Mining Interesting Patterns and Rules in a Time-evolving Graph

Yuuki Miyoshi*, Tomonobu Ozaki†, and Takenao Ohkawa‡

***Abstract*—Time-evolving graphs, i.e. dynamic networks changing their structures with time, are becoming ubiquitous recently. A typical example is an email communication network whose vertex corresponds to an individual and whose edge corresponds to an email communication within a time period. For the effective analysis of such time-evolving graphs, it must be important to utilize representative local structures in networks as well as time information on edge formation simultaneously.**

In this paper, we consider a problem of mining frequent patterns and valid rules representing graph evolutions or structural changes in a network with time information. In addition to an effective mechanism for extracting representative patterns and rules, we devise graph-based summarization of discovered rules. By using certain measures provided by the summary, we can expect to find more interesting information that are difficult to obtain in the traditional support and confidence framework. The effectiveness of the proposed framework was confirmed by preliminary experiments using real world email data.

***Index Terms*—frequent pattern mining, graph mining, graph evolution rule**

I. Introduction

Recently, the study of human activities on social and communication networks attract a large amount of attention. In general, patterns on local structures in those networks can provide useful insight for understanding human activities. Furthermore, because a network changes its structure dynamically with time, taking account of temporal information on edge formation will help precise analysis. In this paper, we consider the problem of mining frequent patterns and valid rules that represents *graph evolutions*, i.e. structural changes in a network with time information.

Two types of graph evolutions are discussed. One is an evolution by temporal aspect, and the other is by the surrounding situation. These evolutions are explained with a simple example shown in Fig. 1. In this figure, we assume that each vertex corresponds to a person and each edge represents a communication between persons. The number associated to an edge represents a time period when the communication is conducted. In Fig. 1(a), P_1 and P_2 represent two graph evolutions with different time periods after the communication between A and B. The order and timing of communications contained in P_1 and P_2 will give useful insight to understand the evolutions. On the other hands, Fig. 1(b) represents an evolution influenced by the surrounding situation. P_4 states that a triangle (P_3) causes an communication between A and D while we do not care how the triangle is constructed. This kind of information must be useful to understand what structure acts as a trigger for graph evolutions.

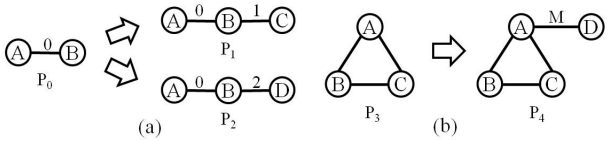

Fig. 1. two types of graph evolutions

Previous researches on graph pattern discovery have drawback on analyzing graph evolutions. First, other than a few exceptions, most graph miners can not handle time information directly. Second, due to a huge amount of extracted patterns and rules, it is difficult to determine important and significant ones. To overcome these difficulties, in this paper, we propose a framework for discovering interesting patterns and rules on graph evolutions. The proposed approach is summarized in Fig. 2. We first mine frequent patterns with a time information ((1) in Fig. 2). Frequent pattern miners often discover a huge amount of patterns. In order to compress a set of frequent patterns into a representative set, we adopt a structural representative approach proposed in [4] ((2) in Fig. 2). Then, the rules focused on the evolutions will be generated from the obtained patterns ((3) in Fig. 2). We devise two graph-based summarizations of a set of rules to help to discover interesting and important patterns ((4) in Fig. 2). Finally, by investigating these summarizations, we extract characteristic patterns and chains of rules based on some network-based importance measure such as degree centrality ((5) in Fig. 2). The aim of employing network-based measure is to find interesting patterns which are difficult to discover in the standard framework based on support and confidence.

The rest of this paper is organized as follows. In section II, basic notations and definition will be introduced. In section III, our framework for discovering patterns and rules in time-evolving graphs will be explained in detail. Preliminary experimental results are reported in section IV. After describing related work in section V, finally we conclude this paper in section VI.

II. Notations and Definitions

According to the previous studies[1], [4], [3], we give formal notations and definitions on graphs, patterns and rules with time information.

A. Patterns and Rules in a Time-evolving Graph

A *time-evolving graph* $G = (V_G, E_G, l_G, t_G)$ on a set of labels $\mathcal{L}$ and a set of time-stamps $\mathcal{T}$ is represented as a

*Graduate School of Engineering, Kobe University, 1-1 Rokkodai, Nada, Kobe 657-8501, Japan. Email: miyoshi@cs25.scitec.kobe-u.ac.jp

†Cybermedia Center, Osaka University, 1-32 Machikaneyama, Toyonaka, Osaka 560-0043, Japan. Email: tozaki@dcm.cmc.osaka-u.ac.jp

‡Graduate School of System Informatics, Kobe University, 1-1 Rokkodai, Nada, Kobe 657-8501, Japan. Email: ohkawa@kobe-u.ac.jp

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

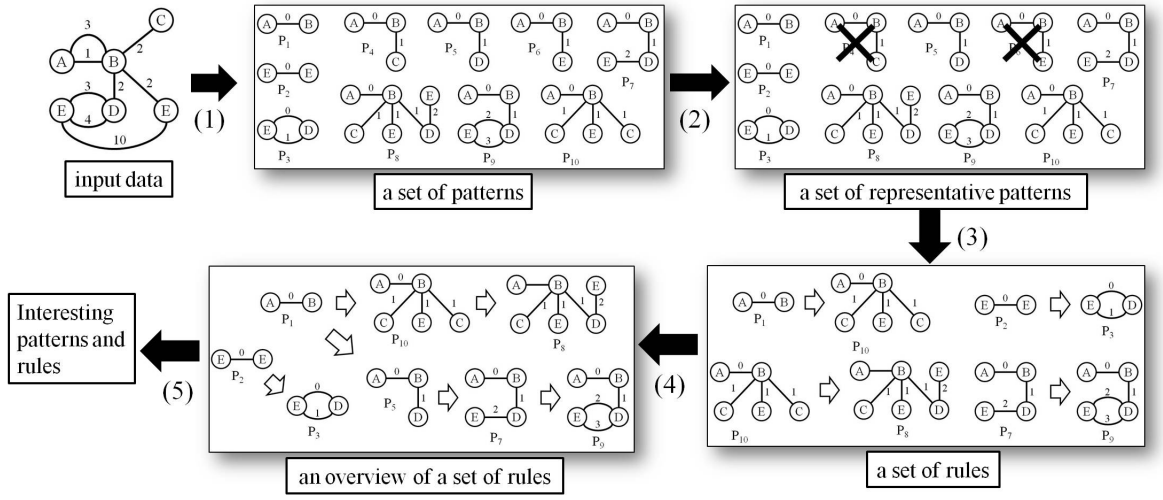

Fig. 2. An overview of the proposed framework

labeled graph having a set of vertices V_G, a set of edges $E_G \subseteq V_G \times V_G$, a labeling function $l_G : V_G \cup E_G \rightarrow \mathcal{L}$ that maps each element of a graph to an alphabet in $\mathcal{L}$, and a time-stamp function $t_G : E_G \rightarrow \mathcal{T}$ that assigns a time-point in $\mathcal{T}$ to each edge. While multi-edges having different time-stamps are allowed in a time-evolving graph, we assume that labels of vertices and edges never change with time for the simplicity. A graph "input data" in Fig. 2 is an example of time-evolving graphs. In the graph, there are two edges between vertices A and B. One has "1" of time-stamp, the other has "3".

As a pattern in a time-evolving graph, we employ a *relative time pattern* which is a connected time-evolving graph $P = (V_P, E_P, l_P, t_P)$ whose lowest time-stamp is zero ($\min_{e \in E_p} t_p(e) = 0$). The concept of relative time pattern is originally introduced in [1]. In this paper, we extend the original definition to allow multi-edges among vertices to represent plural relations and repeated communications in different time-points. Note that, time-points associated to edges in relative time patterns represent the relative relation among the periods of edge formations. For example, a relative time pattern P_4 in Fig. 2 states that an edge between two vertices B and C is generated in the next period when an edge between A and B is established. Hereafter, we call relative time pattern as time pattern for the sake of simplicity.

We introduce a subgraph relation and support value on time-evolving graphs. A time-evolving graph P_1 is said to be a subgraph of another time-evolving graph P_2, denoted as $P_1 \subset P_2$, if there exists a parameter $\Delta \in \mathcal{R}$ and an injective function $f : V_{P_1} \rightarrow V_{P_2}$ which satisfies the conditions: (i) $\forall v \in V_{P_1} [l_{P_1}(v) = l_{P_2}(f(v))]$, (ii) $\forall (u,v) \in E_{P_1} [l_{P_1}(u,v) = l_{P_2}(f(u), f(v))]$, and (iii) $\forall (u,v) \in E_{P_1} [t_{P_2}((f(u), f(v))) = t_{P_1}((u,v)) + \Delta]$. An *occurrence* of P_1 in P_2 with respect to a function f satisfying the conditions on subgraph relation is defined as a list $o = [v_1, v_2, \cdots, v_{|V_{P_1}|}](v_i \in V_{P_2}, v_i = f(u_i), u_i \in V_{P_1})$ of vertices in P_2 mapped from vertices in P_1 by f. The notation $o(v_i)$ is used to represent the i-th element in an occurrence o. All occurrences of P_1 in P_2 is denoted as $\Phi_{P_2}^{P_1}$. Fig. 3 shows an example of a set of occurrences of P in G (Φ_G^P) and their parameter Δ. A list o_1 consists of three vertices

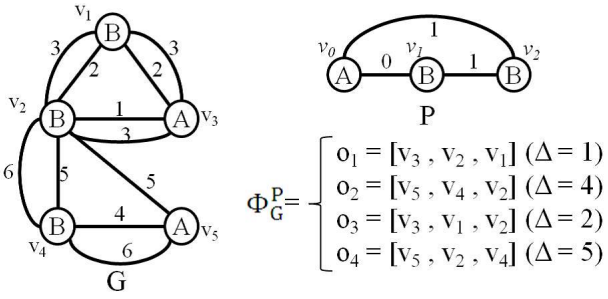

Fig. 3. Occurrences of a time pattern P in a time-evolving graph G

v_3, v_2 and v_1 in G which correspond to v_0, v_1 and v_2 in P, respectively.

The *support* value of a time pattern P in a time-evolving graph G is defined as $sup_G(P) = \min_{v_i \in V_P} |\{o(v_i) | o \in \Phi_G^P\}| / |V_G|$, i.e. the ratio of minimum number of unique vertices $o(v_i)$ in $o \in \Phi_G^P$ to the total number of vertices. This definition is a simple adoption of the support value of a pattern in a single graph setting[3]. In Fig. 3, $|o(v_0)|$ is the minimum number and the support value of P in G is calculated as $sup_G(P) = 0.4$. Given a threshold σ, a time pattern P is said to be *frequent* if $sup_G(P) \geq \sigma$ holds.

Then, we consider rules between two time patterns for representing graph evolutions. We adopt *graph evolution rules*[1] in the form of $P_b \rightarrow P_h$ where P_b and P_h are time patterns and P_b is obtained by deleting all the edges having the last or maximal time-point from P_h. The formal definition is given below. For two time patterns P_h and P_b, a rule $P_b \rightarrow P_h$ is defined as a *graph evolution rule* with respect to P_h if the following conditions hold: (i)$E_{P_b} = \{e \in E_{P_h} | t_{P_h}(e) < \max_{e' \in E_{P_h}}(t_{p_h}(e'))\}$, (ii)$V_{P_b} = \{v \in V_{P_h} | deg(v, E_{P_b}) > 0\}$ where $deg(v, E_{P_b})$ denotes the degree of v in P_b, and (iii)P_b is connected.

While every time pattern P_h can not form a graph evolution rule $P_b \rightarrow P_h$, P_h determines its body P_b uniquely if such P_b exists. A notation $r(P_h)$ is used to represent a unique rule $P_b \rightarrow P_h$ obtained by P_h. A rule of $r(P_7) = P_5 \rightarrow P_7$ in Fig. 2 is an example of graph evolution rule. This rule states

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

that, if a graph pattern having two edges is established by adding edges A–B and B–D in this order, then an additional edge between D and E will be generated in the next period with a certain probability.

The probability or confidence of a graph evolution rules is discussed below. While two definitions on the strength of graph evolution rules are employed in the previous study[1], we propose another definition of confidence of a graph evolution rule $P_b \rightarrow P_h$ based on the occurrences of P_b. The *confidence* value of a graph evolution rule $P_b \rightarrow P_h$ is defined as $conf_G(P_b \rightarrow P_h) = |O_{P_b}(P_h)|/|\Phi_G^{P_b}|$ where $O_{P_b}(P_h)$ denotes a set of occurrence $o_b \in \Phi_G^{P_b}$ which can be extendable to an occurrence $o_h \in \Phi_G^{P_h}$. It is natural to consider that an occurrence of P_b grows into an occurrence of P_h by evolutions. Thus, we believe that this definition, i.e. the ratio of occurrences in P_b grown into an occurrence in P_h, is suitable for capturing the strength of graph evolution rules. Given a threshold τ, a graph evolution rule $P_b \rightarrow P_h$ is said to be *valid* if $conf_G(P_b \rightarrow P_h) \geq \tau$.

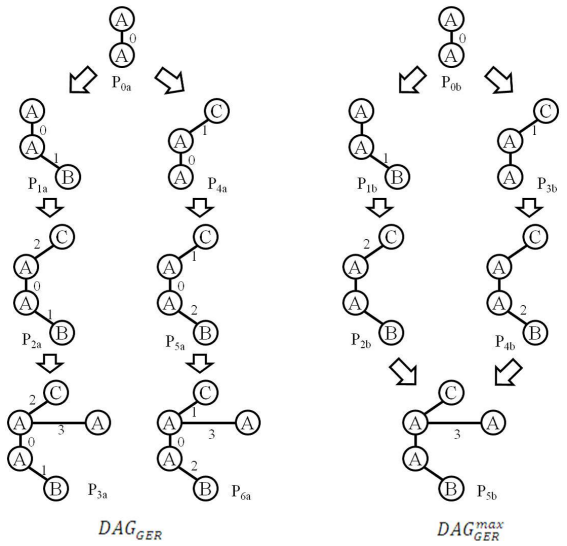

Fig. 4. An example of DAG_{GER} and DAG_{GER}^{max}

B. Representative Time Patterns

Frequent pattern miners often discover a huge amount of patterns[8]. In order to compress a set of frequent time patterns into a representative set of manageable size, we apply a structural representative approach proposed in [4] which considers two aspects, (i)structural representability and (ii)support preservation. This approach is originally proposed for patterns in a graph database (a set of graphs) without time-stamps, we extend it to handle relative time patterns in a time-evolving graph.

The structural representability is discussed first. Intuitively speaking, the structural difference between two time patterns having the same vertex sets is measured as a degree of difference between edge sets. Given an error tolerance parameter δ, the *structural difference* between two time patterns P' and P is defined as follows:

$$diff(P', P) = \begin{cases} \min_{f \in \mathcal{F}(P', P)} d(P', P, f) & (\mathcal{F}(P', P) \neq \emptyset) \\ \infty & (otherwise) \end{cases}$$

where $\mathcal{F}(P', P)$ denotes a set of bijective mappings satisfying (i)$\forall v' \in V_{P'}\,[l_{P'}(v') = l_P(f(v'))]$ and (ii) $d(P', P, f) = \sum_{v'_1, v'_2 \in V_{P'}} |I^{P'}(v'_1, v'_2) - I^P(f(v'_1), f(v'_2))| \leq \delta$ in which $I^P(u, v)$ is an indicator function such that $I^P(u, v) = 1$ if $(u, v) \in E_p$ and $I^P(u, v) = 0$ otherwise. We define the structure representable by using the structural difference. A time pattern P' is said to *be structure representable by* another time pattern P if an inequality

$$Rs(P', P) = \min_{|V_{P'}| = |V_{P''}|, P'' \subseteq_0^{induce} P} diff(P', P'') \leq \delta$$

holds where $\subseteq_0^{induce}$ denotes an induced subgraph relationship[4]. In simple words, P' is structure representable by P if P has a subgraph P'' which shares the same vertex set and similar edge set with P'.

Then, on specifying the degree of support preservation, *smoothed support* for time pattern is proposed by slightly modifying the original definition in [4]. Given a set of frequent time patterns $TP = \{P_1, \cdots, P_n\}$ in a time-evolving graph G, the smoothed supporting occurrence set of P_i is defined as $S_{P_i} = \bigcup_{P_j \in TP, diff(P_i, P_j) \leq \delta} \Phi_G^{P_J}$. In other words, S_{P_i} contains all the occurrences in structural similar patterns P_j with P_i. Then, the *smoothing support* value of $P_i \in TP$ is defined as $ssup_G(P_i) = \min_{1 \leq i \leq |V_{P_i}|} |\{o(i) \,|\, o \in S_{P_i}\}|$. By using smoothing support, we define the preservation of support value. Given a tolerance parameter ϵ, we say that the support of a pattern P' *is preserved by* another pattern P, if an inequality

$$Ps(P', P) = \frac{|ssup_G(P) - ssup_G(P')|}{\max\{ssup_G(P), ssup_G(P')\}} \leq \epsilon$$

holds.

Given two parameters δ and ϵ, a time pattern P_i will be judge as representative in a set of frequent time patterns $TP = \{P_1, \cdots, P_n\}$ if a set

$$C_{TP}^{\delta,\epsilon}(P_i) = \{P_j \in TP \mid Rs(P_j, P_i) \leq \delta, Ps(P_j, P_i) \leq \epsilon\}$$

is large enough. A concrete algorithm for determining representative time patterns will be introduced in section III.

C. Graph Evolution DAGs

Compared with the examination of individual graph evolution rule one by one, consideration of a set of graph evolution rules *all at once* must be effective for finding interesting and important patterns related to graph evolution. For this purpose, we consider a graph-based summarization of a set of graph evolution rules.

Given a set of graph evolution rules GER, a *graph evolution DAG* on GER, denoted as DAG_{GER}, is defined as a directed acyclic graph $G = (V, E)$ where $E = \{(p_b, p_h) \,|\, p_b \rightarrow p_h \in GER\}$ is a set of edges representing GER, and $V = \{p \,|\, p \rightarrow p_h \in GER \vee p_b \rightarrow p \in GER\}$ is a set of time patterns in GER. In other words, DAG_{GER} is obtained by merging the same heads and bodies among graph evolution rules. DAG_{GER} consists of a set of trees by definition. An example of DAG_{GER} is shown in Fig. 2. We can expect to obtain new findings and precise insights on graph evolutions by observing the relationships among time patterns and graph evolution rules summarized in a graph evolution DAG from a global point of view precisely.

In order to capture the process of graph evolution in

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

an abstract point of view with respect to time-stamps, we consider a generalized time pattern obtained by removing all time points except the last or maximal ones from a time pattern. In other words, edges in a time pattern are categorized into two groups, (i)edges created in the last period and (ii)edges before the last period. For example, a time pattern P_{3a} will be abstracted into P_{5b} in Fig. 4 by removing the time points '0', '1' and '2' associated to three edges. By this abstraction, since the detailed order of edge formation is ignored, plural time patterns can be recognized as an identical one in an abstract sense. Thus, we can expect to provide an abstract overview of graph evolutions in a time-evolving graph by using the abstract time patterns.

Motivated by the above discussion, we construct another directed acyclic graph from a graph evolution DAG DAG_{GER} by replacing every time pattern in DAG_{GER} with their abstractions and merging vertices having identical abstract time pattern into one. We call such DAG as an abstract graph evolution DAG and denote it as DAG^{max}_{GER}. An example of abstract graph evolution DAG is shown in Fig. 4 (DAG^{max}_{GER}).

Algorithm TP-Miner(G, σ, $\mathcal{L}$)
1: $TP := \{\}$
2: **for each** $P \in \mathcal{L}$
3: TP-Enum(P, G, σ, $\mathcal{L}$, TP)
4: **return** TP

Subroutine TP-Enum(P, G, σ, $\mathcal{L}$, TP)
1: **if** $\neg$isCan(P) $\vee$ $sup_G(P) < \sigma$ **then return**
2: $TP := TP \cup \{P\}$
3: **for each** $e \in$ RMB(P)
4: **for each** $t \in$ TL(P, e)
5: $P' := P \cdot e$; setTime(e, t) ;
6: call TP-Enum(P', G, σ, $\mathcal{L}$, TP)

Fig. 5. Pseudo code of frequent time pattern miner

Algorithm RTP-Selector(TP, τ, δ, ϵ, RTP)
1: $RP := \{P \in TP \mid \exists r(P)\ s.t.\ conf(r(P)) \geq \tau\}$
2: $RTP := \{\}$
3: **while** ($RP \neq \emptyset$)
4: select a pattern $P \in RP$
5: s.t. $|\, C^{\delta,\epsilon}_{TP}(P)\,| = \max_{P_j \in RP} |\, C^{\delta,\epsilon}_{TP}(P_j)\,|$
6: $RTP := RTP \cup \{P\}$
7: $RP := RP \setminus C^{\delta,\epsilon}_{TP}(P)$; $TP := TP \setminus C^{\delta,\epsilon}_{TP}(P)$
8: **return** RTP

Fig. 6. Pseudo code for selecting representative time patterns

III. Discovery of Interesting Patterns and Rules

A. An Overview

Our objective in this paper is to find interesting patterns and rules related to evolutions in time-evolving graphs. To achieve this objective, we propose the following procedures:

1) Discover a set TP of all frequent time patterns from a time-evolving graph G.
2) Select a set RTP of representative time patterns from TP by considering structural representability and support preservation.
3) Build a set GER of valid graph evolution rules from RTP.
4) Construct a graph evolution DAG DAG_{GER} from GER as well as an abstract graph evolution DAG DAG^{max}_{GER}.
5) Extract interesting patterns and rules related with graph evolution by examining DAG_{GER} and DAG^{max}_{GER}.

Several criteria on importance and significance of patterns and rules can be considered in the last step of the above procedures. This issue will be discussed in section IV.

Given a set RTP of representative time patterns obtained in step 2, we can construct a set of graph evolution rules and graph evolution DAGs in a straightforward way. Thus, in the following, only a method for obtaining representative time patterns will be explained in detail.

B. Mining Representative Relative Time Patterns

We take advantages of a frequent subgraph discovery algorithm named gSpan[7], which uses the rightmost extension and the canonical representation[2] of graph patterns. As pointed out in [1], while gSpan is originally designed for pattern mining from a set of graphs, it can be easily applicable for mining frequent time patterns (i)by a slight modification on the support computation and (ii)by an extension of canonical representation for graph pattern having time-points.

An algorithm for mining frequent relative time patterns is shown in Fig. 5. While we extend the algorithm to handle multi-edge patterns, this algorithm is essentially the same as an algorithm GERM proposed in [1]. In the algorithm, G, σ and $\mathcal{L}$ denote a time-evolving graph, a minimum support threshold, and a set of labels, respectively. A set TP is used for storing frequent time patterns obtained during the execution. For each graph pattern P having one vertex, new time patterns will be generated by repeatedly applying a procedure TP-Enum (line 2,3 of TP-Miner). In TP-Enum, if a time pattern P is not canonical ($\neg isCan(P)$ in line 1), then P will be pruned to avoid the duplicated enumerations of the same patterns. As similar, infrequent pattern P will be also pruned ($sup_G(P) < \sigma$ in line 1) since no frequent time patterns can be obtained by the specialization of P. After storing frequent relative time patterns (in line 2), the rightmost extension[7] will be applied for generating new candidates (in line 3–6). In this extension, a new candidate of frequent time pattern $P' = P \cdot e$ will be generated (i)by adding an edge e in a set RMB(P) of the rightmost branches[7] to P and (ii)by assigning a time-point t in a set TL(P, e) of relative time-stamps for e with respect to the occurrence of $P \cdot e$ (setTime(e, t) in line 5).

After obtaining a set TP of all frequent time patterns, a set RTP of representative patterns is extracted from TP by using a greedy covering algorithm shown in Fig. 6. To build a useful graph evolution DAGs, we avoid selecting a time pattern $P \in TP$ as a representative, if P has no contribution to build a graph evolution DAGs. For such purpose, we prepare a set RP of time patterns from which valid graph evolution rules can be obtained (line 1 in Fig. 6).

While representative patterns will be selected from RP, the size of $C^{\delta,\epsilon}_{TP}$ obtained by TP will be employed as an evaluation measure. The algorithm selects a time pattern $P \in RP$ having the maximal significance as a representative pattern and stores it into RTP (line 4–6). Then, all patterns covered by P are removed from RP and TP, respectively (line 7). These processes are iterated until RP becomes empty (line 3).

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

IV. EXPERIMENTS

In order to assess the effectiveness of the proposed framework, we implement a series of algorithms in Java language and conduct preliminary experiments by using the Enron Email Dataset[5] on a PC (CPU:Intel(R) Xeon(R), 3.3GHz) with 32GByte of main memory running Windows XP.

By extracting email communications within a particular year from the Enron data, a time-evolving graph G_1 with daily granularity is prepared in which each vertex corresponds to a person and each edge represents a certain communication between persons. While the position in the occupation is used as a vertex label, the sort of email communications (To, Cc and Bcc) is employed as edge labels. The resulting time-evolving graph G_1 consists of 155 vertices and 5,606 edges. In addition, we prepare another time-evolving graph G_2 with monthly granularity in the same manner. G_2 contains 155 of vertices and 2,208 edges, respectively.

A. Effects of thresholds on extracting time patterns and graph evolution rules

A time-evolving graph G_1 is used as a target data in the first experiments. Given several combinations of three thresholds, (i)σ for support value of time patterns, (ii)τ for confidence value of graph evolution rules, and (iii)ϵ for representative time patterns, we measure the number of frequent time patterns and the number of graph evolution rules obtained from representative patterns. In addition, the number of vertices and edges in graph evolution DAGs (DAG_{GER}) as well as abstract graph evolution DAGs (DAG^{max}_{GER}) are examined. Note that, the number of edges in a graph evolution DAG is identical with the number of graph evolution rules. In all experiments, fourth threshold δ for representative time patterns was set to 1.

The experimental results are shown in Table II. All results are obtained within a reasonable computation time. Number of discovered frequent time patterns are greatly different between two support threshold $\sigma = 0.07$ and $\sigma = 0.05$. The same tendency can be observed on the number of graph evolution rules. Compared with σ, thresholds τ and ϵ seem to give small impact on the results. While it does not necessarily to hold because of the greedy algorithm, the number of representative time patterns increases as a threshold ϵ becomes smaller.

We succeeded in compressing a set of frequent time patterns into a small representative set. The numbers of representative patterns are reduced to 26.4% and 59.8% in $\sigma = 0.07$ and $\sigma = 0.05$, respectively. In case of $\sigma = 0.05$, the number of vertices in the DAGs is reduced to 84.9% in average and to 82.3% in the maximal by the abstraction. Compared with the reduction of the vertices, the reduction rate of edges is very small.

B. Interesting time patterns and graph evolution rules

In Table I (left), we show a ranking of top-10 time patterns having high value of centralities in the graph evolution DAG having 419 vertices and 421 edges obtained from G_2 under the condition of $\sigma = 0.07$, $\tau = 0.1$ and $\epsilon = 0.1$.

Since most patterns having high degree centrality consist of two vertices, they correspond to the beginning of graph evolutions. On the other hand, all patterns with high closeness centrality do not have high support value. And, most patterns ranked in high place consist of more than two vertices. Thus, these patterns correspond to certain intermediate steps in graph evolutions. While the concrete time patterns are not shown, a similar tendency can be observed in an abstract graph evolution DAG.

These results show the ability of our proposal to extract time patterns that are difficult to obtain in the traditional support-based framework.

TABLE I
RANKING OF TIME PATTERNS W.R.T NETWORK CENTRALITIES

ranking in DAG_{GER}						ranking in DAG^{max}_{GER}			
C^d	sup	pat.	C^c	sup	pat.	C^d	sup	C^c	sup
1	1	P_1	1	13	P_5	1	1	1	5
2	5	P_2	2	5	P_2	2	13	2	63
3	88	P_3	3	250	P_{11}	3	63	3	88
4	9	P_4	4	1	P_4	4	63	4	245
5	13	P_5	5	88	P_{12}	5	5	5	88
6	23	P_6	6	250	P_{13}	6	88	6	88
7	64	P_7	7	64	P_{14}	7	9	7	63
8	9	P_8	8	88	P_{15}	8	16	8	88
9	64	P_9	9	88	P_{16}	9	23	9	1
10	16	P_{10}	10	64	P_9	10	88	10	63

C^d: rank on degree centrality. C^c: rank on closeness centrality.
sup: rank on support value. pat.: pattern number in Fig. 7

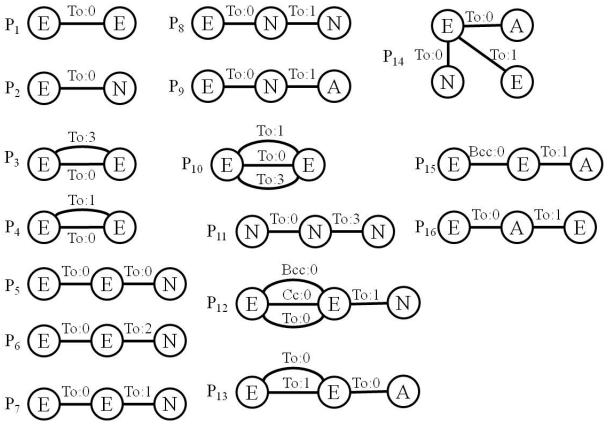

Fig. 7. Time patterns having high centrality

C. Examples of graph evolutions

In Fig. 8, we show examples of the process of graph evolutions found in a time-evolving graph G_2 under the same conditions in the second experiments ($\sigma = 0.07$, $\tau = 0.1$ and $\epsilon = 0.1$).

In this figure, while the left process (a) is obtained from a graph evolution DAG (DAG_{GER}), the right one (b) represents a process in an abstract graph evolution DAG (DAG^{max}_{GER}). An abstract time pattern P_{4b} in (b) was built by merging two time patterns P_{3a} and P_{5a} in (a) into one. Two rules $P_{2b} \rightarrow P_{4b}$ and $P_{3b} \rightarrow P_{4b}$ in (b) say that if the four communications within three time periods are conducted between two individuals labeled E, one additional communication will be performed between one individual E and new individual A in fourth time period. As this simple example shows, on one hand, DAG_{GER} shows the detailed graph evolutions with concrete time points. Precise insights on discovered graph evolutions can be expected to obtain by

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE II
EXPERIMENTAL RESULTS

τ	ϵ	σ	pat.	rule	$\lvert V_D\rvert$	$\lvert V_D^m\rvert$	$\lvert E_D^m\rvert$	time	σ	pat.	rule	$\lvert V_D\rvert$	$\lvert V_D^m\rvert$	$\lvert E_D^m\rvert$	time
	0.9			4	7	6	4	4.2			2,738	3,029	2,615	2,635	257.0
0.3	0.5			4	7	6	4	4.2			2,735	3,045	2,643	2,650	258.6
	0.1			4	7	6	4	4.0			2,795	3,090	2,656	2,691	261.5
	0			4	7	6	4	4.0			2,828	3,103	2,657	2,724	261.1
	0.9			10	17	15	10	4.1			3,645	3,887	3,318	3,542	279.7
0.1	0.5	0.07	79	10	17	15	10	4.0	0.05	6,041	3,776	3,991	3,351	3,673	284.1
	0.1			34	37	35	34	4.0			3,983	4,159	3,450	3,879	289.5
	0			34	37	35	34	4.2			4,164	4,248	3,534	4,060	311.6
	0.9			20	28	28	20	4.2			3,870	4,115	3,540	3,767	285.9
0.05	0.5			20	28	28	20	4.2			3,983	4,206	3,585	3,880	288.6
	0.1			53	56	54	53	4.5			4,296	4,469	3,739	4,192	304.7
	0			53	56	54	53	4.3			4,560	4,588	3,847	4,456	415.9

pat.: number of discovered time patterns. rule: number of graph evolution rules. $|V_D|$: number of vertices in DAG_{GER}
$|V_D^m|$: number of vertices in DAG_{GER}^{max}. $|E_D^m|$: number of edges in DAG_{GER}^{max}. time: execution time in second.

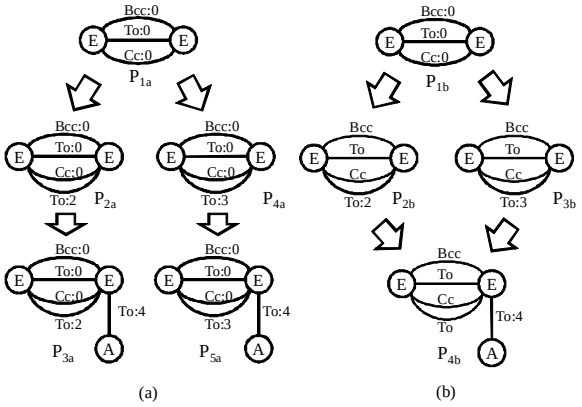

Fig. 8. Examples of graph evolutions represented in two DAGs

considering the whole structure of a graph pattern as well as the order of edge formation. On the other hand, DAG_{GER}^{max} advises us the processes of graph evolutions in an abstract level.

V. RELATED WORK

Two graph miners GERM[1] and LFR-Miner[6] are the most related work with our proposal. GERM[1] extracts relative time patterns and graph evolution rules in a time-evolving graph. As mentioned previously, our proposal in this paper can be regarded as a modification of [1] by allowing multi-edges in a pattern as well as by changing the definition of confidence. LFR-Miner[6] finds Link formation rules which capture the formation of a new link between specified two vertices as a postcondition of existing connections between the two vertices. Link formation rules are closely related to the abstract time patterns. But they are different from the abstract time patterns in the following two points: (i) all edges in a rule must be connected to pre-specified two vertices and (ii) only one edge in a rule is allowed to have the last time period. An abstract time pattern does not have the above restrictions.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a framework for discovering interesting patterns and rules in time-evolving graphs. The usefulness of proposed framework was evaluated by preliminary experiments with real world datasets. In the framework, graph evolution DAGs built from representative and abstract graph evolution rules will help users' understand by giving an brief overview of discovered patterns and rules. In addition, those DAGs enable us to evaluate the significance of patterns from the aspect of relationships among patterns. In other words, graph evolution DAGs provide additional criteria for pattern discovery other than traditional interestingness measures such as support and confidence.

For future work, further experiments with large-scale datasets and detailed assessment of the quality of obtained graph evolution DAGs are necessary. Furthermore, we investigate to develop a frequent subgraph miner specialized for discovering abstract time patterns having last time-stamp only, i.e. patterns in DAG_{GER}^{max}, directly from time-evolving graphs. By combining the rules with detailed time information and the abstract ones, we can expect to discover useful patterns and rules having appropriate time granularity for capturing important and critical processes in time-evolving graphs.

REFERENCES

[1] M. Berlingerio, F. Bonchi, B. Bringmann and A. Gionis, "Mining Graph Evolution Rules," in *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases, Part I*, 2009, pp.115–130.

[2] C. Borgelt, "On canonical forms for frequent graph mining", *Working Notes of the Third International ECML/PKDD-Workshop on Mining Graphs, Trees and Sequences*, 2005, pp.1–12.

[3] B. Bringmann and S. Nijssen, "What is frequent in a single graph?", *Proc. of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2008, pp.858–863.

[4] C. Chen, C. X. Lin and X. Yan and J. Han, "On effective presentation of graph patterns: a structural representative approach", *Proc. of the 17th ACM conference on Information and knowledge management*, 2008, pp.299–308.

[5] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Proc. of the 15th European Conference on Machine Learning*, 2004, pp. 217–226.

[6] C. W.-k. Leung, E.-P. Lim, D. Lo and J. Weng, "Mining Interesting Link Formation Rules in Social Networks", in *Proc. of the 19th ACM Conference on Information and Knowledge Management*, 2010, pp.209-218.

[7] X. Yan and J. Han, "gSpan : Graph-based substructure pattern mining," in *Proc. of the 2nd IEEE International Conference on Data Mining* , 2002, pp. 721–724.

[8] X. Yan and J. Han, "CloseGraph: Mining closed frequent graph patterns," in *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 286–295.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)