PAPER

FRISSMiner: Mining Frequent Graph Sequence Patterns Induced by Vertices*

Akihiro INOKUCHI^{†a)} and Takashi WASHIO[†], Members

SUMMARY The mining of a complete set of frequent subgraphs from labeled graph data has been studied extensively. Furthermore, much attention has recently been paid to frequent pattern mining from graph sequences (dynamic graphs or evolving graphs). In this paper, we define a novel subgraph subsequence class called an “induced subgraph subsequence” to enable the efficient mining of a complete set of frequent patterns from graph sequences containing large graphs and long sequences. We also propose an efficient method for mining frequent patterns, called “FRISSs (Frequent Relevant, and Induced Subgraph Subsequences)”, from graph sequences. The fundamental performance of the method is evaluated using artificial datasets, and its practicality is confirmed through experiments using a real-world dataset.

key words: frequent pattern mining, graph sequence, induced subgraph subsequence

1. Introduction

Studies on data mining have established many approaches for finding characteristic patterns from various structured data. Graph Mining, which efficiently mines all subgraphs appearing more frequently than a given threshold from a set of graphs, focuses on the topological relations between vertices in the graphs [5], [19]. AGM [10], gSpan [20], and Gaston [15] mine frequent subgraphs starting with those of size 1 by using the anti-monotonic property of the support values. Although the major algorithms for Graph Mining are quite efficient in practice, they require much computation time to mine complex frequent subgraphs due to the NP-completeness of subgraph isomorphism matching [8]. Accordingly, these conventional methods are not suitable for very complex graphs such as graph sequences.

However, graph sequences can be used to model objects for many real world applications. For example, a human network can be represented as a graph whose vertexes and edges correspond to humans and relationships among humans, respectively. If a human joins (or leaves) the community in the human network, the numbers of vertexes and edges in the graph increase (or decrease). Similarly, a gene network consisting of genes and their interactions produces a graph sequence in the course of their evolutionary history by acquiring new genes, deleting genes, and mutating genes.

Recently, much attention has been paid to frequent pattern mining from graph sequences [12] (dynamic graphs [3] or evolving graphs [2]). Figure 1 (a) shows an example of a graph sequence containing 4 steps (elements) and 5 vertex IDs denoted by the numbers attached to vertices. In [12], we proposed a novel method, called GTRACE (Graph Transformation sequence mining), to mine frequent patterns as shown in Fig. 1 (b) from graph sequences under the assumption that the change in each graph is gradual and applied it to graph sequences generated from the Enron dataset. Although GTRACE is tractable for the Enron graph sequences containing about 7 steps and 100 vertex IDs, it is intractable for graph sequences containing more steps and vertex IDs than those in the Enron graph sequences.

In this paper, we define a novel class of subgraph subsequence called an “induced subgraph subsequence” of a graph sequence to mine frequent patterns efficiently from graph sequences containing long sequences and large graphs. In addition, we propose an efficient method for mining frequent patterns called FRISSs (Frequent, Relevant, and Induced Subgraph Subsequences) from graph sequences. The fundamental performance of the proposed method is evaluated using artificial datasets, and its practicality is confirmed by experiments using a real world dataset.

The rest of this paper is organized as follows. The remainder of this section reviews conventional frequent graph mining and frequent graph sequence mining which has recently attracted much attention. In Sect. 2, we introduce the induced subgraph subsequence of a graph sequence by extending the definition of an induced subgraph of a graph and we define the problem addressed in this paper. Section 3 proposes an efficient method for mining all frequent patterns from a set of graph sequences. In Sect. 4, the fundamental performance and practicality of the proposed method are demonstrated through experiments. Finally, Sect. 5 presents a discussion and concludes this paper.

Manuscript received October 13, 2011.

Manuscript revised February 7, 2012.

[†]The authors are with the Institute of Scientific and Industrial Research, Osaka University, Ibaraki-shi, 567–0047 Japan.

*The main idea of this paper was presented at the SIAM International Conference on Data Mining in 2010.

a) E-mail: inokuchi@ar.sanken.osaka-u.ac.jp

DOI: 10.1587/transinf.E95.D.1590

Fig. 1 Examples of the graph sequence and subgraph subsequence for mining.

1.1 Frequent Graph Mining

Graph Mining is the task of finding novel, useful, and “understandable” graph-theoretic patterns in a graph representation of data [4]. Frequent Graph Mining is a representative task in Graph Mining that efficiently mines all subgraphs that appear more frequently than a given threshold from a set of labeled graphs.

A labeled graph g is represented as $g = (V, E, L, l)$, where $V = \{v_1, \dots, v_n\}$ is a set of vertices, $E \subseteq \{(v, v') \mid (v, v') \in V \times V\}$ is a set of edges[†], and L is a set of labels such that $l : V \cup E \rightarrow L$. $V(g)$, $E(g)$, and $L(g)$ are sets of vertices, edges, and labels of g , respectively. If an edge exists between two vertices, the vertices are said to be adjacent. The set of edges from vertex v to v' is called a path. A graph in which there is a path between any two vertices is said to be connected, and an unconnected graph consists of some connected components.

Given two graphs $g=(V, E, L, l)$ and $g'=(V', E', L', l')$, g' is called a subgraph of g , denoted as $g' \sqsubseteq g$, if there exists an injective function $\phi : V' \rightarrow V$ that satisfies the following three conditions for $\forall v, v_1, v_2 \in V'$.

1. $(\phi(v_1), \phi(v_2)) \in E$, if $(v_1, v_2) \in E'$,
2. $l'(v) = l(\phi(v))$,
3. $l'((v_1, v_2)) = l((\phi(v_1), \phi(v_2)))$.

A graph database DB is a set of tuples $\langle tid, g \rangle$, where tid is a graph ID and g is a labeled graph. A tuple $\langle tid, d \rangle$ is said to contain a graph p , if p is a subgraph of g , i.e., $p \sqsubseteq g$. The support of a graph p in the database DB is the number of tuples in the database containing p , i.e.,

$$\sigma(p) = |\{tid \mid (\langle tid, g \rangle \in DB) \wedge (p \sqsubseteq g)\}|. \quad (1)$$

Given a positive integer σ' as the support threshold, a graph p is called a “frequent subgraph” pattern in the graph database DB , if at least σ' tuples in the database contain p , i.e., $\sigma(p) \geq \sigma'$. Representative methods for frequent graph mining, such as AGM [10], gSpan [20], and Gaston [15], mine frequent subgraphs starting with those of size 1 by using the anti-monotonic property of the support values.

AGM enumerates a complete set of frequent subgraph patterns. However, for practical reasons, some of the methods often focus on frequent and “connected” subgraph patterns which are contained as “induced subgraphs” in labeled graphs in the graph database DB . Here, a subgraph g' of g is an “induced subgraph”, denoted as $g' \sqsubseteq_i g$, if and only if two vertices in $V(g')$ are adjacent in g' and they are also adjacent in g . If they mine frequent subgraph patterns which are contained as induced subgraphs in labeled graphs in DB , the definition of support is altered to

$$\sigma_i(p) = |\{tid \mid (\langle tid, g \rangle \in DB) \wedge (p \sqsubseteq_i g)\}|. \quad (2)$$

Frequent subgraph patterns mined under this definition are called “frequent and induced subgraph” patterns.

If an edge does not exist between two vertices in a

Fig. 2 A graph database and subgraph patterns.

frequent subgraph pattern mined under the support definition (1), the pattern suggests that an edge may or may not exist between two corresponding vertices in many of the graphs in DB containing the pattern. On the other hand, if an edge does not exist between two vertices in a frequent and induced subgraph pattern mined under the support definition (2), the pattern suggests that an edge does not exist between two corresponding vertices in many of the graphs in DB containing the pattern. For example, Fig. 2 (a) shows three labeled graphs in the graph database. Although all three graphs contain loops (shown as triangles), both the subgraph pattern shown in Fig. 2 (c) and that shown in Fig. 2 (b) which does not contain any loops are mined under the support definition (1). We, therefore, can not know all connectivities between vertices in the pattern without accessing graph in DB containing this pattern. On the other hand, the subgraph pattern shown in Fig. 2 (b) is not mined under the support definition (2), because the subgraph pattern shown in Fig. 2 (b) is not an induced subgraph of the graphs shown in Fig. 2 (a). We, therefore, do not need to access the graph database to understand the frequent and induced subgraph pattern.

Next, we consider a case that both connected and unconnected subgraph patterns are mined from DB consisting of connected graphs under the definition (2). In this case, we can not know the relevancy between two components in an unconnected subgraph pattern without accessing the original graphs in DB containing this pattern. Therefore, practically, we focus on frequent connected subgraph patterns which are contained as induced subgraphs in the graphs in DB . Since we do not need to access the graphs in DB to understand frequent, connected, and induced subgraph patterns, we consider the patterns to be easily understandable.

1.2 Frequent Graph Sequence Mining

Figure 1 (a) shows an example of an observed graph sequence. The graph $g^{(j)}$ is the j -th labeled graph in the sequence. The problem we address in this paper is how to mine patterns that appear more frequently than a given threshold from a set of graph sequences. In [12], we proposed transformation rules to represent graph sequences compactly under the assumption that “the change is gradual”. In other word, only a small part of the structure changes, while the other part remains unchanged between

[†]Although this paper focuses on undirected graphs only, the proposed method is also applicable to directed graphs without loss of generality.

Fig. 3 Change between two successive graphs.

two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in a graph sequence. For example, the change between two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in the graph sequence shown in Fig. 3 is represented as an ordered list of two transformation rules $\langle vi_{[1,A]}, ed_{[(2,3),\bullet]} \rangle$. This list implies that a vertex with vertex ID 1 and label A is inserted (*vi*), and then an edge between vertices with vertex IDs 2 and 3 is deleted (*ed*). By assuming the change in each graph to be gradual, we can represent a graph sequence compactly even if the graph in the graph sequence has many vertices and edges. Moreover, the method called GTRACE efficiently mines all frequent patterns from ordered lists of transformation rules which are compactly represented.

However, when the temporal resolution is low for observing a graph sequence, the length of ordered list of transformation rules representing the graph sequence increases. This is because large portions of two successive graphs in the graph sequence change. Furthermore, as the length of the ordered list of transformation rules to represent the graph sequence increases, GTRACE becomes intractable for mining frequent patterns from the long ordered lists under the low support threshold, because according to the anti-monotonic property of the support, sub-patterns of the frequent patterns are also frequent; hence, a huge set of frequent patterns has to be mined.

In this paper, we propose an efficient method for mining understandable frequent patterns from a set of graph sequences containing many vertex IDs and steps (elements) under the assumption that the change in each graph is NOT gradual.

2. Problem Definition

2.1 Graph Sequence

A graph sequence is an ordered list of labeled graphs and is represented as $d = \langle g^{(1)} g^{(2)} \dots g^{(l)} \rangle$, where the superscript integer j ($1 \leq j \leq l$) for each labeled graph $g^{(j)}$ represents an ordered step in the graph sequence. $g^{(j)}$ is called an “element” of the sequence. The number of elements in the graph sequence, l , is called the “length” of the graph sequence, which is denoted as $len(d)$. Furthermore, the number of vertices in the graph sequence, $\sum_{j=1}^l |V(g^{(j)})|$, is called the “size” of the graph sequence. We assume that each vertex v is mutually distinct from the others in any $g^{(j)}$ and has a “vertex ID” $id(v)$ in d . We define the set of vertex IDs $ID(d)$ as

$$ID(d) = \{id(v) \mid v \in V(g^{(j)}), g^{(j)} \in d\}.$$

Example 1: In the human network mentioned in Sect. 1,

(a) A graph sequence d_1 .

(b) A subgraph subsequence d_2 of d_1 .

Fig. 4 Inclusion relation between two graph sequences.

each person has a vertex ID, and his/her gender is an example of a vertex label.

Example 2: Figure 4(a) shows a graph sequence $d_1 = \langle g_1^{(1)} g_1^{(2)} g_1^{(3)} g_1^{(4)} \rangle$ with four elements. In this graph sequence, each vertex has one of the vertex labels $\{A, B, C\}$, and all edges have the same edge label. In addition, the number attached to each vertex depicts the vertex ID of the vertex, that is, $ID(d_1) = \{1, 2, 3, 4\}$. In a graph sequence, the numbers of vertices and edges increase and decrease, and their labels can also change.

We now define the inclusion relation between two graph sequences α and β .

Definition 1: A graph sequence $\alpha = \langle a^{(1)} a^{(2)} \dots a^{(n)} \rangle$ is called a “subgraph subsequence” of another graph sequence $\beta = \langle b^{(1)} b^{(2)} \dots b^{(m)} \rangle$, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ and an injective function $\phi : ID(\alpha) \rightarrow ID(\beta)$ such that

- $a^{(1)} \subseteq b^{(j_1)}, a^{(2)} \subseteq b^{(j_2)}, \dots, a^{(n)} \subseteq b^{(j_n)}$, and
- for $v \in V(a^{(i)})$ and $v' \in V(a^{(i')})$, if $id(v) = id(v')$, then $\exists(u \in V(b^{(j_i)}))$ and $u' \in V(b^{(j_{i'})})$ s.t. $id(u) = \phi(id(v)) \wedge id(u') = \phi(id(v'))$. ■

The first condition is similar to the definition of a subsequence used in conventional sequential pattern mining [16]. The second condition implies that if the vertex IDs of two vertices v and v' in two different elements $a^{(i)}$ and $a^{(i')}$ in the graph sequence α are the same, the vertex IDs of vertices u and u' in the graph sequence β to which the vertices v and v' are mapped by ϕ are also the same.

Example 3: Figure 4 shows an example that demonstrates the inclusion relation between two graph sequences $d_1 = \langle g_1^{(1)} g_1^{(2)} g_1^{(3)} g_1^{(4)} \rangle$ and $d_2 = \langle g_2^{(1)} g_2^{(2)} g_2^{(3)} \rangle$, i.e., $d_2 \subseteq d_1$. Vertices with vertex IDs 1, 2, and 3 in the graph sequence d_2 are mapped onto hatched vertices with vertex IDs 2, 3, and 4 in the graph sequence d_1 , respectively, by an injective function ϕ . That is, $\phi(1) = 2, \phi(2) = 3$, and $\phi(3) = 4$. In other words, $g_1^{(1)} \subseteq g_2^{(1)}, g_1^{(2)} \subseteq g_2^{(2)}$, and $g_1^{(3)} \subseteq g_2^{(3)}$ hold.

A graph sequence database DB is a set of tuples $\langle sid, d \rangle$, where sid is a graph sequence ID and d is a graph

Fig. 5 Graph sequence d and its union graph $g_u(d)$.

sequence. A tuple $\langle sid, d \rangle$ is said to contain a graph sequence α , if α is a subgraph subsequence of d , i.e., $\alpha \sqsubseteq d$. The “support” of a graph sequence α in a graph sequence database DB is the number of tuples in the database containing α , i.e.,

$$\sigma(\alpha) = |\{sid \mid (\langle sid, d \rangle \in DB) \wedge (\alpha \sqsubseteq d)\}|.$$

According to the definitions of a subgraph subsequence and the support, the anti-monotonicity of this support value holds. That is, if $\alpha \sqsubset \beta$, then $\sigma(\alpha) \geq \sigma(\beta)$. Given a positive integer σ' as the support threshold, a graph sequence α is called a Frequent Subgraph Subsequence (FSS) pattern in the graph sequence database DB , if at least σ' tuples in the database contain α , i.e., $\sigma(\alpha) \geq \sigma'$. In addition, we assume that an FSS $\alpha = \langle a^{(1)} a^{(2)} \dots a^{(n)} \rangle$ does not contain an element $a^{(i)}$ such that $V(a^{(i)}) = \emptyset$.

2.2 Union Graph and Relevancy

As mentioned in Sect. 1.1, many algorithms of conventional frequent graph mining mines frequent patterns that are understandable by limiting the frequent subgraph patterns to connected and induced subgraph patterns. In this and the following subsections, we extend the definitions of a connected subgraph and an induced subgraph, respectively, to a graph sequence.

First, we define the relevancy between vertex IDs in a subgraph subsequence based on the connectivity between vertices with the vertex IDs in the subgraph subsequence. For example, consider the given the subgraph subsequence shown in Fig. 5(a). The vertices with vertex IDs 2 and 4 in the subgraph subsequence are not connected directly in any of the three elements. However, they are connected to the vertex with vertex ID 3 in the first and third elements, respectively. In this case, we consider the vertices 2 and 4 to be mutually relevant via the vertex 3. On the other hand, we can not know the relevancy between the vertex with vertex ID 1 and vertices with the other vertex IDs in the subgraph subsequence without accessing a huge set of original graph sequences DB containing this pattern, because the relevancy between 1 and the other vertex IDs is not provided in this pattern. Therefore, we aim to eliminate all vertices with vertex ID 1 in this pattern by defining the relevancy between vertex IDs in a subgraph subsequence.

Definition 2: Unique IDs in a graph sequence $d = \langle g^{(1)} g^{(2)} \dots g^{(l)} \rangle$ are relevant to one another, and d is called a “relevant” graph sequence, if the “union graph” $g_u(d)$ of d is a connected graph. Here, we define the union graph of d

as $g_u(d) = (V(g_u(d)), E(g_u(d)))$ such that

$$\begin{aligned} V(g_u(d)) &= \{id(v) \mid v \in V(g^{(j)}), g^{(j)} \in d\}, \text{ and} \\ E(g_u(d)) &= \{(id(v), id(v')) \mid (v, v') \in E(g^{(j)}), g^{(j)} \in d\}. \end{aligned}$$

In this definition, vertices and edges in a union graph do not have any labels, because we assume that the labels of the vertices and edges in a graph sequence can change. In addition, each element in a relevant graph sequence is not always connected. By limiting to frequent subgraph subsequence patterns, the union graphs of which are connected, we can know the relevancy between all pairs of vertex IDs in each of the frequent subgraph subsequence patterns without accessing the huge set of graph sequences DB containing the patterns.

Example 4: Figure 5(b) shows a union graph $g_u(d)$ of the graph sequence d shown in Fig. 5(a). The union graph has four vertices, because the graph sequence contains four distinct vertex IDs are included in the graph sequence. Additionally, the union graph contains two edges, because each of these edges exists in at least one of the elements in the graph sequence. The graph sequence d is irrelevant, because its union graph is not connected.

2.3 Induced Subgraph Subsequence

Any induced subgraph of a graph g can be obtained by deleting some of the vertices from g , along with any edges that connect to the deleted vertices. Thus, an induced subgraph is determined by its vertex set [9]. In this subsection, we extend the definition of an induced subgraph of a graph to a graph sequence to define an “induced subgraph subsequence”.

Definition 3: Let the graph sequence $\alpha = \langle a^{(1)} a^{(2)} \dots a^{(n)} \rangle$ be a subsequence of $\beta = \langle b^{(1)} b^{(2)} \dots b^{(m)} \rangle$, where $a^{(i)} \sqsubseteq b^{(j_i)}$, and u_α and u'_α be vertex IDs in α mapped, respectively, to vertex IDs u_β and u'_β in β by ϕ . The graph sequence α is an induced subgraph subsequence of the graph sequence β , denoted as $\alpha \sqsubseteq_i \beta$, if the following conditions are satisfied:

- a vertex with vertex ID u_α exists in $a^{(i)}$ if and only if a vertex with vertex ID u_β exists in $b^{(j_i)}$, and
- an edge connecting two vertices with vertex IDs u_α and u'_α exists in $a^{(i)}$ if and only if an edge connecting to vertices with vertex IDs u_β and u'_β exists in $b^{(j_i)}$. ■

Any induced subgraph subsequence of β can be obtained by deleting vertices with some of the vertex IDs from β , along with any edges that connect to the deleted vertices. Thus, an induced subgraph subsequence is determined by its vertex ID set.

Example 5: The graph sequence d_3 shown in Fig. 6 is an induced subgraph subsequence of the graph sequence d_1 shown in Fig. 4(a), whereas the graph sequence d_2 shown in Fig. 4(b) is not an induced subgraph subsequence of d_1 ,

Fig. 6 Induced subgraph subsequence d_3 of the graph sequence d_1 in Fig. 4.

because no edge exists between vertices with vertex IDs 1 and 2 in the second element of d_2 . The induced subgraph subsequence d_3 is determined by the vertex ID set $\{2, 3, 4\}$ in d_1 .

By limiting to the frequent subgraph subsequence patterns contained as induced subgraph subsequences in graph sequences in DB , we can know the relation between any two vertices in the frequent subgraph subsequence pattern without accessing the huge set of original graph sequences containing the pattern. If there is no edge between two vertices in an element of a subgraph subsequence pattern, the pattern states that there is no edge between two corresponding vertices in an element of many of the graph sequences in DB containing the pattern.

2.4 Problem Definition

Based on the inclusion relation ($\sqsubseteq_i$) of the induced subgraph subsequence, the definition of support of a graph sequence α is modified as

$$\sigma_i(\alpha) = |\{sid \mid (\langle sid, d \rangle \in DB) \wedge (\alpha \sqsubseteq_i d)\}|.$$

According to the definition of an induced subgraph subsequence, $\sigma_i(\alpha) \leq \sigma(\alpha)$ holds, because $\alpha \sqsubseteq_i \beta \Rightarrow \alpha \sqsubseteq \beta$. Therefore, the number of frequent subgraph subsequence patterns mined under the definition for $\sigma_i(\alpha)$ is less than or equal to that mined under the definition for $\sigma(\alpha)$.

By using the definitions of a union graph and induced subgraph subsequence, the problem of mining frequent subgraph subsequences is defined as follows.

Problem 1: Given a database $DB = \{\langle sid, d \rangle \mid d = \langle g^{(1)} \dots g^{(l)} \rangle\}$ and a minimum support threshold σ' as the input, the problem is to enumerate all frequent patterns $F = \{f \mid \sigma_i(f) \geq \sigma'\}$, the union graphs of which are connected. The frequent patterns are called Frequent, Relevant, and Induced Subgraph Subsequences (FRISs).

Example 6: Figure 7 shows a graph sequence database DB that contains two graph sequences and one of the FRISs mined from DB under $\sigma' = 2$. Vertices in the FRISS match with the hatched vertices in each of the graph sequences in DB .

The merits of mining only FRISs from a graph sequence database are given below.

- (1) FRISs are easily understandable, because the union graphs thereof are relevant, and the FRISs are included as induced subgraph subsequences in the graph sequences in DB .

Fig. 7 Mining FRISs from DB .

- (2) Since the total number of frequent subgraph subsequence (FSS) patterns with connected union graphs is much smaller than the number of all FSSs, we can reduce the computation time required to mine all FSSs with connected union graphs.
- (3) Since the total number of frequent subgraph subsequence patterns contained as induced subgraphs in the graph sequences in DB is much smaller than the number of all FSSs, we can reduce the computation time required to mine frequent and induced subgraph subsequence patterns.

3. Algorithm for Mining All FRISs

3.1 Principles of FRISS Mining

A straightforward method for mining all FRISs by extending the conventional graph mining algorithm recursively appends a vertex to a current FRISS, together with some edges that connect to the appended vertex. This method needs to append a vertex satisfying that the union graph of a frequent subgraph subsequence pattern is connected according to the definition of FRISS. However, as the depth of recursion increases, the computation time and the required memory also increase, because all sub-patterns of the FRISS are also frequent. For example, to mine the FRISS $\alpha = \langle a^{(1)} a^{(2)} \dots a^{(l)} \rangle$ shown in Fig. 8, the depth of recursion to append vertices in the method is 15, because the number of vertices contained in the pattern is 9, i.e., $\sum_{j=1, \dots, l} |V(a^{(j)})| = 9^\dagger$.

In this section, we propose an efficient algorithm for mining all FRISs from a graph sequence database. By using the definitions of relevancy of a graph sequence and an induced subgraph subsequence, the depth of recursion in our proposed method is given as the number of vertex IDs in a FRISS plus the number of elements in the FRISS. When mining the FRISS shown in Fig. 8, the depth of recursion of

[†]The FRISS shown in Fig. 8 is represented as $\langle (v_{[1,A]}^{(0,1)}, v_{[2,B]}^{(0,2)}, v_{[3,C]}^{(0,3)}, e_{[(1,2),-]}^{(0,4)}, e_{[(2,3),-]}^{(0,5)}, e_{[(3,1),-]}^{(0,6)})(v_{[4,D]}^{(1,1)}, e_{[(3,4),\bullet]}^{(1,2)}, e_{[(1,2),\bullet]}^{(1,3)}, e_{[(3,1),\bullet]}^{(1,4)}, v_{[1,\bullet]}^{(2,1)})(v_{[1,A]}^{(2,2)}, e_{[(1,2),-]}^{(2,3)}, e_{[(1,3),-]}^{(2,4)}, e_{[(3,4),\bullet]}^{(2,5)}, v_{[4,\bullet]}^{(2,6)}) \rangle$ by using transformation rules in the form of $tr_{(j,k)}^{[a,l]}$ proposed in [12]. GTRACE recursively appends a Transformation Rule (TR) 16 times when mining the pattern containing 16 TRs.

Fig. 8 A FRISS to be mined and its union graph.

the proposed method is $|V(g_u(\alpha))| + l = 4 + 3 = 7$, where l is the length of the FRISS α .

Another straightforward method for mining all FRISSEs by extending the conventional graph mining algorithm recursively appends an element to a current FRISS. In this case, we must append all possible induced subgraphs of $g^{(n')}$ to the tail of the FRISS $\alpha = \langle a^{(1)} \dots a^{(n)} \rangle$, where $\langle sid, d \rangle \in DB$, $g^{(n')} \in d$, $g^{(n'')} \in d$, $n' < n''$, and $g^{(n')}$ corresponds to $a^{(n)}$ under $\alpha \sqsubseteq d$. Since the number of all of the possible induced subgraphs of $g^{(n')}$ is too huge, we propose an efficient method which mines frequent, connected, and induced subgraphs from union graphs of DB in advance of mining FRISSEs.

To propose an efficient method for mining all FRISSEs from a graph sequence database, we introduce the following lemmas.

Lemma 1: Given two graph sequences α and β , if $\alpha \sqsubseteq_i \beta$, then the union graph $g_u(\alpha)$ of α is an induced subgraph of the union graph $g_u(\beta)$ of β . ■

Proof 1: $g_u(\alpha)$ is a subgraph of $g_u(\beta)$, because a vertex and edge which correspond to a vertex and edge in $V(g_u(\alpha))$ and $E(g_u(\alpha))$ exist in $V(g_u(\beta))$ and $E(g_u(\beta))$, respectively, and vertices and edges in the union graphs $g_u(\alpha)$ and $g_u(\beta)$ do not have any labels. In addition, any two vertices in the union graph $g_u(\alpha)$ are adjacent if and only if their corresponding vertices are adjacent in $g_u(\beta)$. Therefore, $\alpha \sqsubseteq_i \beta \Rightarrow g_u(\alpha) \sqsubseteq_i g_u(\beta)$. □

Lemma 2: If a graph sequence α is a FRISS among DB , its union graph $g_u(\alpha)$ is also a frequent, connected and induced subgraph pattern among a set of graphs $G_u = \{\langle sid, g_u(d) \rangle \mid \langle sid, d \rangle \in DB\}$. ■

Proof 2: If a graph sequence α is frequent, there are at least σ' graph sequences containing α among DB . Union graphs of these graph sequences contains the union graph of α as an induced subgraph according to Lemma 1. Therefore, the union graph $g_u(\alpha)$ of the graph sequence α is a frequent, connected and induced subgraph pattern among G_u . □

To enumerate all FRISSEs efficiently, we first generate union graphs for all graph sequences in DB based on the definition of a union graph. Subsequently, all frequent, “connected”, and “induced” subgraphs and their embeddings in these union graphs are enumerated by using the conventional Graph Mining algorithm. At every time when the algorithm outputs a frequent, connected, and induced subgraph, an altered version of PrefixSpan to be explained in Sect. 3.3 is called, with its input set of graph sequences generated by the projection given in the following definition.

Fig. 9 An example of projection.

Definition 4: Given a graph sequence $\langle sid, d \rangle \in DB$ and a connected and induced subgraph g of $g_u(d)$, we define a function “*proj*” to project the graph sequence d onto its maximum induced subgraph subsequences as follows:

$$\begin{aligned} \text{proj}(\langle sid, d \rangle, g) &= \{\langle sid, d', \phi \rangle \mid d' \sqsubseteq_i d \\ &\quad \wedge g_u(d') = g \wedge \perp \notin d' \\ &\quad \wedge \nexists d'' \text{ s.t. } d' \subset_i d'' \sqsubseteq_i d\}, \end{aligned}$$

where ϕ is an injective mapping from $V(g)$ to $V(g_u(d))$. ■

In this definition, $\perp \notin d'$ means that the graph sequence d' does not contain any element with an empty set of vertices. Since g is connected, a union graph of the projected graph sequence d' in $\text{proj}(\langle sid, d \rangle, g)$ is also connected. Therefore, vertex IDs in d' are relevant to one another.

Example 7: Figures 9(a) and (b) show the graph sequence $d = \langle g^{(1)} g^{(2)} g^{(3)} g^{(4)} \rangle$ and the union graph $g_u(d)$ for d , respectively. Given a graph g which is an induced subgraph of $g_u(d)$ as shown in Fig. 9(d), one of the projected graph sequences in $\text{proj}(\langle sid, d \rangle, g)$ is depicted in Fig. 9(c), where vertices in this projected graph sequence matches with the hatched vertices in Fig. 9(a).

Since the union graph of a FRISS is also frequent in the union graphs of all $\langle sid, d \rangle \in DB$ according to Lemma 2, we can enumerate all FRISSEs from the projected graph sequences if all frequent connected, and induced subgraphs among the union graphs of all $\langle sid, d \rangle \in DB$ are given.

The upper part of Fig. 10 shows an algorithm for enumerating all FRISSEs F from DB . First, a set G_u of the union graphs of graph sequences DB is generated in Line 3. We assume that the function call `FreqSubgraphMiner` in Line 4 repeatedly and exhaustively mines a frequent, connected and induced subgraph g and its embeddings H from G_u one by one. In this process, H is a set of injective functions mapping from $V(g)$ to $V(g_u(d))$, where $\langle sid, d \rangle \in DB$. Subsequently, in Line 5, graph sequences in DB are projected to obtain $\text{proj}(DB, g)$. In this projection, the injective functions H is provided as input from the external rather than it is computed within the projection. Then, `SeqPatternMiner` is called in Line 6 in Fig. 10 with the graph sequences projected in Line 5 to mine FRISSEs from $\text{proj}(DB, g)$. Finally, FRISSEs mined from $\text{proj}(DB, g)$ whose union graphs are isomorphic to g are added to F in Line 6. These processes are continued until the frequent, connected, and induced

```

1)  $FRISSMiner(DB, \sigma')$ 
2)  $F = \emptyset;$ 
3)  $G_u = \{\langle sid, g_u(d) \rangle \mid \langle sid, d \rangle \in DB\};$ 
4) for  $(g, H) = \text{FreqSubgraphMiner}(G_u, \sigma')$  until  $g \neq null$ 
5)    $proj(DB, g) = \bigcup_{\langle sid, d \rangle \in DB} proj(\langle sid, d \rangle, g, H);$ 
6)    $FSP = \text{SeqPatternMiner}(proj(DB, g), \sigma');$ 
7)    $F = F \cup \{\alpha \mid \alpha \in FSP \wedge g_u(\alpha) = g\};$ 
8) return  $F;$ 

1)  $proj(\langle sid, d \rangle, g, H)$ 
2)  $D_p = \emptyset;$ 
3) forall  $\phi \in H$ 
4)    $d' = \langle \rangle;$ 
5)   forall  $j = 1, 2, \dots, len(d)$ 
6)     if  $g^{(j)} \cap \phi(g) \neq \perp$ 
7)        $d' = d' \diamond (g^{(j)} \cap \phi(g));$ 
8)    $D_p = D_p \cup \{\langle sid, d', \phi \rangle\};$ 
9) return  $D_p;$ 

```

Fig. 10 Algorithm to mine FRISs.

subgraph g is exhausted in FreqSubgraphMiner . We have implemented FreqSubgraphMiner using AcGM [11] which is one of the conventional Graph Mining methods, because AcGM enumerates all embeddings in each union graph that are isomorphic with a frequent, connected, and induced subgraph g , thus enabling the efficient projection of g onto all of its subgraph subsequences $proj(\langle sid, d \rangle, g, H)$.

In addition, the lower part of Fig. 10 shows an algorithm for projecting a graph sequence to its subgraph subsequences. For each injective function in H , this algorithm generates a subgraph subsequence. In this algorithm, $\phi(g)$ is an induced subgraph of $g_u(d)$ induced by vertices $\{\phi(v) \mid v \in V(g)\}$, and $g^{(j)} \cap \phi(g)$ represents a common graph of $g^{(j)}$ and $\phi(g)$. In Line 7, a graph $(g^{(j)} \cap \phi(g))$ is appended to the tail of a graph sequence d' , which is denoted as $d' \diamond (g^{(j)} \cap \phi(g))$, if the graph is not an element with an empty set of vertices. Since the injective functions H is provided as input of this algorithm, the algorithm is solvable in linear time for the length of d and the size of H .

3.2 Recursively Appending an Element

By projecting the graph sequence database DB , we obtain a set of projected graph sequences each of whose vertex IDs are relevant to one another. The projected graph sequences become the inputs to SeqPatternMiner in Fig. 10. By recursively appending an element, but not a vertex, to a current FRISS in SeqPatternMiner , we can mine the complete set of FRISs with union graphs isomorphic to g from $proj(DB, g)$; this can be proven as follows.

Lemma 3: By recursively appending an element, but not a vertex, to a current FRISS in SeqPatternMiner , we can mine the complete set of FRISs with union graphs isomorphic to g from $proj(DB, g)$.

Proof 3: Let g be a frequent, connected, and induced subgraph mined from G_u , and α be a FRISS of length l mined from $proj(DB, g)$. Since union graphs of all FRISs mined from $proj(DB, g)$ are isomorphic to g , $g_u(\alpha) = g$. In ad-

dition, let α^- be a graph sequence of length l generated by deleting a vertex from the i -th element in α while keeping $g_u(\alpha^-)$ connected.

If $g_u(\alpha) \neq g_u(\alpha^-)$, then $g_u(\alpha^-)$ is not isomorphic to g , because $g = g_u(\alpha)$. Therefore, the frequent, connected, and induced subgraph g^- , where $g^- \sqsubset_i g$ and $g^- = g_u(\alpha^-)$ is mined from G_u , because the connected and induced subgraph of frequent connected subgraph is also frequent. Since α^- is an induced subgraph subsequence of α which is frequent, α^- is also frequent. Therefore, we do not need α^- to mine the FRISS α from $proj(DB, g)$, since α^- is mined from $proj(DB, g^-)$.

On the other hand, if the union graph of α is isomorphic to the union graph of α^- , i.e., $g_u(\alpha) = g_u(\alpha^-)$, let d^- be a graph sequence satisfying $\alpha^- \sqsubseteq_i d^-$ and $\langle sid, d^- \rangle \in proj(DB, g)$, where we assume that the i -th element in α^- corresponds to the j_i -th element in d^- . Even if $\alpha \sqsubseteq_i d^-$, the i -th element in α does not correspond to the j_i -th element in d^- according to the definition of the induced subgraph subsequence.

Therefore, an element corresponding to the i -th element in α exists between the j_{i-1} -th and j_{i+1} -th elements in d^- , or $\langle sid, d' \rangle$ satisfying $\alpha \sqsubseteq_i d'$ and $\alpha^- \not\sqsubseteq_i d'$ exists in $proj(DB, g)$. Accordingly, we do not need α^- to mine the FRISS α from $proj(DB, g)$, since α certainly be mined from $proj(DB, g^-)$, regardless of whether α^- is mined from $proj(DB, g)$.

For $g_u(\alpha) \neq g_u(\alpha^-)$ and $g_u(\alpha) = g_u(\alpha^-)$, we do not need to mine a graph sequence α^- with length equal to the length of the FRISS α to mine α from the projected graph sequence $proj(DB, g)$. Consequently, we can mine all FRISs with union graphs isomorphic to g from the projected graph sequences $proj(DB, g)$ by recursively appending an element, but not a vertex, to the current FRISS. $\square$

According to Lemma 3, we can mine all FRISs from DB by recursively mining frequent, connected, and induced subgraphs g from G_u . In addition, the lemma for the depth of recursion required to mine a FRISS from DB is obtained as follows.

Lemma 4: The depth of recursion required to mine a FRISS α from the graph sequence database DB by the proposed method is the number of vertex IDs in α plus the number of elements in α . $\blacksquare$

Proof 4: The proposed method recursively appends a vertex together with an edge (or some edges) by using AcGM in Line 3 of Fig. 10 to mine all frequent, connected, and induced subgraphs from G_u . Therefore, to mine a frequent, connected, and induced subgraph g containing $|V(g)|$ vertices from G_u , the method recursively appends a vertex $|V(g)|$ times, where $|V(g)|$ is equal to the number of vertex IDs in a FRISS α to be mined. In addition, because the method recursively appends an element by using PrefixSpan , the proposed method appends an element l times to mine the FRISS α containing l elements from $proj(DB, g)$. Therefore, the depth of recursion in the proposed method for mining the FRISS α is $|V(g)| + l$. $\square$

Fig. 11 Reassignment of vertex IDs.

The disadvantage of recursively appending an item or element based on the pattern growth principle is that often large amount of memory is required if the depth of recursion is very large. However, the depth of recursion in the proposed method is smaller than that of the method in which a vertex is recursively appended to a current FRISS. Therefore, we can efficiently mine all FRISSs from large and long graph sequences.

3.3 Isomorphism Matching of Elements

In the rest of this section, we explain that the complexity of checking whether the elements (graphs) in graph sequences are equivalent is $O(1)$ in our implementation. For example, we assume that DB contains only two graph sequences d_1 and d_2 and the two graph sequences d_1 and d_2 shown in Fig. 11 are projected onto $\langle sid_1, d'_1, \phi_1 \rangle$ and $\langle sid_2, d'_2, \phi_2 \rangle$ using a connected graph g . The projected graph sequences $\langle sid_1, d'_1, \phi_1 \rangle$ and $\langle sid_2, d'_2, \phi_2 \rangle$ are the inputs for SeqPatternMiner as shown in Fig. 10. First, SeqPatternMiner mines a FRISS α with one element $a^{(1)}$ from the projected graph sequences. To check whether the element $a^{(1)}$ is equivalent to an element $g_1^{(i)} \cap \phi_1(g)$ in d'_1 and $g_2^{(j)} \cap \phi_2(g)$ in d'_2 , we require $O(|V(g)|!)$, since $g_1^{(i)} \cap \phi_1(g)$ and $g_2^{(j)} \cap \phi_2(g)$ potentially contain $|V(g)|$ vertices, where $\phi_1(g)$ is an induced subgraph of $g_u(d'_1)$ induced by vertices $\{\phi_1(v) \mid v \in V(g)\}$, $g_1^{(i)} \cap \phi_1(g)$ represents a common graph of $g_1^{(i)}$ and $\phi_1(g)$, and $|V(g)|$ is the number of vertices in g . This check corresponds to graph isomorphism matching. However, since we know all embeddings of g in union graphs $g_u(d'_1)$ and $g_u(d'_2)$ of the projected graph sequences and also know their mappings from g to $g_u(d'_1)$ and $g_u(d'_2)$ according to Definition 4, we reassign all vertex IDs in the projected graphs sequences d'_1 and d'_2 as described below. For the sake of simplicity, we consider only two of the mappings from the graph g to the

Fig. 12 Mapping from a graph to graph sequences.

union graphs of the projected graph sequences d'_1 and d'_2 as shown in Fig. 12, although there are other mappings. The two mappings ϕ_1 and ϕ_2 are given as

$$\begin{aligned} \phi_1(1) &= 3, \phi_1(2) = 2, \phi_1(3) = 4, \\ \phi_2(1) &= 2, \phi_2(2) = 3, \phi_2(3) = 4. \end{aligned}$$

In this example, we reassign vertex ID 1 to all vertices with vertex ID 3 in d'_1 , because we know $\phi_1(1) = 3$. The re-assigned graph sequences d''_1 and d''_2 are shown at the bottom of Fig. 11. By reassigning the vertex IDs in d'_1 and d'_2 , we can check in $O(|V(g)|^2)$ whether the element $a^{(1)}$ in α is equivalent to $\phi_1^{-1}(g_1^{(i)} \cap \phi_1(g))$ in d'_1 and $\phi_2^{-1}(g_2^{(j)} \cap \phi_2(g))$ in d'_2 , because two corresponding vertices in any two elements have the same vertex IDs, where $\phi_1^{-1}(g_1^{(i)} \cap \phi_1(g))$ represents a graph whose vertex IDs are reassigned by ϕ_1 .

Moreover, since our proposed method recursively appends an element to a current FRISS based on the pattern growth principle, we compile each element in the reassigned graph sequences to an item by using a hash table that maps an element to an item. For example in Fig. 11, the reassigned graph sequences d''_1 and d''_2 are compiled into

$$\begin{aligned} \langle sid_1, \langle i_1 \ i_2 \ i_3 \ i_4 \rangle \rangle, \\ \langle sid_2, \langle i_1 \ i_2 \ i_4 \ i_4 \rangle \rangle, \end{aligned}$$

where the first elements in d''_1 and d''_2 are compiled to i_1 , the second elements to i_2 , and so on. Now, we have two sequences of items, so we require $O(1)$ to check whether the element $a^{(1)}$ represented by an item is equivalent to each element (item) in the item sequences[†]. After mining frequent sequence patterns containing items from the item sequences by using PrefixSpan, we decompile (reverse-compile) each item in the frequent sequence patterns to an element (graph) to obtain FRISSs.

By reassigning vertex IDs in projected graph sequences and compiling graph sequences to item sequences, we avoid graph isomorphism matching between two elements (graphs). Since we can quickly check whether two elements are equivalent by comparing two items, we can efficiently mine all FRISSs from large and long graph sequences.

4. Experiments

The proposed method was implemented in C++. The experiments were executed on an HP xw4600 with an Intel Core

[†]We require $O(|V(g)|^2)$ to check whether each element in a projected graph sequence is equivalent to a key in the hash table.

2 8600 3.33 GHz processor and 2 GB of main memory and running Windows XP. The performance of the proposed method was evaluated using artificial and real world graph sequence data.

4.1 Artificial Datasets

We evaluated the performance of the proposed method by using artificial datasets generated from the parameters listed in Table 1. First, starting from $g^{(1)}$ generated with edge existence probability p_e , we grew each graph sequence up to with $|V_{avg}|$ vertex IDs and with $|l_{avg}|$ elements on average by inserting, deleting and relabeling vertices and edges at each step. This process was carried out under the assumption of a low temporal resolution in observing graph sequences, *i.e.*, the change in each graph was NOT gradual. Similarly, we generated N FRISs with $|V'_{avg}|$ vertex IDs and $|l'_{avg}|$ elements on average. Then, we generated the DB where each graph sequence was overlaid by each FRIS with probability $1/N$. Each graph sequence contained $|L_v|$ vertex labels and $|L_e|$ edge labels.

Table 2 lists the computation time [sec] of FRISs, number of FRISs mined by the proposed method, computation time [sec] of FreqSubgraphMiner, number of frequent, connected, and induced subgraph patterns mined by SeqPatternMiner, computation time [sec] of SeqPatternMiner, and number of subgraph subsequences projected from the input of FRISMiner for varying values of $|DB|$, $|l'_{avg}|$, $|V_{avg}|$, and σ' , with the other parameters set to their default values.

The first part of Table 2 shows that the computation time is proportional to the number of graph sequences $|DB|$, as the case in conventional frequent pattern mining. The second and third parts of the table indicate that the computation time for the proposed method is exponential with respect to the average number of elements $|l'_{avg}|$ in the embedded FRISs and the average number of vertices $|V_{avg}|$ in each graph in the graph sequence database. The main reason that the computation time increases according to $|l'_{avg}|$ or $|V_{avg}|$ is considered to be the increase in the number of frequent patterns in both cases. The fourth part in Table 2 shows that the proposed method is tractable even with a low minimum support threshold.

Broadly speaking, the computation time of FRISMiner is found by adding the computation time of FreqSubgraphMiner to the computation time for n calls of SeqPatternMiner, where n is the number of frequent, connected, and induced subgraph patterns mined by AcGM. As mentioned in [11], the computation time of the original AcGM is proportional to the number of graphs in the input, and the number of patterns mined by AcGM is almost constant relative to the number of graphs in the input. In addition, the computation time and the number of mined patterns increase exponentially when the average number of vertices in the graphs increases or the minimum support threshold decreases. Therefore, the computation time needed by FreqSubgraphMiner in FRISMiner is propor-

Table 1 Parameters of artificial datasets.

Parameter	Default values
average number of vertices	
in each graph in graph sequences	$ V_{avg} = 7$
average number of elements	$ l_{avg} = 5$
average number of vertex IDs	
in embedded FRISs	$ V'_{avg} = 3$
average number of elements	
in embedded FRISs	$ l'_{avg} = 5$
number of vertex labels	$ L_v = 5$
number of edge labels	$ L_e = 1$
number of embedded FRISs	$N = 10$
number of graph sequences	$ DB = 1,000$
minimum support threshold	$\sigma' = 10\%$
edge existence prob. between 2 vertices	$p_e = 10\%$

tional to the number of union graphs G_u in Fig. 10, and increases exponentially when the average number of vertices in the union graphs increases or the minimum support threshold decreases. The problem of enumerating all frequent, connected, and induced subgraph patterns cannot be solved in time that is polynomial in the output, which is a class of algorithms that find all patterns in polynomial time both in the total input size and the number of patterns, unless $P = NP$ [1], [18].

On the other hand, the complexity of the original PrefixSpan is polynomial delay, which is a class for which the maximum computation time between two consecutive outputs is bounded by a polynomial in the total input size [1]. The input data for SeqPatternMiner in FRISMiner are a set of subgraph subsequences projected from the input of FRISMiner. The number of projected subgraph subsequences is the number of mappings from a subgraph pattern g mined by AcGM to the union graphs of graph sequences in the input of FRISMiner, and the average length of the projected subgraph subsequences is bounded by the average length of the graph sequences in the input of FRISMiner. The number of mappings is not always one when the union graph contains g . Therefore, the total computation time needed by SeqPatternMiner is not bounded by a polynomial in the total input size. However, by limiting the frequent subgraph subsequences to relevant and induced subgraph subsequences only, the proposed method efficiently mines a complete set of FRISs from a set of graph sequences. Additionally, since the proposed method checks in $O(1)$ time whether elements (graphs) in graph sequences are equivalent, the efficiency of mining FRISs from graph sequences is further enhanced. Therefore, the proposed method is practical for graph sequences that are both long and large.

4.2 Real World Dataset

To assess the practicality of the proposed method, it was applied to the Enron Email Dataset [6]. In the dataset, we assigned a vertex ID to each person participating in email communication, and assigned an edge to a pair communicating via email on a particular day, thereby obtaining a daily graph $g^{(j)}$. In addition, one of the vertex labels {CEO, Employee, Director, Manager, Lawyer, President, Trader, Vice

Table 2 Results for various $|DB|$, $|l'_{avg}|$, $|V_{avg}|$, and σ' .

$ DB $	comp. time of FRISSs	# of FRISSs	comp. time of AcGM	# of freq. pat. by AcGM	comp.time of SeqPatternMiner	# of subgraph subsequences
1,000	1.1	631	0.5	11	0.2	143,223
3,000	5.9	755	3.8	11	0.7	445,042
5,000	10.2	1,229	5.6	11	2.0	833,268
7,000	14.9	983	9.4	11	2.2	1,139,341
9,000	17.7	803	10.7	11	2.6	1,373,871
20,000	43.0	713	28.6	11	5.7	3,109,734
40,000	94.7	923	64.5	11	12.8	6,534,150
60,000	148.7	962	99.1	11	22.2	10,124,834
80,000	191.9	817	129.3	11	27.4	12,814,847
100,000	207.0	734	149.9	11	22.2	14,135,877
$ l'_{avg} $	comp. time of FRISSs	# of FRISSs	comp. time of AcGM	# of freq. pat. by AcGM	comp.time of SeqPatternMiner	# of subgraph subsequences
5	1.1	631	0.5	11	0.2	143,223
7	2.5	1,724	1.0	11	0.9	187,285
9	4.9	5,092	1.6	18	2.0	284,235
10	7.0	8,644	2.0	20	3.3	365,462
12	8.1	18,170	1.6	24	4.8	361,111
14	12.3	35,424	1.3	14	8.9	378,442
16	29.2	91,215	3.0	19	22.7	544,116
18	52.2	183,233	3.2	22	45.0	514,271
20	114.3	437,611	4.0	29	104.5	742,328
25	843.1	2,774,538	11.2	22	818.9	1,268,167
$ V_{avg} $	comp. time of FRISSs	# of FRISSs	comp. time of AcGM	# of freq. pat. by AcGM	comp.time of SeqPatternMiner	# of subgraph subsequences
5	0.3	644	0.2	9	0.1	36,421
6	0.7	624	0.3	9	0.1	71,313
7	1.1	631	0.5	11	0.2	143,223
8	2.4	728	1.1	11	0.7	225,977
9	3.9	735	1.7	13	1.2	387,393
10	9.2	1,003	5.0	12	2.3	589,946
12	34.8	2,020	17.0	24	12.9	1,631,136
14	82.1	3,359	32.7	18	41.0	2,942,595
16	186.7	3,908	66.7	26	104.0	5,530,720
σ'	comp. time of FRISSs	# of FRISSs	comp. time of AcGM	# of freq. pat. by AcGM	comp.time of SeqPatternMiner	# of subgraph subsequences
10	1.1	631	0.5	11	0.2	143,223
8	1.4	664	0.7	11	0.3	143,439
6	1.5	731	0.6	11	0.4	144,728
4	2.2	1,030	1.0	14	0.6	158,476
2	2.7	2,208	1.2	18	1.0	167,905
0.9	4.8	6,294	1.6	29	2.4	196,848
0.7	4.6	10,955	0.9	45	3.1	212,301
0.5	7.0	16,529	1.8	56	4.5	225,570

comp. time of FRISSs: the computation time [sec] of FRISSs.

of FRISSs: the number of FRISSs mined by the proposed method.

comp. time of AcGM: the computation time [sec] of FreqSubgraphMiner.

of freq. pat. by AcGM: the number of frequent, connected, and induced subgraphs mined by SeqPatternMiner.

comp.time of SeqPatternMiner: the computation time [sec] of SeqPatternMiner.

of graph sequences: the number of subgraph subsequences projected from the input of FRISSMiner.

President} was assigned to each vertex. We then obtained a set of weekly graph sequence data, *i.e.*, a *DB*. The total number of weeks, *i.e.*, number of sequences, was 200. We randomly sampled $|V| (= 1 \sim 182)$ persons to form each *DB*.

Table 3 shows the computation time (comp. time [sec]), number of FRISSs mined by the proposed method or rFTSs mined by GTRACE (# of FRISSs or # of rFTSs), and computation time per FRISSs or rFTS (comp./FRISS or comp./FTS) obtained for various numbers of vertex IDs (persons) $|V|$, minimum support σ' , and numbers of elements $|l_{avg}|$ in each graph sequence for the dataset. All the other parameters were set to default values indicated at the

bottom of the table. Thus, the dataset with the default values contained 124 graph sequences each consisting of 182 persons (vertex IDs) and 7 elements. The parameter $|l_{avg}| = 4, 5, 6$, or 7 indicates that each sequence d in *DB* consists of 4, 5, 6, or 7 elements from Monday to Thursday, Friday Saturday, or Sunday, respectively. In addition, PM and GT denote the proposed method and GTRACE, respectively. Where the required computation time exceeds one hour or a memory overflow occurs, the results are indicated by “–”.

The upper, middle, and lower parts of the table show the practical scalability of the proposed method with regard to the number of persons (vertex IDs), the minimum support

Table 3 Results for the Enron dataset.

# of persons $ V $	80	110	120	130	140	150	160	170	182
PM comp. time	0.047	0.39	0.88	0.86	0.99	4.6	9.7	20.0	34.5
# of FRISSs	150	451	720	811	827	1230	1573	1993	2255
comp./FRISS	3.1E-4	8.6E-4	12E-3	1.1E-3	1.2E-3	3.7E-3	6.2E-3	1.0E-2	1.5E-2
GT comp. time	0.13	1913.8	-	-	-	-	-	-	-
# of rFTSs	290	15681	-	-	-	-	-	-	-
comp./FTS	4.3E-4	1.2E-1	-	-	-	-	-	-	-
min. sup. σ' [%]	60	55	50	45	40	30	20	15	
PM comp. time	0.27	0.38	0.39	0.44	0.75	2.5	10.9	34.5	
# of FRISSs	50	87	125	166	242	531	1324	2255	
comp./FRISS	5.3E-3	3.7E-3	3.1E-3	2.6E-3	3.1E-3	4.6E-3	8.2E-3	1.5E-2	
GT comp. time	5.72	50.0	79.22	-	-	-	-	-	
# of rFTSs	267	601	1376	-	-	-	-	-	
comp./FTS	2.1E-2	8.3E-2	5.8E-2	-	-	-	-	-	
# of elements $ l_{avg} $	3	4	5	6	7				
PM comp. time	0.45	2.99	9.23	24.30	34.52				
# of FRISSs	163	413	898	1498	2255				
comp./FRISS	2.7E-3	7.2E-3	1.0E-2	1.6E-2	1.5E-2				
GT comp. time	-	-	-	-	-				

Default: minimum support $\sigma' = 15\%$, # of vertex labels $|L_v| = 8$, # of edge labels $|L_e| = 1$, # of persons $|V| = 182$, # of elements $|l_{avg}| = 7$,

PM: Proposed Method (FRISSMiner), GT: GTRACE, comp./FRISS (or rFTS): comp. time per FRISS (or rFTS) [sec]

threshold, and the number of elements in graph sequences in the graph sequence database, respectively. GTRACE proved intractable for the graph sequence dataset generated from the default values, although change in each graph in this graph sequence database is gradual. On the other hand, the proposed method is tractable with respect to the database. Although we used the same graph sequence database to mine frequent patterns, the resulting numbers of FRISSs and rFTSs are very different due to the definitions of a FRISS and rFTS. Accordingly, we also use the computation time per FRISS or rFTS in the comparison. Good scalability of the proposed method is indicated in Table 3, since the computation time per FRISS for the proposed method are smaller than those per rFTS for GTRACE. The scalability of the proposed method comes from limiting frequent subgraph subsequence patterns to relevant and induced subgraph subsequences only and isomorphism matching between elements as mentioned in Sect. 3.3. In addition, the proposed method is practicable, because frequent subgraph subsequences mined by the proposed method are easily understandable according to the definitions of relevant and induced subgraph subsequences.

5. Discussion and Conclusion

Recently, much attention has been paid to frequent pattern mining from graph sequences [12] (dynamic graphs [3] or evolving graphs [2]). In [12], GTRACE focuses on the differences between two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in a graph sequence to compactly represent the graph sequence under the assumption that the change in each graph is gradual. The differences between graphs $g^{(j)}$ and $g^{(j+1)}$ are interpolated by a virtual graph sequence such that the edit distance between any two successive graphs in the virtual sequence is 1 and the edit distance between any two graphs is the minimum. Two successive graphs in the virtual se-

Table 4 Characteristics of graph sequences.

	graph sequence	dynamic graph	evolving graph
# of vertices	increase & decrease	constant	increase [†]
# of edges	increase & decrease	increase & decrease	increase [†]
labels of vertices	assigned & changed	assigned & not changed	assigned & not changed
labels of edges	assigned & changed	not assigned	assigned & not changed

[†]: A vertex in the evolving graph always comes with an edge connected to the vertex.

quence are represented by one of 6 transformation rules, namely vertex insertion (vi), vertex deletion (vd), vertex relabeling (vr), edge insertion (ei), edge deletion (ed), and edge relabeling (er). For example, the differences between two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in the graph sequence shown in Fig. 3 is represented as an ordered list of two transformation rules $\langle vi_{[1,A]}^{(j,1)}, ed_{[(2,3),\bullet]}^{(j,2)} \rangle$ as discussed in Sect. 1.3. Based on the assumption that the change in each graph is gradual, GTRACE can compactly represent a graph sequence using the transformation rules even if each graph in the graph sequence has many vertices and edges.

In [3], Borgwardt et al. proposed a method for mining frequent patterns from a graph sequence represented by the dynamic graph. They assumed that the number of edges in a dynamic graph increases and decreases while the number of vertices remains constant. They also assumed that labels assigned to vertices in the dynamic graph do not change and that no label is assigned to edges. The characteristics of the dynamic graph are summarized in the third column of Table 4. In this method, the existence and non-existence of an edge in each element is represented by 1 and 0, respectively, and an edge is labeled with the binary string consisting of these 1 and 0 as shown in the upper part of Fig. 13. In this

Fig. 13 Dynamic graph and evolving graph.

figure, the labels of vertices in the dynamic graph are omitted for the sake of simplicity.

On the other hand, Berlingerio et al. proposed a method for mining frequent patterns from a graph sequence represented by the evolving graph [2]. They assumed that the numbers of vertices and edges in an evolving graph increase but do not decrease and that labels assigned to vertices and edges in the dynamic graph do not change. In addition, a vertex in an evolving graph always comes with an edge connected to the vertex. The characteristics of the evolving graph are summarized in the fourth column of Table 4. In this method, the graph sequence in the lower left of Fig. 13 is represented as an evolving graph in the lower right of Fig. 13, where labels of vertices and edges in the evolving graph are omitted. The number attached to each edge in the evolving graph depicts the element in which the edge appears. Although these two methods can represent a complex graph sequence using a graph, they can not represent graph sequences where the numbers of vertices and edges increase and decrease and labels change, such as in human communities and gene networks mentioned in Sect. 1. Therefore, graph sequences addressed in this paper and in [12] belong to a more general class than the dynamic graph and evolving graph.

Both the method proposed in this paper and GTRACE are applicable to the general class of graph sequences with the characteristics shown in the second column of Table 4. The left hand side in Fig. 14 shows 4 frequent patterns called FTSs (Frequent Transformation Subsequences) mined by GTRACE under $\sigma' = 2$ from the graph sequence database consisting of 2 graph sequences shown in the upper part of Fig. 14, while the right hand side in Fig. 14 shows 9 FSSs (Frequent Subgraph Subsequences) embedded in the graph database. It does not make sense to directly compare the relationship between the mined FTSs and FSSs, because FTSs represent sequences of common changes in the graph sequence database, whereas FSSs represent sequences of common structures therein. For example, the first FTS indicated by a dashed arrow in Fig. 14 indicates that a vertex with label A is inserted in some element but it is not known in how many elements the vertex appears. On the other hand, the second FSS indicated by another dashed arrow means that a vertex with label A appears in at least two elements in many of the graph sequences in the graph sequence database.

In this paper, we proposed a novel method for min-

Fig. 14 Mining FSSs and FTSs from graph sequences.

ing all frequent, relevant, and induced subgraph subsequences from given graph sequences. We developed a graph sequence mining program, and confirmed the efficient and practical performance of the proposed method through computational experiments using artificial and real world datasets. The method proposed in this paper efficiently enumerates all FRISSs from a set of graph sequences, while methods in [2], [3] mine all frequent patterns from a long graph sequence. In [7], [14], it is shown that the principle of enumerating possible patterns can be distinguished from the principle of counting support values of the patterns. Therefore, the proposed method in this paper can be extended to mine FRISSs from a long and large graph sequence.

References

- [1] H. Arimura and T. Uno, "Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems," Proc. SIAM International Conference on Data Mining, pp.1087–1098, 2009.
- [2] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis, "Mining graph evolution rules," Proc. European Conference on Principles and Practice of Knowledge Discovery in Databases, pp.115–130, 2009.
- [3] K.M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther, "Pattern mining in frequent dynamic subgraphs," Proc. International Conference on Data Mining, pp.818–822, 2006.
- [4] D.J. Cook, L.B. Holder, J. Coble, and J. Potts, "Graph-based mining of complex data," Advanced Methods for Knowledge Discovery from Complex Data, pp.75–93, 2005.
- [5] D.J. Cook and L.B. Holder, Mining Graph Data, Wiley-Interscience, 2006.
- [6] Enron Email Dataset, <http://www.cs.cmu.edu/~enron/>
- [7] M. Fiedler and C. Borgelt, "Subgraph support in a single large graph," Workshops Proc. Workshop on Mining Graphs and Complex Structures, pp.399–404, 2007.
- [8] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.
- [9] C. Godsil and G. Royle, Algebraic Graph Theory Springer, 2001.
- [10] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," Proc. European Conference on Principles of Data Mining and Knowledge Discovery, pp.13–23, 2000.
- [11] A. Inokuchi, T. Washio, Y. Nishimura, and H. Motoda, "A fast al-

gorithm for mining frequent connected subgraphs,” IBM Research Report, RT0448, 2002.

- [12] A. Inokuchi and T. Washio, “A fast method to mine frequent subsequences from graph sequence data,” *Proc. IEEE International Conference on Data Mining*, pp.303–312, 2008.
- [13] J.P. Kukluk, L.B. Holder, and D.J. Cook, “Inference of node replacement graph grammars,” *Intelligent Data Analysis*, vol.11, no.4, pp.377–400, 2007.
- [14] M. Kuramochi and G. Karypis, “Finding frequent patterns in a large sparse graph,” *Proc. SIAM International Conference on Data Mining*, 2004.
- [15] S. Nijssen and J.N. Kok, “A quickstart in frequent structure mining can make a difference,” *Proc. International Conference on Knowledge Discovery and Data Mining*, pp.647–652, 2004.
- [16] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, “PrefixSpan: Mining sequential patterns by prefix-projected growth,” *Proc. International Conference on Data Engineering*, pp.2–6, 2001.
- [17] A. Sanfeliu and K.-S. Fu, “A distance measure between attributed relational graphs for pattern recognition,” *IEEE Trans. Syst. Man Cybern.*, vol.13, no.3, pp.353–362, 1983.
- [18] T. Horváth, B. Bringmann, and L. De Raedt, “Frequent hypergraph mining,” *Proc. International Conference on Inductive Logic Programming*, pp.244–259, 2006.
- [19] T. Washio and H. Motoda, “State of the art of graph-based data mining,” *SIGKDD Explorations*, vol.5, Issue 1, pp.59–68, 2003.
- [20] X. Yan and J. Han, “gSpan: Graph-based substructure pattern mining,” *Proc. IEEE International Conference on Data Mining*, pp.721–724, 2002.

Takashi Washio received the PhD degree in nuclear engineering from Tohoku University, Japan, in 1983, on the topic of process plant diagnosis based on qualitative reasoning. He is a professor in the Institute of Scientific and Industrial Research (ISIR), Osaka University. At ISIR, he works on the study of scientific discovery, graph mining, and high-dimensional data mining. He received the best paper award from the Atomic Energy Society of Japan in 1996, the best paper award from the Japanese Society for

Artificial Intelligence in 2001, and the Journal Award of Computer Aided Chemistry in 2002, Contribution Award from the Japanese Society for Artificial Intelligence in 2009. He is a member of the IEEE Computer Society. Department of Reasoning for Intelligence, The Institute of Science and Industrial Research, Osaka University, 8-1 Mihogaoka Ibaraki, Osaka, 567-0047 Japan.

Akihiro Inokuchi received the PhD degree in communication engineering from Osaka University, Japan, in 2004. He is an assistant professor in the Institute of Scientific and Industrial Research (ISIR), Osaka University, and also a PRESTO Researcher in Japan Science and Technology Agency. At ISIR, he works on the study of data mining, machine learning, and multidimensional database. He received the best paper award from the Journal Award of Computer Aided Chemistry in 2002, and the incentive

awards from Japanese Society for Artificial Intelligence in 2004 and 2008. Department of Reasoning for Intelligence, The Institute of Science and Industrial Research, Osaka University, 8-1 Mihogaoka Ibaraki, Osaka, 567-0047 Japan.