynamic graph $DG_2 = (V_2, E_2, L_2, es_2)$ with n states ($DSG_1 \subseteq_{k\_dynamic} DG_2$) if the following conditions hold: $V_1 = V_2$, $E_1 = E_2$, $L_1 = L_2$, and for all corresponding edges e_1 and e_2 from E_1 and E_2: $es_1(e_1) = substr(es_2(e_2), start_1, start_1 + (k-1))$, where $start_1$ is the state of DG_2 in which DSG_1 starts, with $1 \leq start_1 \leq n - (k-1)$*

Note the difference between a *dynamic subgraph* and a *topological subgraph* of a dynamic graph: A topological subgraph is a subgraph in the classic sense, it contains the same number of states n as its dynamic supergraph, but a subset of its nodes. A dynamic subgraph contains the same set of nodes and edges, but only a sub-interval of k out of the n states of its dynamic supergraph.

The next two definitions will state under which conditions we consider such a dynamic subgraph to be frequent within a given dynamic graph.

Definition 6 (Dynamic embedding) *A dynamic subgraph $DSG_1 = (V_1, E_1, L_1, es_1, start_1)$ has a* **(dynamic) embedding** *in a dynamic graph DG_2 if there exists a topological subgraph S of the dynamic graph DG_2 such that $DSG_1 \subseteq_{k\_dynamic} S$.*

To illustrate this definition: DSG_1 is dynamically embedded in DG_2 if it is a dynamic subgraph of a topological subgraph of DG_2.

Definition 7 (Frequent dynamic subgraph) *A dynamic subgraph $DSG_1 = (V_1, E_1, L_1, es_1, start_1)$ is a* **frequent dynamic subgraph (FDS)**[1] *of a dynamic graph DG_2 if DG_2 contains at least t identical embeddings of DSG_1, where t is a user-defined frequency threshold parameter.*

One part of the above definition has not been specified so far, namely when we deem two dynamic subgraphs identical. Depending on how we define this identity, we can study different types of FDS.

[1] FDS is short for frequent dynamic subgraph(s) in this article.

2.3 Types of frequent dynamic subgraphs

We define two alternative versions of identity among dynamic subgraphs next. The first is based on the idea that all embeddings of a dynamic subgraph in a dynamic graph DG must occur synchronously, i.e. start and end at identical states of DG. The second is based on the idea that embeddings of the same dynamic subgraph can be asynchronous, i.e. the dynamic subgraphs must be identical, but they can start at different states of the dynamic graph DG. These two types of FDS are formalized in the following definitions.

Definition 8 (Equivalence of dynamic subgraphs) *Two dynamic subgraphs are* ***equivalent*** *if* $DSG_1 = (V_1, E_1, L_1, es_1, start_1)$ *is isomorphic to* $DSG_2 = (V_2, E_2, L_2, es_2, start_2)$*, and* $es_1 = es_2$*. If* e_1 *and* e_2 *are corresponding edges from* E_1 *and* E_2*, respectively, then* $es_1(e_1) = es_2(e_2)$ *is referred to as the (common)* **edge pattern** *of* e_1 *and* e_2*.*

Definition 9 (Pairwise identity of dynamic subgraphs) *Two dynamic subgraphs* DSG_1 *and* DSG_2 *are* **inter-synchronous identical***, if* DSG_1 *and* DSG_2 *are equivalent and* $start_1 = start_2$*.*

Two dynamic subgraphs DSG_1 *and* DSG_2 *are* **inter-asynchronous identical***, if* DSG_1 *and* DSG_2 *are equivalent and* $start_1 \neq start_2$*.*

Within these classes of FDS, we can further distinguish between *fixed* patterns and *changing* patterns. Fixed patterns contain existence strings consisting of all 1's only, whereas changing patterns allow for existence strings that contain at least one 1 and an arbitrary number of 0's.

2.4 Finding edge patterns via suffix trees

Given a set of corresponding edges from a set of topological frequent subgraphs, frequent edge patterns can be detected via suffix trees in a very efficient manner. It is well-known from [4] that common frequent substrings in a set of strings can be determined for all arbitrary frequency thresholds in time linear in the total added length of the strings, i.e. the time it takes to read all strings. We refer to this algorithm as the *Common Frequent Substring (CFS)* algorithm in the following (see [3] for further details).

2.5 Finding single-edge FDS

Using the efficiency of suffix trees, we are now able to detect frequent dynamic subgraphs consisting of one single edge.

First of all, edges in dynamic graphs can only be (topologically) frequent if their type is frequent. 'Type' is referring to the concatenation of the label of the source node and that of the target node of this edge. This, however implies that edges can only be frequent when adjacent to nodes with frequent type, i.e. frequent node labels. In short, to find frequent edges, the first step is to find all frequent node types and the second step is to find all frequent edge types.

Up to this point, the edge existence strings are not considered at all, i.e. we only care about topological frequency. After determining frequent edge types, the frequent common edge patterns for those are computed, which means that we are looking at dynamic frequency next.

Finding asynchronous common edge patterns Asynchronous frequent common edge patterns can be found by applying the CFS algorithm to the edge existence strings of a set of edges of the same type.

Finding synchronous common edge patterns Finding synchronous frequent common edge patterns cannot be performed in exactly the same fashion as for asynchronous edges, as one has to consider the time interval in which existence strings occur as well. This problem can be overcome by a simple trick: a timestamp is added to each character in the existence strings, which represents the time point each character is derived from. For example, the existence string "0 0 1 1 0 1 0 1" is transformed to "(0 1) (0 2) (1 3) (1 4) (0 5) (1 6) (0 7) (1 8)". If this is done for all existence strings, then the CFS algorithm — treating terms in brackets as one single character — can be directly applied to find synchronous common frequent substrings.

2.6 Finding larger FDS

After detecting FDS comprising one edge, one next wants to search for FDS of more than one edge, i.e. larger FDS. To find larger frequent dynamic subgraphs in an apriori-like fashion, one has to construct candidates for larger frequent dynamic subgraphs by joining smaller frequent dynamic subgraphs already determined. One then has to evaluate each of the candidates to check if it is a real FDS.

The central step in this iterative process is to evaluate whether the union of an edge with frequent edge type e and a frequent subgraph S is a frequent dynamic subgraph itself. For this purpose, let us denote the set of isomorphic adjacent occurrences of S and e by $C(S, e) = \{(S_1, e_1), (S_2, e_2), \ldots, (S_l, e_l)\}$, the set of all edges of type e in $C(S, e)$ by $e_C = \{e_1 \ldots, e_l\}$ and the set of all subgraphs of type S in $C(S, e)$ by $S_C = \{S_1 \ldots, S_l\}$. We now define algorithms to detect our 2 types of FDS, namely inter-synchronous and inter-asynchronous FDS.

Inter-synchronous FDS These FDS appear synchronously. Consequently, all edge patterns have a common start and end time point i and $i+(k-1)$, respectively. If S_C

is such an inter-synchronous FDS, then the union of S_C and e_C can only be such an FDS, if e_C contains an edge pattern that is synchronous to the edge patterns in S_C. To detect these edge patterns, one considers all substrings of existence strings of edges in e_C for the interval $[i; i + (k - 1)]$ of S_C only. Via CFS, one determines whether these substrings are frequent in e_C. In detail, one computes the longest common frequent substring and checks whether its length is k. If not, then there is no frequent substring pattern in e_C and hence $C(S, e)$ is evaluated not to be an FDS.

Inter-asynchronous FDS These FDS appear asynchronously. For each pair (S_j, e_j) in $C(S, e)$, consisting of an embedding of the FDS S and a frequent edge e, one determines the start point i and the end point $i + (k - 1)$ of S_j. One then extracts the substring for the interval $[i; i+(k-1)]$ from the existence string of e_j. CFS can afterwards be applied to these substrings. If a frequent substring of length k is encountered, the candidate $C(S, e)$ is a real FDS.

Fixed and changing FDS To detect fixed and changing edge patterns, one first prunes edge patterns that do not contain at least one 1; these are the changing edge patterns. To find fixed FDS within those, one prunes frequent edge patterns that contain 0's.

2.7 Dynamic GREW

Based on our definitions from the previous sections, the extension of frequent subgraph mining (FSM) algorithms to dynamic graphs is straightforward. One has two options: Either one first uses an FSM algorithm to detect all frequent topological subgraphs and then searches for dynamic patterns within those, or one selects dynamic patterns from frequent subgraphs in each iteration of an FSM algorithm. We exemplify the latter approach for the FSM algorithm GREW [9]. We explain this extension of GREW to dynamic graphs ("Dynamic GREW") following the pseudo-code in Table 1.

In the first step (1) of Dynamic GREW, we search for edges whose edge type is frequent within the dynamic graph (neglecting edge existence strings). Among these, we first make sure that embeddings of candidates do not overlap by applying a greedy maximal independent set algorithm (2). Then we count non-overlapping occurrences of each candidate (3). Finally, we check if these embeddings constitute a FDS (4), and if so, we mark these embeddings and rewrite the graph such that each marked embedding of this FDS is represented by one super-node (5). These five steps are iterated (6), as long as at least one candidate is marked in each iteration.

Table 1. Dynamic GREW

```
INPUT:
o Dynamic graph DG
o t = frequency threshold
o k = length of patterns

1) Candidate Generation:
find edges that occur more than
t times in DG
2) Employ Greedy Maximal Independent
set algorithm on overlap graph
for candidates -> DG*
3) Candidate Evaluation 1:
Check if there are more than t
occurrences of each candidate in DG*
4) Candidate Evaluation 2:
Check if candidate fulfills criterion
to be an FDS:
if yes, mark embeddings of this candidate
5) Rewrite DG such that each embedding
of each marked candidate
is represented by one super-node
6) if no candidates were marked : end;
otherwise go to 1

OUTPUT: super-nodes = FDS discovered
```

3 Experiments

To check the practical feasibility of our approach, we ran experiments on real-world data. Due to space restrictions, we only present results on one of those, the ENRON email traffic graph.

ENRON email traffic graph The ENRON dataset comprises records of the email traffic between employees of the ENRON company [6]. For each email, the dataset provides information about a) the sender, b) the recipient(s), and c) the time point of sending.

We turned this dataset into a time series of graphs. Nodes in these graphs are employees of ENRON. Nodes are labeled by the rank of the corresponding person within the company. The time interval in which email traffic was recorded is split into 15 equal-sized bins. For each bin, we create one graph, with the i-th bin representing the i-th state of our graph time series. Nodes in the i-th state of the time series are connected if an email was sent between these two persons in this time interval. As a result, we obtain a dynamic graph with 130 nodes and approx. 273 edges per time step. Within this dynamic graph, we tried to detect FDS in ENRON's email traffic.

Table 2. Frequency of FDS in ENRON

Class	Number of FDS	Size of these FDS
fixed synchr.	22	21(1), 1(2)
fixed asynchr.	24	21(1), 2(2), 1(3)
changing synchr.	18	16(1), 1(2), 1(3)
changing asynchr.	21	16(1), 4(2), 1(3)

Parameters: fixed t=10, k=3; changing t=20, k=12; (synchr. = synchronous, asynchr. = asynchronous; x(y) means that x FDS with y edges were discovered)

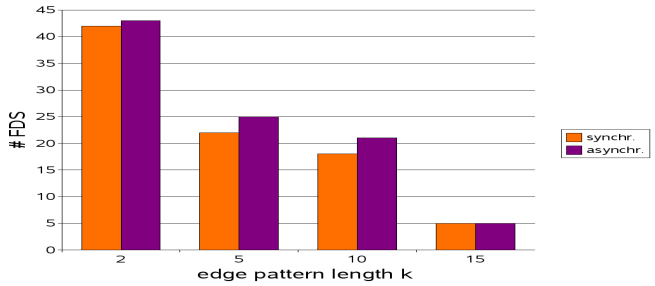

Figure 1. Number of FDS vs. length of edge patterns k ($t = 20$, changing patterns)

Results First, we searched for both types of fixed FDS using Dynamic GREW with default parameters of t=10 and k=3; we also searched for changing patterns with default parameters of t=20 and k=12. Note that — enhanced by the extreme sparsity of the ENRON graph — fixed patterns are quite rare, requiring a lower threshold and length parameter than the changing patterns. The number of changing and fixed FDS discovered for these parameters are reported in Table 2.

Next, we examined the impact of the threshold parameter t and the edge pattern length k on the number of patterns detected. We tested values for k in $\{2, 5, 10, 15\}$ with t constant as $t = 20$ for changing FDS, and for t in $\{10, 20, 30, 40\}$ with k constant as $k = 12$ for changing FDS. Results are illustrated in Figures 1 and 2. As expected, shorter edge patterns and lower thresholds lead to Dynamic GREW detecting more FDS.

This experiment confirms that asynchronous patterns are more frequent than synchronous patterns, as they are not 'attached' to one common starting point. Hence one should search for asynchronous FDS with higher choices of t than those selected for synchronous patterns. Unsurprisingly, frequent shorter edge patterns are more abundant than long edge patterns. The shorter k is chosen, the higher the frequency threshold should be set.

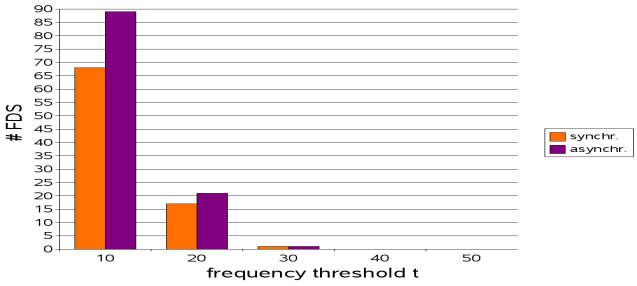

Figure 2. Number of FDS vs. frequency threshold t ($k = 12$, changing patterns)

4 Discussion and Conclusions

In this paper, we have extended frequent subgraph mining algorithms to time series of graphs. In particular, we are looking for subgraphs that are topologically frequent within a large graph and that show insertions and deletions of edges in the same temporal order.

Our technique might be used to study frequent motifs in protein-protein interaction dynamics, as well as in social or telecommunication networks.

References

[1] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res. (JAIR)*, 1:231–255, 1994.

[2] P. Desikan and J. Srivastava. Mining temporally evolving graphs. In *WebKDD Workshop*, 2004.

[3] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, June 1997. ISBN 0–521–58519–8.

[4] L. Hui. Color set size problem with applications to string matching. In *Proc. 3rd Symp. on Combinatorial Pattern Matching*, volume 664, pages 227–40, 1992.

[5] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[6] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *ECML*, pages 217–226, 2004.

[7] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

[8] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *SDM*, 2004.

[9] M. Kuramochi and G. Karypis. Grew-a scalable frequent subgraph discovery algorithm. In *ICDM*, pages 439–442, 2004.

[10] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.