PAPER

GTRACE: Mining Frequent Subsequences from Graph Sequences**

Akihiro INOKUCHI†*a) *and* **Takashi WASHIO**†, *Members*

SUMMARY In recent years, the mining of a complete set of frequent subgraphs from labeled graph data has been studied extensively. However, to the best of our knowledge, no method has been proposed for finding frequent subsequences of graphs from a set of graph sequences. In this paper, we define a novel class of graph subsequences by introducing axiomatic rules for graph transformations, their admissibility constraints, and a union graph. Then we propose an efficient approach named "GTRACE" for enumerating frequent transformation subsequences (FTSs) of graphs from a given set of graph sequences. The fundamental performance of the proposed method is evaluated using artificial datasets, and its practicality is confirmed by experiments using real-world datasets.

key words: *frequent pattern mining, graph sequence, union graph*

1. Introduction

Data mining research in recent years has resulted in the development of many approaches for finding characteristic patterns in a variety of structured data. Sequential Pattern Mining, such as AprioriSome [1] and PrefixSpan [17], is used to efficiently find a complete set of subsequences that appear more frequently than a minimum support threshold in a given set of itemset sequences. These approaches are limited to mining total order relations among itemsets representing the co-occurrence of events. In contrast, WINEPI and MINEPI [14], more advanced approaches to Sequential Pattern Mining, find frequent episodes representing partial order relations among event occurrences. However, the relations are still limited to orders and do not cover any semantic and/or topological issues.

Graph Mining, which efficiently mines all subgraphs appearing more frequently than a given threshold from a set of graphs, focuses on the topological relations among events [21]. AGM [7], gSpan [22], and Gaston [16] mine frequent subgraphs levelwise starting from those of size 1, by using the anti-monotonic property of the support values.

AGM [7] enumerates candidate patterns stepwise by adding an extra vertex, whereas FSG [11], gSpan [22], and Gaston [16] enumerate patterns by adding an edge. Moreover, AGM and FSG use a breadth-first search, while gSpan and FFSM incorporate depth-first search algorithms. Gaston is known to be the fastest algorithm. Its efficiency stems from the use of the sparsity of graphs in most real-world applications. It first figures out frequent free trees embedded in the graph data while avoiding redundant enumerations and then extends these to frequent subgraphs by adding some loops. Despite the main algorithms for Graph Mining being quite efficient in practice, they do require much computation time when mining complex frequent subgraphs due to the NP-completeness of subgraph isomorphism matching [5]. Accordingly, these conventional methods are not well suited to more complex graphs such as graph sequences [8].

Nevertheless, in many real-world applications, objects are modeled using graph sequences. For example, a human network can be represented as a graph in which humans and the relationships between them correspond, respectively, to the vertices and edges. If a person joins or leaves a community, the numbers of vertices and edges in the graph increase or decrease accordingly. Similarly, a gene network consisting of genes and their interactions produces a graph sequence in their evolutionary history. The distribution of vertex degrees in most of these real-world graphs is known to have a long tail with most of the vertices having only a few edges [2], and thus such graphs are sparse. We focus on the topological changes in sequences of sparse graphs, because in many cases their changes are caused by some underlying mechanism restricted by the topology, such as distances (dissimilarities) between vertices and information traveling on the graphs (networks).

The primary objective of this paper is to establish a novel framework for mining a complete set of frequent subsequences embedded in a given set of observed sequences of sparse graphs. We introduce a novel representation for graph sequences where each change between two observed successive graph states is interpolated by axiomatic transformation rules that follow their associated admissibility constraints. We further propose a new method for mining subsequences called "frequent transformation subsequences (FTSs)", represented by transformation rules of the graph sequence data. A secondary objective is to mine FTSs corresponding to "relevant" graph subsequences based on "union graphs", which represent relevant vertices in graph sequences. The relevant FTSs are important, since, in many cases, we focus on the relations among relevant persons, events, phenomena, and so on. Our proposed approach based on these principles is called "GTRACE (Graph TRAnsformation sequenCE mining)". The final objective is

---

Manuscript received February 23, 2010.
Manuscript revised June 16, 2010.

†The authors are with the Institute of Scientific and Industrial Research, Osaka University, Ibaraki-shi, 567–0047 Japan.
*Presently, with PRESTO, Japan Science and Technology Agency.

**The main idea of this paper was presented at the International Conference on Data Mining in 2008.

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

DOI: 10.1587/transinf.E93.D.2792

to characterize experimentally the efficiency of GTRACE using artificially generated data and to demonstrate its practicality by applying it to real-world datasets.

The rest of this paper is organized as follows. Section 2 introduces the interpolation of observed graph changes, the transformation rules and their associated constraints, and also provides an algorithm for compiling graph sequences into transformation sequences. In Sect. 3 we propose the principles of and an algorithm for mining FTSs from the transformation sequences, while in Sect. 4 we further propose the mining of relevant FTSs. The fundamental performance and practicality of GTRACE are demonstrated through experiments in Sect. 5. Section 6 presents a discussion, and Sect. 7 concludes this paper.

2. Principles for Representing Graph Sequences

Figure 1 (a) shows an example of an observed graph sequence. A graph $g^{(j)}$ is the j-th labeled graph in the sequence. Here, we introduce two practical **Assumptions**. The first is that "change is gradual"; that is, between two successive graphs $g^{(j)}$ and $g^{(j+1)}$ only a small part of the structure changes while the rest remains unchanged. The other assumption is that "every graph $g^{(j)}$ is sparse". In the aforementioned examples of human networks and gene networks, these assumptions certainly hold, since most of the changes to the vertices are progressive, and the vertices are not very densely coupled to one another at any step. Although this paper focuses on undirected graphs only, our proposed principles are also applicable to directed graphs without loss of generality.

A compact representation of graph sequences is needed to reduce both the computational and spatial cost in mining the frequent transformation subsequences (FTSs). The direct representation of a graph sequence is not compact, since many parts of a graph remain unchanged over multiple steps and are therefore redundant in the representation.

One way of deriving a compact representation is by using a Graph Grammar [12], which is a collection of rules for transforming one graph into another. In concrete terms, the Graph Grammar rule $S \rightarrow P$, where S and P are graphs, transforms subgraphs that are isomorphic with S in a given graph to P. For example, given the Graph Grammar rule shown in Fig. 2 and the graph $g^{(j)}$ in a graph sequence, a subgraph that is isomorphic with S in the graph $g^{(j)}$ is transformed to P to produce another graph $g^{(j+1)}$. A few data mining studies have incorporated a Graph Grammar. Recently, Holder and Cook incorporated a Graph Grammar into their modified SUBDUE, which iteratively replaces a frequent subgraph S in a graph to a vertex v using the rule $S \rightarrow v$ based on the Minimum Description Length [10], [13]. Because the head of the rule consists of a single vertex, this technique is not applicable to general graph subsequence mining. In concrete terms, given two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in a graph sequence, we cannot represent transformations between $g^{(j)}$ and $g^{(j+1)}$ using rules of the form $S \rightarrow v$, since we assume that the numbers of vertices and edges in the graphs in the graph sequence can increase as well as decrease. On the other hand, since the general framework of a Graph Grammar is too generic to derive a compact representation of graph sequences, we need a large set of rules to represent transformations between any two successive graphs in a graph sequence.

Accordingly, we propose the following novel graph grammatical framework adapted to describe a graph sequence easily and compactly by introducing rules of insertion, deletion, and relabeling of vertices and edges under the assumption of gradual change.

2.1 Representation of Graph Sequences

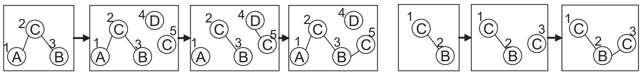
A labeled graph g is represented as $g = (V, E, L, f)$, where $V = \{v_1, v_2, \cdots, v_z\}$ is a set of vertices, $E = \{(v, v') \mid (v, v') \in V \times V\}$ is a set of edges, and L is a set of labels determined by the function $f : V \cup E \rightarrow L$. $V(g)$, $E(g)$, and $L(g)$ are sets of vertices, edges and labels of g, respectively. An observed graph sequence is represented as $d = \langle g^{(1)}\ g^{(2)} \cdots g^{(n)} \rangle$, where the integer superscript of each g represents the ordered step in the observation. The j-th graph included in d is represented as $g^{(j)} \in d$. We assume that each vertex v is mutually distinct from all other vertices in any $g^{(j)}$, and has a vertex ID $id(v)$ in d. We define a set of vertex IDs $ID_V(d)$ and a set of pairs of vertex IDs $ID_E(d)$ as follows.

$$\begin{aligned} ID_V(d) &= \{id(v) | v \in V(g^{(j)}), g^{(j)} \in d\}, \\ ID_E(d) &= \{(id(v), id(v')) | (v, v') \in E(g^{(j)}), g^{(j)} \in d\}. \end{aligned}$$

Example 1: In the human network mentioned in Sect. 1, each person has a vertex ID, and his/her gender is an example of a vertex label.

To represent a graph sequence compactly, we focus on the differences between two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in the sequence.

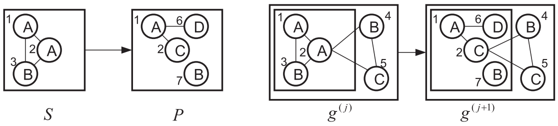
Definition 1: Each observed graph $g^{(j)}$ in a graph sequence $d = \langle g^{(1)} g^{(2)} \cdots g^{(n)} \rangle$ is called an "interstate". The differences between interstates $g^{(j)}$ and $g^{(j+1)}$ in d are interpolated by a virtual sequence $\langle g^{(j,1)} g^{(j,2)} \cdots g^{(j,m_j)} \rangle$, where $g^{(j,1)} = g^{(j)}$ and $g^{(j,m_j)} = g^{(j+1)}$, and each graph $g^{(j,k)}$ in the sequence is



(a) An example of a graph sequence

(b) One of mined graph subsequences

Fig. 1 A graph sequence and a mined graph subsequence.



(a) A rule to transform a graph

(b) Two successive graphs in a graph sequence

Fig. 2 A rule of Graph Grammar.

Table 1 Transformation rules (TRs) for representing graph sequence data.

Vertex Insertion $vi^{(j,k)}_{[u,l]}$	Insert a vertex with label l and vertex ID u into $g^{(j,k)}$ to transform to $g^{(j,k+1)}$.
Vertex Deletion $vd^{(j,k)}_{[u,\bullet]}$	Delete an isolated vertex with vertex ID u in $g^{(j,k)}$ to transform to $g^{(j,k+1)}$.
Vertex Relabeling $vr^{(j,k)}_{[u,l]}$	Relabel the label of a vertex with vertex ID u in $g^{(j,k)}$ to be l to transform to $g^{(j,k+1)}$.
Edge Insertion $ei^{(j,k)}_{[(u_1,u_2),l]}$	Insert an edge with label l between 2 vertices with vertex IDs u_1 and u_2 into $g^{(j,k)}$ to transform to $g^{(j,k+1)}$.
Edge Deletion $ed^{(j,k)}_{[(u_1,u_2),\bullet]}$	Delete an edge between 2 vertices with vertex IDs u_1 and u_2 in $g^{(j,k)}$ to transform to $g^{(j,k+1)}$.
Edge Relabeling $er^{(j,k)}_{[(u_1,u_2),l]}$	Relabel the label of an edge between 2 vertices with vertex IDs u_1 and u_2 in $g^{(j,k)}$ to be l to transform to $g^{(j,k+1)}$.

called an "intrastate". The observed graph sequence ("interstate sequence") d is represented by the interpolations as $d = \langle s^{(1)} s^{(2)} \cdots s^{(n-1)} \rangle$. ■

The order of interstates represents the order of graphs in an observed sequence. On the other hand, the order of intrastates is the order of graphs in the artificial interpolation, with various interpolations possible between graphs $g^{(j)}$ and $g^{(j+1)}$. To ensure that the interpolations are compact and unambiguous, we select the one with the shortest length in terms of graph edit distance [18], thereby reducing both the computational and spatial cost.

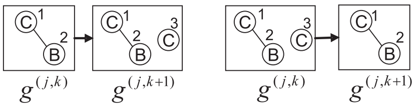
Definition 2: Let the transformation of a graph by either insertion, deletion or relabeling of a vertex or an edge be a unit, and let each unit have an edit distance of 1. An "intrastate sequence" $s^{(j)} = \langle g^{(j,1)} g^{(j,2)} \cdots g^{(j,m_j)} \rangle$ is defined as an interpolation where the edit distance between any two successive intrastates is 1 and the edit distance between any two intrastates is the minimum. ■

Transformations are represented in this paper by the following "transformation rule (TR)".

Definition 3: A transformation rule (TR) transforming $g^{(j,k)}$ to $g^{(j,k+1)}$ is expressed as $tr^{(j,k)}_{[o_{jk},l_{jk}]}$, where

- tr is a transformation type, which is either insertion, deletion, or relabeling of a vertex or an edge,
- o_{jk} is the element in $ID_V(d) \cup ID_E(d)$ to which the transformation is applied, and
- $l_{jk} \in L$ is a label to be assigned to the vertex or edge in the transformation. ■

For the sake of simplicity, we simplify the transformation rule $tr^{(j,k)}_{[o_{jk},l_{jk}]}$ to $tr^{(j,k)}_{[o,l]}$ by omitting the subscripts of o_{jk} and l_{jk}, except in cases where the original notation is essential. We introduce six TRs in Table 1. For example, the transformation $vi^{(j,k)}_{[u,l]}$ inserts a vertex with vertex ID u and label l between the k-th and $k+1$-th intrastates in the j-th interstate. Since the transformations by vertex deletion vd and edge deletion ed do not assign any labels to the vertex or edge respectively, their arguments l are dummy and are represented by '•' without loss of generality. The reader may notice that a relabeling transformation can be decomposed into a deletion and subsequent insertion of a vertex or edge with the same o_{jk}. However, we retain the relabeling transformation, because it increases the compactness of the graph sequences and the mined frequent transformation subsequences (FTSs).



(a) Vertex insertion (b) Vertex deletion

Fig. 3 Examples of transformations.

In the remainder of this subsection, we introduce axioms governing the transformation rules and the Admissibility Theorem for the transformation rules to prove that any graph sequence can be represented by the six transformation rules given in Table 1. The transformation rules for vertex insertion and vertex deletion are governed by the following axioms due to their nature.

Axiom 1: A transformation rule $vi^{(j,k)}_{[u,l]}$ is used to transform $g^{(j,k)}$ to $g^{(j,k+1)}$ if and only if

- a vertex with vertex ID u is not contained in $g^{(j,k)}$, and
- a vertex with label l and vertex ID u is contained in $g^{(j,k+1)}$. ■

For example, in Fig. 3 (a), $g^{(j,k)}$ is transformed to $g^{(j,k+1)}$ by inserting a vertex with vertex ID 3, where the number attached to each vertex denotes the vertex ID of the vertex. The inserted vertex becomes an isolated vertex.

Axiom 2: A transformation rule $vd^{(j,k)}_{[u,\bullet]}$ is used to transform $g^{(j,k)}$ to $g^{(j,k+1)}$ if and only if

- a vertex with vertex ID u is contained in $g^{(j,k)}$,
- no edges connected to this vertex are contained in $g^{(j,k)}$, and
- a vertex with vertex ID u is not contained in $g^{(j,k+1)}$. ■

This rule is not applicable to vertices that are not isolated. For example, $g^{(j,k)}$ in Fig. 3 (b) is transformed to $g^{(j,k+1)}$ by deleting the isolated vertex with vertex ID 3. All axioms governing the rules in Table 1 are listed in Table 2.

In summary, we introduce the following representation of a transformation sequence.

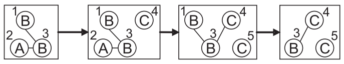
Definition 4: An intrastate sequence $s^{(j)} = \langle g^{(j,1)} g^{(j,2)} \cdots g^{(j,m_j)} \rangle$ is represented by the following sequence of TRs.

$$seq(s^{(j)}) = \langle tr^{(j,1)}_{[o,l]} tr^{(j,2)}_{[o,l]} \cdots tr^{(j,m_j-1)}_{[o,l]} \rangle.$$

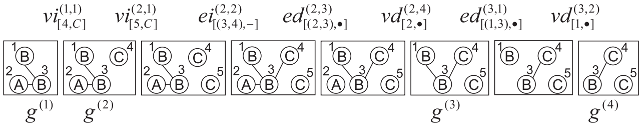
This is called an "intrastate transformation sequence". Moreover, an observed graph sequence (interstate sequence) $d = \langle g^{(1)} g^{(2)} \cdots g^{(n)} \rangle$ is represented by an "interstate transformation sequence" $seq(d) = \langle seq(s^{(1)}) seq(s^{(2)}) \cdots seq(s^{(n-1)}) \rangle$ together with its initial interstate $g^{(1)}$. ■

Table 2 Axioms governing transformation rules (TRs).

Axiom 1	Vertex Insertion $vi^{(j,k)}_{[u,l]}$	$\nexists v \in V(g^{(j,k)})\ s.t.\ id(v) = u$, and $\exists v' \in V(g^{(j,k+1)})\ s.t.\ (id(v') = u) \wedge (f(v') = l)$.
Axiom 2	Vertex Deletion $vd^{(j,k)}_{[u,\bullet]}$	$\exists v \in V(g^{(j,k)})\ s.t.\ id(v) = u$, $\nexists (v_1, v_2) \in E(g^{(j,k)})\ s.t.\ (id(v_1) = u) \vee (id(v_2) = u)$, and $\nexists v' \in V(g^{(j,k+1)})\ s.t.\ id(v') = u$.
Axiom 3	Vertex Relabeling $vr^{(j,k)}_{[u,l]}$	$\exists v \in V(g^{(j,k)})\ s.t.\ (id(v) = u) \wedge (f(v) \neq l)$, and $\exists v' \in V(g^{(j,k+1)})\ s.t.\ (id(v') = u) \wedge (f(v') = l)$.
Axiom 4	Edge Insertion $ei^{(j,k)}_{[(u_1,u_2),l]}$	$\nexists (v_1, v_2) \in E(g^{(j,k)}),\ \exists v_1 \in V(g^{(j,k)}),\ \exists v_2 \in V(g^{(j,k)})\ s.t.\ (id(v_1) = u_1) \wedge (id(v_2) = u_2)$, and $\exists (v'_1, v'_2) \in E(g^{(j,k+1)})\ s.t.\ (id(v'_1) = u_1) \wedge (id(v'_2) = u_2) \wedge (f((v'_1, v'_2)) = l)$.
Axiom 5	Edge Deletion $ed^{(j,k)}_{[(u_1,u_2),\bullet]}$	$\exists (v_1, v_2) \in E(g^{(j,k)})\ s.t.\ (id(v_1) = u_1) \wedge (id(v_2) = u_2)$, and $\nexists (v'_1, v'_2) \in E(g^{(j,k+1)}),\ \exists v'_1 \in V(g^{(j,k+1)}),\ \exists v'_2 \in V(g^{(j,k+1)})\ s.t.\ (id(v'_1) = u_1) \wedge (id(v'_2) = u_2)$.
Axiom 6	Edge Relabeling $er^{(j,k)}_{[(u_1,u_2),l]}$	$\exists (v_1, v_2) \in E(g^{(j,k)})\ s.t.\ (id(v_1) = u_1) \wedge (id(v_2) = u_2) \wedge (f((v_1, v_2)) \neq l)$, and $\exists (v'_1, v'_2) \in E(g^{(j,k+1)})\ s.t.\ (id(v'_1) = u_1) \wedge (id(v'_2) = u_2) \wedge (f((v'_1, v'_2)) = l)$.



(a) An example of a graph sequence d .


$vi^{(1,1)}_{[4,C]}$
$vi^{(2,1)}_{[5,C]}$
$ei^{(2,2)}_{[(3,4),-]}$
$ed^{(2,3)}_{[(2,3),\bullet]}$
$vd^{(2,4)}_{[2,\bullet]}$
$ed^{(3,1)}_{[(1,3),\bullet]}$
$vd^{(3,2)}_{[1,\bullet]}$
$g^{(1)}$
$g^{(2)}$
$g^{(3)}$
$g^{(4)}$

(b) Representation of the graph sequence d by using TRs.

Fig. 4 A graph sequence and its TRs.

The notation for intrastate transformation sequences is far more compact than the original graph based representation, since only differences between two successive intrastates are kept in the sequence.

Example 2: In Fig. 4 (a), a graph sequence d can be expressed as a sequence of insertions and deletions of vertices and edges as shown in Fig. 4 (b). The transformation sequence is represented as $seq(d) = \langle vi^{(1,1)}_{[4,C]} vi^{(2,1)}_{[5,C]} ei^{(2,2)}_{[(3,4),-]} ed^{(2,3)}_{[(2,3),\bullet]} vd^{(2,4)}_{[2,\bullet]} ed^{(3,1)}_{[(1,3),\bullet]} vd^{(3,2)}_{[1,\bullet]} \rangle$, where "–" denotes an edge label.

Let $tr^{(j,k)}_{[o,l]} \prec tr^{(j,k')}_{[o',l']}$ indicate that $tr^{(j,k)}_{[o,l]}$ precedes $tr^{(j,k')}_{[o',l']}$ in an intrastate transformation sequence, so that $k < k'$. Based on the axioms in Table 2 and Definition 2, we define the following admissibility constraint on a pair of TRs in a sequence.

Definition 5: Let all TRs $tr^{(j,k'')}_{[o'',l'']}$ where $k < k'' < k'$ have o'' different from o and o' for a TR pair $tr^{(j,k)}_{[o,l]} \prec tr^{(j,k')}_{[o',l']}$ in a transformation sequence. Any TR pair is admissible unless

- o and o' correspond to either a vertex or an edge that is incident with the vertex.

Otherwise, the TR pair is admissible if

- the condition for applying $tr^{(j,k')}_{[o',l']}$ to o' is satisfied by the resultant o of the preceding $tr^{(j,k)}_{[o,l]}$ under the axioms in Table 2, and
- we cannot rewrite the transformation subsequence, including the TR pair, as a shorter and equivalent transformation sequence that does not include the TR pair. ■

We limit this admissibility to the case in which objects o'' of all TRs between k and k' are different from o and o', since the TR that transforms an object o appears only once in an intrastate transformation sequence, as shown later. In addition, we request that a transformation sequence including the TR pair not be rewritable as a shorter transformation sequence that does not include the TRs, due to the minimum edit distance condition in Definition 2.

Based on the axioms in Table 2 and Definition 5, we prove the admissible pairs and inadmissible pairs of TRs that appear in the sequence as shown in Table 3. For example, the pair $vd^{(j,k)}_{[u_1,\bullet]} \prec vi^{(j,k')}_{[u_1,l']}$ marked by † is not admissible, because it can be replaced by $vr^{(j,k)}_{[u_1,l']}$, and thus Definition 2 does not hold. On the other hand, the pair $ei^{(j,k)}_{[(u_1,u_2),l]} \prec vr^{(j,k')}_{[u_1,l']}$ marked by ‡ is admissible. Applying $ei^{(j,k)}_{[(u_1,u_2),l]}$ indicates the existence of a vertex whose vertex ID is u_1 in the objective graph. This suffices for applying $vr^{(j,k')}_{[u_1,l']}$. The following theorem summarizes the admissibility of TR pairs shown in Table 3.

Theorem 1: (Admissibility Theorem) Given an intrastate transformation sequence $seq(s^{(j)}) = \langle tr^{(j,1)}_{[o,l]} tr^{(j,2)}_{[o,l]} \cdots tr^{(j,m_j-1)}_{[o,l]} \rangle$, the admissible pairs of TRs that may be applied to a vertex with vertex ID u_1 or an edge incident with the vertex with ID u_1 in the sequence is limited to one of the following 8 ordered pairs where $1 \leq k < k' \leq m_j - 1$.

$$\begin{gathered} vi^{(j,k)}_{[u_1,l]} \prec ei^{(j,k')}_{[(u_1,u_2),l']},\ vr^{(j,k)}_{[u_1,l]} \prec ei^{(j,k')}_{[(u_1,u_2),l']}, \\ vr^{(j,k)}_{[u_1,l]} \prec ed^{(j,k')}_{[(u_1,u_2),\bullet]},\ vr^{(j,k)}_{[u_1,l]} \prec er^{(j,k')}_{[(u_1,u_2),l']}, \\ ei^{(j,k)}_{[(u_1,u_2),l]} \prec vr^{(j,k')}_{[u_1,l']},\ ed^{(j,k)}_{[(u_1,u_2),\bullet]} \prec vr^{(j,k')}_{[u_1,l']}, \\ er^{(j,k)}_{[(u_1,u_2),l]} \prec vr^{(j,k')}_{[u_1,l']},\ ed^{(j,k)}_{[(u_1,u_2),\bullet]} \prec vd^{(j,k')}_{[u_1,\bullet]}. \end{gathered}$$ ■

As this theorem is a summary of Table 3, its proof is omitted. From Theorem 1, we can obtain in a straightforward manner, a sequence of TRs for interpolating between two

Table 3 Admissibility of ordered transformation rules.

$tr^{(j,k)}_{[o,l]}$	$tr^{(j,k')}_{[o',l']}$	Vertex Insertion $o' = u_1$	Vertex Deletion $o' = u_1$	Vertex Relabeling $o' = u_1$	Edge Insertion $o' = (u_1, u_2)$	Edge Deletion $o' = (u_1, u_2)$	Edge Relabeling $o' = (u_1, u_2)$
Vertex Insertion	$o = u_1$	X	X	X	O	X	X
Vertex Deletion	$o = u_1$	X†	X	X	X	X	X
Vertex Relabeling	$o = u_1$	X	X	X	O	O	O
Edge Insertion	$o = (u_1, u_2)$	X	X	O‡	X	X	X
Edge Deletion	$o = (u_1, u_2)$	X	O	O	X	X	X
Edge Relabeling	$o = (u_1, u_2)$	X	X	O	X	X	X

O and X denote being admissible and not admissible, respectively.
$tr^{(j,k)}_{[o,l]}$ precedes $tr^{(j,k')}_{[o',l']}$ in an intrastate transformation sequence.

successive graphs $g^{(j)}$ and $g^{(j+1)}$ under the definitions in Table 1 and the constraints of Theorem 1. We now show that any graph sequence is represented by our six TRs.

Theorem 2: (Representation Theorem) Any graph sequence can be represented by the six TRs in Table 1 with a given initial interstate $g^{(1)}$. ■

Proof. Let $g^{(j)}$ and $g^{(j+1)}$ ($j = 1, \cdots, n-1$) be two successive graphs in a given graph sequence $d = \langle g^{(1)}, \cdots, g^{(n)} \rangle$. Let f and f' be the labeling functions of $g^{(j)}$ and $g^{(j+1)}$, respectively. We define

$$
\begin{aligned}
V_{vi} &= \{v \mid v \in V(g^{(j+1)}),\\
&\quad id(v) \in ID_V(g^{(j+1)}) \setminus ID_V(g^{(j)})\},\\
V_{vd} &= \{v \mid v \in V(g^{(j)}),\\
&\quad id(v) \in ID_V(g^{(j)}) \setminus ID_V(g^{(j+1)})\},\\
V_{vr} &= \{v \mid v \in V(g^{(j)}), v' \in V(g^{(j+1)}),\\
&\quad id(v) = id(v'), f(v) \neq f'(v')\},\\
V &= \{v \mid v \in V(g^{(j)}), v' \in V(g^{(j+1)}),\\
&\quad id(v) = id(v'), f(v) = f'(v')\},\\
E_{ei} &= \{(v_1, v_2) \mid (v_1, v_2) \in E(g^{(j+1)}),\\
&\quad (id(v_1), id(v_2)) \in ID_E(g^{(j+1)}) \setminus ID_E(g^{(j)})\},\\
E_{ed} &= \{(v_1, v_2) \mid (v_1, v_2) \in E(g^{(j)}),\\
&\quad (id(v_1), id(v_2)) \in ID_E(g^{(j)}) \setminus ID_E(g^{(j+1)})\},\\
E_{er} &= \{(v_1, v_2) \mid (v_1, v_2) \in E(g^{(j)}),\\
&\quad (v'_1, v'_2) \in E(g^{(j+1)}),\\
&\quad (id(v_1), id(v_2)) = (id(v'_1), id(v'_2)),\\
&\quad f((v_1, v_2)) \neq f'((v'_1, v'_2))\},\\
E &= \{(v_1, v_2) \mid (v_1, v_2) \in E(g^{(j)}),\\
&\quad (v'_1, v'_2) \in E(g^{(j+1)}),\\
&\quad (id(v_1), id(v_2)) = (id(v'_1), id(v'_2)),\\
&\quad f((v_1, v_2)) = f'((v'_1, v'_2))\}.
\end{aligned}
$$

V_{vi}, V_{vd}, and V_{vr} are the sets of vertices that are inserted, deleted, and relabeled, respectively. Similarly, E_{ei}, E_{ed}, and E_{er} are the sets of edges that are inserted, deleted, and relabeled, respectively. V and E are the sets of vertices and edges to which no rules are applied. Thus, $g^{(j)}$ and $g^{(j+1)}$ are expressed as

$$g^{(j)} = (V \cup V_{vr} \cup V_{vd}, E \cup E_{er} \cup E_{ed}, L(g^{(j)}), f) \textit{ and}$$

$$g^{(j+1)} = (V \cup V_{vr} \cup V_{vi}, E \cup E_{er} \cup E_{ei}, L(g^{(j+1)}), f'),$$

respectively. First, $g^{(j)} = g^{(j,1)}$ is transformed to the graph

$$g^{(j,k_1)} = (V \cup V_{vr} \cup V_{vd} \cup V_{vi}, E \cup E_{er} \cup E_{ed}, L(g^{(j)}), f)$$

by inserting all vertices in V_{vi} into $g^{(j,1)}$ in a stepwise manner, where $k_1 = 1 + |V_{vi}|$. Next, $g^{(j,k_1)}$ is transformed to the graph

$$g^{(j,k_2)} = (V \cup V_{vr} \cup V_{vd} \cup V_{vi}, E \cup E_{er} \cup E_{ed} \cup E_{ei}, L(g^{(j)}), f)$$

by inserting all edges in E_{ei} into $g^{(j,k_1)}$ where $k_2 = k_1 + |E_{ei}|$. Subsequently, $g^{(j,k_2)}$ is transformed to the graph

$$
\begin{aligned}
&g^{(j,k_3)} = \\
&(V \cup V_{vr} \cup V_{vd} \cup V_{vi}, E \cup E_{er} \cup E_{ed} \cup E_{ei}, L(g^{(j+1)}), f')
\end{aligned}
$$

by relabeling the labels of all vertices in V_{vr} and all edges in E_{er} where $k_3 = k_2 + |V_{vr}| + |E_{er}|$. $g^{(j,k_3)}$ is further transformed to

$$g^{(j,k_4)} = (V \cup V_{vr} \cup V_{vd} \cup V_{vi}, E \cup E_{er} \cup E_{ei}, L(g^{(j+1)}), f')$$

by deleting all edges in E_{ed} from $g^{(j,k_3)}$ where $k_4 = k_3 + |E_{ed}|$. Finally, $g^{(j,k_4)}$ is transformed to

$$g^{(j,k_5)} = (V \cup V_{vr} \cup V_{vi}, E \cup E_{er} \cup E_{ei}, L(g^{(j+1)}), f')$$

by deleting all vertices in V_{vd} from $g^{(j,k_4)}$ where $k_5 = k_4 + |V_{vd}| = m_j$. The graph $g^{(j,k_5)}$ becomes isomorphic with $g^{(j+1)}$. In addition, because V_{vi}, V_{vd}, V_{vr}, and V are mutually disjoint, and E_{ei}, E_{ed}, E_{er}, and E are also mutually disjoint, the number of TRs applied to each object o in a TR $tr^{(j,k)}_{[o,l]}$ is at most 1. The order of TRs applied in the above transformation process is consistent with the admissibility of ordered TRs indicated by Theorem 1. Accordingly, any intrastate sequence from $g^{(j)}$ to $g^{(j+1)}$ can be represented by the six TRs and the initial intrastate $g^{(j)} = g^{(j,k)}$. Finally, because this statement holds for all $j = 1, \cdots, n-1$, we conclude that any graph sequence d can be represented by the six TRs and the initial interstate $g^{(1)}$. □

2.2 Compilation of Graph Sequences

In this subsection, we propose an algorithm for compiling a given observed graph sequence to a transformation sequence. An observed graph sequence (interstate sequence)

```
1)  **GraphSequenceCompiler**($d$)
2)    $seq(d) = \langle\rangle$
3)    for each $j$
4)      Define $V_{vi}$, $V_{vd}$, $V_{vr}$, $E_{ei}$, $E_{ed}$, and $E_{er}$
          as in the proof of Theorem 2.
5)      $k = 1$
6)      for each $v \in V_{vi}$
7)        $seq(d) = seq(d) \cup vi^{(j,k{+}{+})}_{[id(v),f(v)]}$
8)      for each $(v, v') \in E_{ei}$
9)        $seq(d) = seq(d) \cup ei^{(j,k{+}{+})}_{[(id(v),id(v')),f((v,v'))]}$
10)     for each $v \in V_{vr}$
11)       $seq(d) = seq(d) \cup vr^{(j,k{+}{+})}_{[id(v),f'(v)]}$
12)     for each $(v, v') \in E_{er}$
13)       $seq(d) = seq(d) \cup er^{(j,k{+}{+})}_{[(id(v),id(v')),f'((v,v'))]}$
14)     for each $(v, v') \in E_{ed}$
15)       $seq(d) = seq(d) \cup ed^{(j,k{+}{+})}_{[(id(v),id(v')),\bullet]}$
16)     for each $v \in V_{vd}$
17)       $seq(d) = seq(d) \cup vd^{(j,k{+}{+})}_{[id(v),\bullet]}$
18)   output $seq(d)$
```

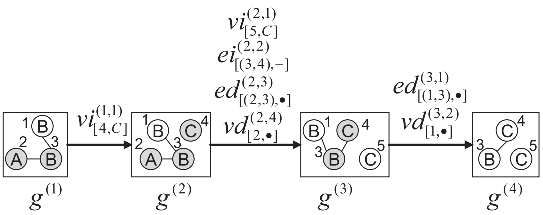
Fig. 5 Algorithm to compile a graph sequence to a transformation sequence.

$d = \langle g^{(1)} g^{(2)} \cdots g^{(n)} \rangle$ is represented by an interstate transformation sequence $seq(d) = \langle seq(s^{(1)}) seq(s^{(2)}) \cdots seq(s^{(n-1)}) \rangle$ and its initial interstate $g^{(1)}$ by Definition 4. Based on this graph sequence representation, we have developed an algorithm for compiling a graph sequence $d = \langle g^{(1)} \cdots g^{(n)} \rangle$ to a transformation sequence $seq(d)$ as shown in Fig. 5. The procedure basically follows the proof of Theorem 2 for compiling each graph sequence. $seq(d) \cup tr^{(j,k)}_{[o,l]}$ means that $tr^{(j,k)}_{[o,l]}$ is appended to the end of the transformation sequence $seq(d)$. To interpolate the differences between $g^{(j)}$ and $g^{(j+1)}$, the vertex and edge insertions are first appended to the end of the transformation sequence in Lines 9 and 11. Next, relabelings are appended in Lines 13 and 15. Finally, the vertex and edge deletions are appended in Lines 17 and 19.

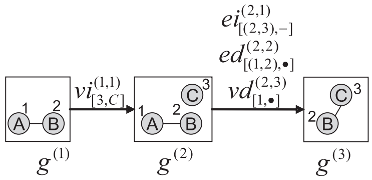
As is evident from this algorithm, the order of TRs in an intrastate transformation sequence is artificial, and does not reflect the order of graphs in a given observed graph sequence, except for the starting and ending graphs in the intrastate sequences. Therefore, we need to focus on the order of interstates only in the mining result. On the other hand, because the intrastate transformation sequence represents an admissible order of TRs for the interpolation between two successive interstates, it enables us to apply Sequential Pattern Mining to the graph sequence mining problem as shown later. In general, computing the edit distance between two graphs is NP-hard, because all combinations of vertices in the two graphs must be considered. However, in our case computing a sequence of TRs based on differences between two graphs is solvable in linear time, because all vertices have vertex IDs.

3. Principles of FTS Mining

In this section, we propose a method for mining frequent transformation subsequences (FTSs) from a given set of graph sequences. Since the FTSs to be mined are included in the given transformation sequences, the FTSs are also governed by the Admissibility Theorem. To mine FTSs from a given set of compiled graph sequences, that is, a set of transformation sequences, we define an inclusion relation between the intrastate transformation sequences.



(a) A graph sequence d.



(b) A subsequence d' of d.

Fig. 6 Inclusion relation.

Definition 6: Let

$$seq(s^{(j)}) = \langle tr^{(j,1)}_{[o,l]} \cdots tr^{(j,k)}_{[o,l]} \cdots tr^{(j,m_j-1)}_{[o,l]} \rangle, \tag{1}$$

$$seq(s'^{(h)}) = \langle tr^{(h,1)}_{[o,l]} \cdots tr^{(h,r)}_{[o,l]} \cdots tr^{(h,m_h-1)}_{[o,l]} \rangle \tag{2}$$

be two intrastate transformation sequences. $seq(s'^{(h)})$ is a subsequence of $seq(s^{(j)})$, denoted by $seq(s'^{(h)}) \subseteq_\phi seq(s^{(j)})$, if there is an injective function $\phi : ID_V(s'^{(h)}) \to ID_V(s^{(j)})$ such that $tr1 = tr2$, $\phi(o_{hr}) = o_{jk}$, and $l_{hr} = l_{jk}$ for $tr1_{[o_{hr},l_{hr}]} \in seq(s'^{(h)})$ and $tr2_{[o_{jk},l_{jk}]} \in seq(s^{(j)})$. ■

Note that for $\phi(o_{hr}) = o_{jk}$, $o_{hr} = o_{jk}$ does not always hold as shown in Example 3. Moreover, the order among intrastates is not preserved in either $seq(s^{(j)})$ or $seq(s'^{(h)})$, because intrastate transformation sequences are artificially generated by the interpolation. In addition, we define an inclusion relation between interstate transformation sequences.

Definition 7: Let $seq(d)$ and $seq(d')$ be the transformation sequences

$$seq(d) = \langle seq(s^{(1)}) \cdots seq(s^{(j)}) \cdots seq(s^{(n-1)}) \rangle,$$
$$seq(d') = \langle seq(s'^{(1)}) \cdots seq(s'^{(h)}) \cdots seq(s'^{(n'-1)}) \rangle,$$

where the intrastate sequences $seq(s^{(j)})$ and $seq(s'^{(h)})$ are given in Eqs. (1) and (2), respectively. $seq(d')$ is a subsequence of $seq(d)$, denoted by $seq(d') \sqsubseteq seq(d)$, if there exist integers $1 \le j_1 < \cdots < j_{n'-1} \le n - 1$ such that $seq(s'^{(h)}) \subseteq_\phi seq(s^{(j_h)})$ for $h = 1, \cdots, n' - 1$. ■

Example 3: Given the graph sequence d in Fig. 6 (a) represented by the transformation sequence $seq(d) = \langle \underline{vi^{(1,1)}_{[4,C]}} vi^{(2,1)}_{[5,C]} \underline{ei^{(2,2)}_{[(3,4),-]} ed^{(2,3)}_{[(2,3),\bullet]} vd^{(2,4)}_{[2,\bullet]}} ed^{(3,1)}_{[(1,3),\bullet]} vd^{(3,2)}_{[1,\bullet]} \rangle$, the transformation sequence $seq(d') = \langle vi^{(1,1)}_{[3,C]} ei^{(2,1)}_{[(2,3),-]} ed^{(2,2)}_{[(1,2),\bullet]} vd^{(2,3)}_{[1,\bullet]} \rangle$ of the graph sequence d' in Fig. 6 (b) is a subsequence of

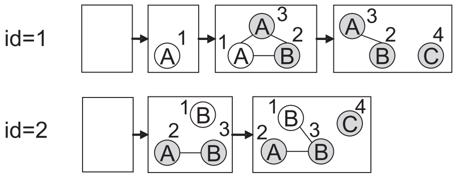


Fig. 7 DB containing graph sequences.

$seq(d)$, and the transformation rules in $seq(d')$ match the underlined rules in $seq(d)$ via $\phi(i) = i + 1$ for $i \in ID_V(d') = \{1, 2, 3\}$.

While the transformation sequence data indicates the progressive change in a graph and as such can be graphically represented as long as the data is valid, the change represented by its subsequences is discontinuous and often difficult to present graphically.

Given a set of data $DB = \{(id, d) \mid d = \langle g^{(1)} g^{(2)} \cdots g^{(n)} \rangle\}$, where id is the ID of d, a support value $\sigma(seq(d'))$ of a transformation subsequence $seq(d')$ is defined by

$$\sigma(seq(d')) = \frac{|\{id \mid (id, d) \in DB, seq(d') \sqsubseteq seq(d)\}|}{|DB|}.$$

We call a transformation subsequence whose support value is greater than or equal to a minimum support threshold σ' a "frequent transformation subsequence (FTS)". The anti-monotonicity of this support value holds; that is, if $seq(d'_1) \sqsubset seq(d'_2)$ then $\sigma(seq(d'_1)) \geq \sigma(seq(d'_2))$. Using these definitions, we now state our mining problem as follows.

Problem 1: Given a dataset $DB = \{(id, d) \mid d = \langle g^{(1)} g^{(2)} \cdots g^{(n)} \rangle\}$ and a minimum support threshold σ' as input, enumerate all frequent transformation subsequences (FTSs).

Example 4: Figure 7 shows a graph sequence database DB containing two graph sequences, which are, respectively, compiled into the following transformation sequences.

$$(1, \langle vi^{(1,1)}_{[1,A]} \underline{vi^{(2,1)}_{[2,B]} vi^{(2,2)}_{[3,A]} ei^{(2,3)}_{[(1,2),-]}} ei^{(2,4)}_{[(1,3),-]} ei^{(2,5)}_{[(2,3),-]}$$
$$\underline{vi^{(3,1)}_{[4,C]}} ed^{(3,2)}_{[(1,3),\bullet]} ed^{(3,3)}_{[(1,2),\bullet]} vd^{(3,4)}_{[1,\bullet]} \rangle),$$
$$(2, \langle vi^{(1,1)}_{[1,B]} \underline{vi^{(1,2)}_{[2,A]} vi^{(1,3)}_{[3,B]} ei^{(1,4)}_{[(2,3),-]} vi^{(2,1)}_{[4,C]}} ei^{(2,2)}_{[(1,3),-]} \rangle).$$

One of the FTSs mined from DB under $\sigma' = 2$ is

$$\langle vi^{(1,1)}_{[1,A]} vi^{(2,1)}_{[2,B]} vi^{(2,2)}_{[3,A]} vi^{(3,1)}_{[4,C]} \rangle,$$

where the TRs in the FTS match the underlined TRs in each of the transformation sequences.

To enumerate a complete set of FTSs in DB, we consider an algorithm that recursively appends a TR to the end of the current FTS using the Pattern Growth principle. For example, let $seq(d) = \langle vi^{(1,1)}_{[1,A]} \rangle$ be a current FTS. If the rule to be appended is a vertex insertion, the transformation subsequences generated are

```
**FTSMiner**($seq(d), seq(DB), \sigma', F, j, k$)
  if $seq(d) = \langle\rangle${
    $AppendRule(seq(d), seq(DB), \sigma', F, 1, 1)$
  }else{
    $AppendRule(seq(d), seq(DB), \sigma', F, j, k+1)$
    $AppendRule(seq(d), seq(DB), \sigma', F, j+1, 1)$ }
  }

**AppendRule**($seq(d), seq(DB), \sigma', F, j, k$)
  for every $tr^{(j,k)}_{[u,l]} \in C(seq(d), seq(DB))${
    if $\sigma(seq(d) \cup tr^{(j,k)}_{[u,l]}) \geq \sigma'${
      $F = F \cup \{seq(d) \cup tr^{(j,k)}_{[u,l]}\}$
      FTSMiner($seq(d) \cup tr^{(j,k)}_{[u,l]}, seq(DB), \sigma', F, j, k$)
    }
  }
```

Fig. 8 Algorithm to enumerate FTSs.

$$\langle vi^{(1,1)}_{[1,A]} vi^{(1,2)}_{[u,l]} \rangle = \langle vi^{(j,k)}_{[1,A]} vi^{(j,k+1)}_{[u,l]} \rangle \; and$$
$$\langle vi^{(1,1)}_{[1,A]} vi^{(2,1)}_{[u,l]} \rangle = \langle vi^{(j,k)}_{[1,A]} vi^{(j+1,1)}_{[u,l]} \rangle$$

where $\exists(id, d) \in DB$ and $\exists(j, k)$; $\langle vi^{(j,k)}_{[1,A]} vi^{(j,k+1)}_{[u,l]} \rangle \sqsubseteq seq(d)$ or $\langle vi^{(j,k)}_{[1,A]} vi^{(j+1,1)}_{[u,l]} \rangle \sqsubseteq seq(d)$. The former subsequence shows that a vertex with vertex ID u is inserted at the same time that the vertex with vertex ID 1 is inserted. The latter represents a vertex with vertex ID u being inserted after the vertex with vertex ID 1 has been inserted. In each case, any inadmissible subsequences such as $\langle vi^{(1,1)}_{[1,A]} vi^{(1,2)}_{[1,l]} \rangle$ are never generated, because such subsequences are not included in any $seq(d)$ with $(id, d) \in DB$ according to the Admissibility Theorem. The following lemma provides the enumeration of the complete set of transformation subsequences.

Lemma 1: Let $seq(DB) = \{(id, seq(d)) \mid (id, d) \in DB\}$. Given a transformation subsequence $seq(d) = \langle tr^{(1,1)}_{[o,l]} \cdots tr^{(j,k)}_{[o,l]} \rangle$, let the candidate set of TRs to be appended to $seq(d)$ be

$$\begin{aligned} & C(seq(d), seq(DB)) \\ & = \{tr^{(j,k+1)}_{[o,l]} \; and \; tr^{(j+1,1)}_{[o,l]} \mid \\ & \qquad \exists(id, seq(d_i)) \in seq(DB); \\ & \qquad\quad seq(d) \cup tr^{(j,k+1)}_{[o,l]} \sqsubseteq seq(d_i) \; or \\ & \qquad\quad seq(d) \cup tr^{(j+1,1)}_{[o,l]} \sqsubseteq seq(d_i)\} \end{aligned}$$

where $seq(d) \cup tr^{(j,k)}_{[o,l]}$ denotes appending a transformation rule $tr^{(j,k)}_{[o,l]}$ to the end of $seq(d)$. By recursively appending $tr^{(j,k+1)}_{[o,l]}$ or $tr^{(j+1,1)}_{[o,l]}$ in $C(seq(d), seq(DB))$ to $seq(d)$, the complete set of candidate FTSs is generated. If $seq(d) = \langle\rangle$, all candidate subsequences $\langle tr^{(1,1)}_{[o,l]} \rangle \in C(seq(d), seq(DB))$ suffice for completeness where

$$\begin{aligned} & C(seq(d), seq(DB)) \\ & = \{tr^{(1,1)}_{[o,l]} \mid \exists(id, seq(d_i)) \in seq(DB); \\ & \qquad\qquad \langle tr^{(1,1)}_{[o,l]} \rangle \sqsubseteq seq(d_i)\}. \end{aligned}$$

■

From Lemma 1, we design an efficient depth-first algorithm as shown in Fig. 8. The argument F accumulates the FTSs found. If $seq(d) = \langle\rangle$, all candidate subsequences $seq(d) = \langle tr^{(1,1)}_{[o,l]}\rangle$ are generated. Otherwise, the algorithm recursively mines FTSs by appending a TR to the end of the current FTS in Line 12. The appended rule is either in the intrastate sequence where the last TR in $seq(d)$ exists (Line 5), or is in the next intrastate sequence (Line 6). Because this algorithm is based on the Pattern Growth principle, transformation subsequences containing inadmissible rules are never generated according to the aforementioned data compilation based on the Admissible Theorem.

We have implemented this algorithm using PrefixSpan [17], which is a representative method for mining frequent subsequences from a set of itemset sequences. Since it enumerates FTSs by projecting the given sequences into shorter sequences, it can efficiently mine long FTSs.

4. Principles of Relevant FTS Mining

The algorithm proposed in Sect. 3 enumerates a complete set of FTSs. However, for practical reasons, we often focus on graph subsequences consisting of mutually relevant vertices and edges only. For example, given the graph subsequence depicted in Fig. 9, the vertex with vertex ID 1 is considered to be irrelevant, because it is not connected to any other vertex in any step. On the other hand, although the vertices with vertex IDs 2 and 4 are not connected directly in any step, they are connected to the vertex with vertex ID 3 in the first and fourth steps, respectively. In this case, we consider the vertices with vertex IDs 2 and 4 to be mutually relevant via the vertex with vertex ID 3. In the case of Fig. 9, many analysts consider it appropriate to include the vertices with vertex IDs 2, 3, and 4 but exclude the vertex with vertex ID 1 in the graph sequence analysis. Based on the above observation and the potential connectivity in a given graph sequence, we define the relevancy among vertex IDs of vertices and edges as follows.

Definition 8: Vertex IDs in a graph sequence $d = \langle g^{(1)}g^{(2)}\cdots g^{(n)}\rangle$ are relevant to one another, and d is called a "relevant graph sequence", if the union graph $g_u(d)$ of d is connected. Here, we define the union graph of d to be $g_u(d) = (V, E)$ where

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

A connected graph in this definition is a graph that has a path between any two vertices, and the path is a set of edges making reachable from a vertex v to another v'. The union graph of a graph sequence d also defines the relevance of the corresponding transformation sequence $seq(d)$.

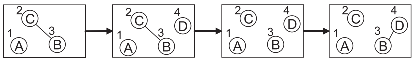


Fig. 9 An irrelevant graph subsequence.

Definition 9: The union graph $g_u(seq(d)) = (V, E)$ of a transformation sequence $seq(d)$ is similarly defined as

$$\begin{aligned} V = {} & \{u \mid tr^{(j,k)}_{[u,l]} \in seq(d), tr \in \{vi, vd, vr\}\}\\ & \cup \{u, u' \mid tr^{(j,k)}_{[(u,u'),l]} \in seq(d), tr \in \{ei, ed, er\}\},\\ E = {} & \{(u, u') \mid tr^{(j,k)}_{[(u,u'),l]} \in seq(d), tr \in \{ei, ed, er\}\}. \qquad \blacksquare \end{aligned}$$

Using the definition of the union graph, we define the following problem for mining relevant FTSs.

Problem 2: Given a dataset $DB = \{(id, d) \mid d = \langle g^{(1)}g^{(2)}\cdots g^{(n)}\rangle\}$ and a minimum support threshold σ' as input, enumerate all FTSs whose union graphs are connected.

To enumerate all relevant FTSs efficiently, the union graphs of all graph sequences in DB are generated. All frequent "connected" subgraphs among these union graphs are then enumerated using the conventional Graph Mining algorithm. Every time the algorithm outputs a frequent connected subgraph, FTSMiner, as given in Fig. 8, is called with its input set of transformation sequences generated by the projection in the following definition.

Definition 10: Given a transformation sequence $(id, seq(d)) \in seq(DB)$ and a connected graph g, we define a function "proj" that projects $seq(d)$ onto its maximum subsequences whose union graphs are identical to g as follows.

$$\begin{aligned} & proj((id, seq(d)), g) = \{(id, seq(d')) \mid \\ & \quad seq(d') \sqsubseteq seq(d)\ s.t.\ g_u(seq(d')) = g \wedge \\ & \quad \nexists seq(d'')\ s.t.\ (seq(d') \sqsubset seq(d'') \sqsubseteq seq(d) \wedge \\ & \qquad g_u(seq(d'')) = g)\}. \qquad \blacksquare \end{aligned}$$

In this definition, the first half $proj((id, seq(d)), g) = \{(id, seq(d')) \mid seq(d') \sqsubseteq seq(d)\ s.t.\ g_u(seq(d')) = g\}$ represents a transformation sequence $seq(d)$ being projected onto its subsequences $seq(d')$ whose union graph is g. The second half of the definition, $\nexists seq(d'')\ s.t.\ (seq(d') \sqsubset seq(d'') \sqsubseteq seq(d) \wedge g_u(seq(d'')) = g)$, guarantees that the projected transformation sequences $seq(d')$ are maximal.

Because each union graph of an FTS is also frequent in the union graphs of all $seq(d)\ s.t.\ (id, seq(d)) \in seq(DB)$, we can enumerate all candidate relevant FTSs from the projected transformation sequences if all frequent connected subgraphs of the union graphs of all $seq(d) \in seq(DB)$ are given.

Example 5: Given the graph sequence d shown in Fig. 10 (a), $seq(d)$ is represented as $seq(d) = \langle \underline{vi^{(1,1)}_{[3,C]}}\,\underline{ei^{(1,2)}_{[(1,3),-]}ei^{(1,3)}_{[(2,3),-]}}\,vi^{(2,1)}_{[4,A]}ei^{(2,2)}_{[(1,4),-]}\underline{ed^{(2,3)}_{[(1,3),\bullet]}}\rangle$, and its union graph $g_u(d)$ is depicted in Fig. 10 (b). Given a graph g which is a subgraph of $g_u(d)$, shown in Fig. 10 (d), an example transformation sequence $seq(d')$ in $proj((id, seq(d)), g)$ is $\langle vi^{(1,1)}_{[3,C]}ei^{(1,2)}_{[(1,3),-]}ei^{(1,3)}_{[(2,3),-]}ed^{(2,1)}_{[(1,3),\bullet]}\rangle$ as depicted in Fig. 10 (c).

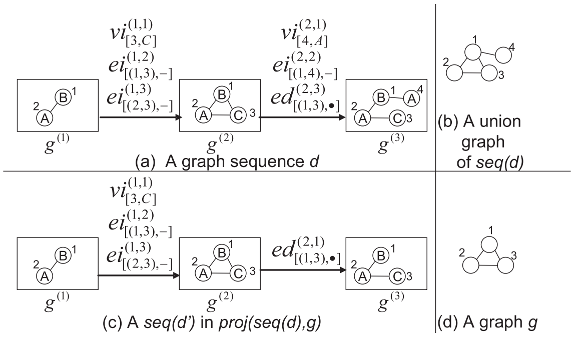


Fig. 10 Example of a projection.

```
1) RelevantFTSMiner($seq(DB), \sigma', F$)
2)    $G_u = \{g_u(d) \mid (id, seq(d)) \in seq(DB)\}$
3)    for $g$ =FrequentSubgraphMiner($G_u, \sigma'$);
         until $g \neq null${
4)       $proj(DB) = \bigcup_{(id,d)\in DB} proj((id, seq(d)), g)$
5)       $F = F \cup$ FTSMiner($\langle\rangle, proj(DB), \sigma', F, 1, 1$)
6)    }
```

Fig. 11 Algorithm to mine relevant FTSs.

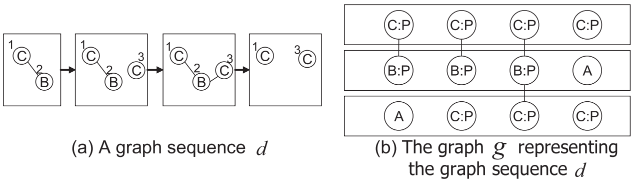
Note that this subsequence matches the underlined rules in $seq(d)$.

Figure 11 shows an algorithm that enumerates all relevant FTSs from DB. First, a set G_u of union graphs of transformation sequences $seq(DB)$ is generated in Line 2. Assuming that the function "FrequentSubgraphMiner" in Line 3 repeatedly and exhaustively outputs the frequent connected subgraphs g in G_u one at a time, FTSMiner (Fig. 5) is called in Line 5 with the transformation sequences projected in Line 4. These processes are continued until the frequent connected subgraph g is exhausted in FrequentSubgraphMiner.

For FrequentSubgraphMiner, we used AcGM [9] which is a conventional Graph Mining method. AcGM enumerates all embeddings in each union graph that are isomorphic with a frequent subgraph g, and this feature of AcGM enables efficient projection of g to all of its isomorphic transformation subsequences $proj((id, seq(d)), g)$.

5. Experiment

Our proposed method, called "GTRACE (Graph TRAnsformation sequenCE mining)", was implemented in C++. An HP xw4400 with Intel Core 2 6700 2.66 GHz and 2 GB of main memory, running Windows XP, was used for the experiments. The performance of GTRACE was evaluated for both artificial and real-world graph sequence data. Owing to the lack of any other established approach comparable with our frequent graph subsequence mining, we set up a mining task, which is functionally similar within the framework of conventional Graph Mining, for comparison. Each graph sequence $(id, d = \langle g^{(1)} \cdots g^{(n)} \rangle) \in DB$ is converted into a graph g as shown in the example in Fig. 12. Each vertex ID u, if it appears as a vertex with label l in $g^{(j)}$, is represented by a vertex with label $l{:}P$ (Presence) in the j-th column of g (Fig. 12 (b)). Otherwise u is represented by a vertex with label A (Absence). If there is an edge between two vertices in $g^{(j)}$, an edge is placed between the corresponding vertices in the j-th column of g. Finally, all vertices corresponding to an identical vertex ID u are mutually connected by edges to form a clique. This ensures that the identity of vertices in the graph can be represented. Each clique is indicated by a rectangle in Fig. 12 (b). Graph Mining on this data extracts information of the presence/absence of each vertex with their associated edges over a sequence of steps. We applied Gaston [16] to this task, since it is known to be the fastest Graph Mining algorithm.



(a) A graph sequence d

(b) The graph g representing the graph sequence d

Fig. 12 Conversion from a graph sequence to a graph.

Table 4 Parameters of test data.

Definition	Default values
probability of vertex and edge insertions in transformation sequences	$p = 80\%$
average number of vertex IDs in transformation sequences	$\lvert V_{avg} \rvert = 7$
probability of vertex and edge insertions in embedded FTSs	$p' = 50\%$
average number of vertex IDs in embedded FTSs	$\lvert V'_{avg} \rvert = 5$
number of vertex labels	$\lvert L_v \rvert = 10$
number of edge labels	$\lvert L_e \rvert = 1$
number of embedded FTSs	$N = 10$
number of transformation sequences	$\lvert DB \rvert = 1{,}000$
minimum support threshold	$\sigma' = 30\%$

5.1 Artificial Datasets

We compared the performance of GTRACE with Gaston using artificial datasets generated based on the parameters listed in Table 4. First, starting from $g^{(1)}$ with $V(g^{(1)}) = \emptyset$, we grew each transformation sequence up to having an average of $|V_{avg}|$ vertex IDs by inserting vertices and edges with probability p at each step. Accordingly, if p is small or $|V_{avg}|$ is large, the transformation sequence generated is long. This process is continued by increasing the numbers of vertices and edges until the sequence becomes relevant, while maintaining the sparsity of each interstate graph. Typically, the average probability of the existence of an edge between two vertices in an interstate was 13% using the default parameter values. This value remained low for all the test data. Similarly, we generated N relevant FTSs with an average of $|V'_{avg}|$ vertex IDs using probability p'. We then generated DB, where each transformation sequence was overlaid by each relevant FTS with probability $1/N$. The overlay was conducted in the graph sequence domain to ensure a topo-

Table 5 Results for $|V_{avg}|$, p, $|L_v|$, and σ'.

	$\lvert V_{avg}\rvert$	6	7	10	35
	avg. length of $seq(d)$	12.1	15.7	22.2	82.1
GTRACE	comp. time	0.27	0.42	0.70	224
	# of FTSs	15	50	84	2622
	comp. time/FTS	0.018	0.0085	0.0084	0.086
Gaston	comp. time	8.05	416.5	–	–
	# of FSGs	600	23594	–	–
	comp. time/FSG	0.013	0.018	–	–
	p	95%	80%	70%	10%
	avg. length of $seq(d)$	14.3	15.7	16.6	30.2
GTRACE	comp. time	0.38	0.42	0.38	0.59
	# of FTSs	44	50	54	133
	comp. time/FTS	0.0085	0.0085	0.0069	0.0045
Gaston	comp. time	119.0	416.5	–	–
	# of FSGs	7289	23594	–	–
	comp. time/FSG	0.016	0.018	–	–
	$\lvert L_v\rvert$	1	9	10	11
	avg. length of $seq(d)$	15.7	15.7	15.7	15.7
GTRACE	comp. time	0.56	0.38	0.42	0.39
	# of FTSs	189	47	50	53
	comp. time/FTS	0.0030	0.0080	0.0085	0.0074
Gaston	comp. time	–	–	416.5	250.2
	# of FSGs	–	–	23594	13737
	comp. time/FSG	–	–	0.018	0.018
	σ'	0.3%	20%	30%	35%
	avg. length of $seq(d)$	15.7	15.7	15.7	15.7
GTRACE	comp. time	14.7	0.89	0.42	0.13
	# of FTSs	160658	3841	50	36
	comp. time/FTS	9.1E-5	2.3E-4	0.0085	3.2E-3
Gaston	comp. time	–	–	416.5	124.89
	# of FSGs	–	–	23594	5292
	comp. time/FSG	–	–	0.018	2.3E-2

# of FTSs: the number of mined FTSs,
# of FSGs: the number of mined Frequent SubGraphs
comp. time/FTS [sec]: comp. time per FTS,
comp. time/FSG [sec]: comp. time per Frequent SubGraph

logically correct overlay. Each relevant transformation sequence contained $|L_v|$ vertex labels and $|L_e|$ edge labels.

Table 5 shows the computation times [sec], the numbers of derived FTSs (or FSGs: Frequent SubGraphs), and the average computation times [sec] to derive an FTS (or FSG) for various values of $|V_{avg}|$, p, $|L_v|$, and σ', with the other parameters remaining fixed at their default values. Though we designed the mining task for Gaston to be as similar as possible to the task for GTRACE, the numbers of FTSs in the respective solutions are very different. Accordingly, we also use the computation time per FTS for comparison. Results depicted as '–' in the table were not obtainable due to either intractable computation time (more than one hour) or memory overflow. The first and second parts of the table indicate that the computation times of both GTRACE and Gaston are exponential in the average length of $seq(d)$ provided by the settings of $|V_{avg}|$ and p. We consider the main reason that the computation time increases with the average length to be the increase in the numbers of FTSs in both cases, because the computation times per FTS do not vary significantly. However, the increased efficiency of GTRACE over Gaston is confirmed in terms of both the computation time per FTS and the number of focused FTSs.

The third part of Table 5 shows the effect of the number of labels on the efficiency. When $|L_v|$ is small, many subgraphs included in the graph g as depicted in Fig. 12 (b) are isomorphic with each other, and thus the computation time for Gaston is enormous. In contrast, the computation time for GTRACE remains small since $|L_v|$ does not affect the length of $seq(d)$. The fourth part of Table 5 shows that GTRACE is tractable even for a low minimum support threshold. This is because it involves mining only small union subgraphs to ensure the relevance of FTSs and the Pattern-Growth based Sequential Pattern Mining for the FTS mining, whereas Gaston's application requires mining large graphs, as depicted in Fig. 12 (b). The good scalability of GTRACE is indicated in every part of Table 5 since the computation time per FTS remains almost the same and is even significantly reduced in some instances.

5.2 Real-World Datasets

To assess the practicality of GTRACE, we applied it to two real-world datasets. The first is the Enron Email Dataset [4], while the other contains phone call histories from the Reality Mining Project at MIT [15]. In both datasets, we assigned a vertex ID to each person participating in the com-

Table 6 Results for Enron dataset.

	# of persons $\lvert V\rvert$	20	40	60	80	90
	avg. length of *seq*(*d*)	9.7	24.9	35.5	65.5	84.3
GTRACE	comp. time	0.031	0.093	0.13	1.20	36.1
(5 days)	# of FTSs	3	10	18	109	420
	comp. time/FTS	0.010	0.0093	0.007	0.011	0.086
Gaston	comp. time	3.2	6.6	10.3	48.8	–
(5 days)	# of FSGs	84	310	481	1463	–
	comp. time/FSG	0.038	0.021	0.022	0.033	–
	# of persons $\lvert V\rvert$	20	40	60	80	90
	avg. length of *seq*(*d*)	11.8	29.4	42.0	77.3	100.1
GTRACE	GT6 comp. time	0.031	0.093	0.14	1.5	303.5
(6 days)	# of FTSs	6	27	48	278	1734
	comp. time/FTS	0.0052	0.0034	0.0029	0.0053	0.18
Gaston	comp. time	59.9	–	–	–	–
(6 days)	# of FSGs	500	–	–	–	–
	comp. time/FSG	0.12	–	–	–	–
	min. sup. σ'	15%	16%	18%	20%	30%
GTRACE	comp. time	6359	4167	1504	668	7.7
(5 days)	# of FTSs	42961	30250	16019	7483	498
	comp. time/FTS	0.15	0.14	0.094	0.089	0.015

Default: minimum support $\sigma' = 50\%$, the number of vertex labels $|L_v| = 3$,
the number of edges labels $|L_e| = 1$, the number of persons $|V| = 50$

Table 7 Results for MIT dataset.

	# of persons $\lvert V\rvert$	5	7	10	15	20
	avg. length of *seq*(*d*)	6.5	11.3	19.5	40.5	62.2
GTRACE	comp. time	0.016	0.031	0.047	9.02	1477.5
(5 days)	# of FTSs	5	18	100	1479	11725
	comp. time/FTS	0.0032	0.0017	0.00047	0.0061	0.13
Gaston	comp. time	0.14	32.03	–	–	–
(5 days)	# of FSGs	90	5596	–	–	–
	comp. time/FSG	0.0016	0.0057	–	–	–
	# of persons $\lvert V\rvert$	5	7	10	15	20
	avg. length of *seq*(*d*)	7.5	13.1	22.6	47.7	73.0
GTRACE	comp. time	0.016	0.032	0.109	104.0	–
(6 days)	# of FTSs	9	23	195	4196	–
	comp. time/FTS	0.0018	0.0014	0.00056	0.025	–
Gaston	comp. time	2.92	–	–	–	–
(6 days)	# of FSGs	435	–	–	–	–
	comp. time/FSG	0.0067	–	–	–	–
	min. sup. σ'	5%	10%	15%	20%	40%
GTRACE	comp. time	63.25	8.08	2.41	0.89	0.093
(5 days)	# of FTSs	724094	47863	11201	3841	263
	comp. time/FTS	8.7E-5	1.7E-4	2.1E-4	2.3E-4	3.5E-4

Default: minimum support $\sigma' = 50\%$, the number of vertex labels $|L_v| = 3$,
the number of edge labels $|L_e| = 1$, the number of persons $|V| = 10$

munication and an edge to a pair if they communicated via email or phone on a given day, and obtained a daily graph $g^{(j)}$. We labeled people according to whether their daily vertex degrees were high, moderate, or low. A person with a high degree label was considered to be a hub in the organization. We obtained a set of weekly graph sequence data for the *DB*. The total number of weeks, that is, the number of sequences, was 200 in the Enron Email Dataset and 40 in the MIT Dataset. We randomly sampled $|V|(= 1 \sim 90)$ people in the Enron Email Dataset and $|V|(= 1 \sim 20)$ people in the MIT Dataset to form each *DB*.

Tables 6 and 7 show the computation times, numbers of enumerated FTSs (or FSGs), and computation times per FTS (or FSG) for various numbers of vertex IDs (persons) $|V|$ and minimum support value σ' for each dataset. The other parameters were set to the values indicated at the bottom of each table. GTRACE (5 days) and Gaston (5 days) indicate that each sequence d in *DB* consists of five graph steps $g^{(j)}$ from Monday to Friday. Similarly, GTRACE (6 days) and Gaston (6 days) indicate six graph steps $g^{(j)}$ from Monday to Saturday. The average length (avg. length) of *seq*(*d*) is the average number of TRs in the observed transformation sequences. In cases where the required computation time was more than one hour or memory overflowed, results are denoted as '–'.

The upper parts of these tables show the superior scalability of GTRACE as seen in the total computation time and computation time per FTS. In contrast, Gaston's application

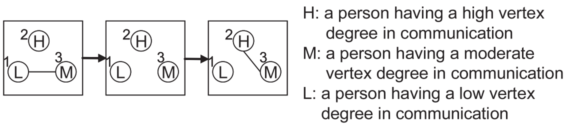

2
H
1
L
3
M
H: a person having a high vertex degree in communication
M: a person having a moderate vertex degree in communication
L: a person having a low vertex degree in communication

Fig. 13 A FTS mined from the MIT dataset.

is not tractable, since according to the data, many people only communicate with one another on a few days. This produces many vertex IDs labeled *A* (Absence) in each relevant graph sequence, and the cliques resulting from these absent vertex IDs produce a vast number of spurious frequent subgraphs. The computation times required in the MIT Dataset are far larger in spite of the smaller number of people. This is because the daily graphs in the MIT Dataset are denser than those for the Enron Email Dataset, since the MIT Dataset is based on phone calls within a tightly coupled community. The lower parts of the tables show the practical scalability of GTRACE with regard to the minimum support threshold.

Figure 13 shows a very simple but informative example mined from the MIT Dataset. Person 3 has communication with hub person 2 a few days after having communication with person 1, who has a low vertex degree. This implies the possibility that person 3, who is not a hub, is an important connection between a hub person and an uncommunicative person.

6. Discussion

Our compilation algorithm in Fig. 5 is linear in both the length of the graph sequence $|d|$ and the size of each graph $|g^{(j)}|$, since the sequence is read only once. The algorithm for enumerating FTSs in Fig. 8 is basically a pattern growth algorithm. Its complexity is identical to that of conventional Graph Mining, because the isomorphism matching problem is solved to check the inclusion relation between two transformation sequences as defined in Sect. 3. The algorithm for mining relevant FTSs in Fig. 11 uses conventional Graph Mining, and hence its complexity is also exponential in $|g^{(j)}|$. The complexity of all these tasks is linear for the number of graph sequences in DB. In total, the complexity of GTRACE is exponential in the average size of $g^{(j)}$, and linear in $|DB|$. This is reflected by our experimental results. The superior computational efficiency of GTRACE over Gaston stems from the characteristics of GTRACE, which are adapted to the graph sequence mining task, and does not indicate the inefficiency of Gaston for ordinary frequent graph mining tasks.

Recently, Borgwardt et al. proposed a method for mining frequent patterns from a set of graph sequences where only insertions and deletions of edges are allowed, and not insertion or deletion of vertices or relabeling of either vertices or edges [3]. In this approach, the existence or non-existence of an edge in each time step is represented by 1 and 0, respectively, and the edge is then labeled by a binary string consisting of these 1's and 0's over all the time steps. As this preprocessing for each graph sequence converts the data into ordinary labeled graphs, the problem of mining the frequent patterns in the graph sequences can be handled by conventional Graph Mining techniques. However, this method cannot be applied to graph sequences where insertions and deletions of vertices and relabeling are included, as in the example of human communities and gene networks mentioned in Sect. 1.

The study of Temporal Logic is another topic in which describing time sequence events is relevant [20]. A recent data mining study used this framework to analyze time sequence data [6]. Temporal logic describes the relations between times and time intervals when events hold. Our TRs can be seen as a set of simple temporal operators. The introduction of advanced temporal logic may enhance the expressiveness of graph subsequences. As mentioned in Sect. 2, another possibility for increasing expressiveness is the use of a Graph Grammar [12]. However, in both cases, computational tractability is a big issue.

In [1], [19], Srikant and Agrawal first defined the problem of finding frequent sequential patterns from a set of sequences of itemsets and proposed a method called GPS for finding the frequent sequential patterns. In the frequent sequential pattern mining problem defined in [19], maximum and/or minimum time gaps between adjacent itemsets of sequential patterns can be specified. Because PrefixSpan is an extended version of GPS to solve the same problem, PrefixSpan can be extended to mine frequent sequential patterns with maximum and/or minimum time gaps between adjacent itemsets. Therefore, GTRACE can also be extended to mine rFTSs where maximum and/or minimum time gaps exist between intrastate transformation sequences.

7. Conclusion

In this paper, we introduced transformation sequences to represent given graph sequences based on

- six transformation rules representing the differences between two successive graphs under the assumption of gradual changes in graphs,
- admissibility conditions for a pair of intrastates, and
- the Representation Theorem.

In addition, we proposed a novel framework for mining a complete set of relevant Frequent Transformation Subsequences (rFTSs) from given graph sequences. The method is based on

- the anti-monotonicity of support values
- relevancies among vertices in each graph sequence, and
- PrefixSpan based Pattern-Growth principle.

Moreover, we developed a graph sequence mining program GTRACE, and confirmed its efficiency and practicality through computational experiments using both artificial and real-world datasets.

References

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," Proc. International Conference on Data Engineering, pp.3–14, 1995.

[2] A.L. Barabási and R. Albert, "Emergence of scaling in random networks," Science, vol.286, pp.509–512, 1999.

[3] K.M. Borgwardt, H. Kriegel, and P. Wackersreuther, "Pattern mining in frequent dynamic subgraphs," Proc. International Conference on Data Mining, pp.818–822, 2006.

[4] Enron Email Dataset, http://www.cs.cmu.edu/~enron/

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

[6] T.B. Ho, C.H. Nguyen, S. Kawasaki, S.Q. Le, and K. Takabayashi, "Exploiting temporal relations in mining hepatitis data," New Generation Computing, vol.25, no.3, pp.247–262, 2007.

[7] 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.

[8] A. Inokuchi and T. Washio, "A fast method to mine frequent subsequences from graph sequence data," Proc. International Conference on Data Mining, pp.303–312, 2008.

[9] A. Inokuchi, T. Washio, Y. Nishimura, and H. Motoda, "A fast algorithm for mining frequent connected subgraphs," IBM Research Report, RT0448, 2002.

[10] 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.

[11] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," Proc. International Conference on Data Mining, pp.313–320, 2001.

[12] E. Jeltsch and H. Kreowski, "Grammatical inference based on hyperedge replacement. graph-Grammars," Lect. Notes Comput. Sci., vol.532, pp.461–474, 1990.

[13] I. Jonyer, L.B. Holder, and D.J. Cook, "MDL-based context-free graph Grammar induction and applications," International Journal on Artificial Intelligence Tools, vol.13, no.1, pp.65–79, 2004.

[14] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering frequent episodes in sequences," Proc. International Conference on Knowledge Discovery and Data Mining, pp.210–215, 1995.

[15] MIT Media Lab: Reality Mining, http://reality.media.mit.edu/

[16] 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.

[17] 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 Eng., pp.2–6, 2001.

[18] 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.

[19] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," Proc. International Conference on Extending Database Technology, pp.3–17, 1996.

[20] Y. Venema, Temporal Logic, Goble, Lou, ed., The Blackwell Guide to Philosophical Logic, Blackwell, 2001.

[21] T. Washio and H. Motoda, "State of the art of graph-based data mining," SIGKDD Explorations, vol.5, no.1, pp.59–68, 2003.

[22] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," Proc. International Conference on Data Mining, pp.721–724, 2002.

Akihiro Inokuchi received his Ph.D. degree in Communication Engineering from Osaka University, Japan, in 2004. He is an Assistant Professor at the Institute of Scientific and Industrial Research (ISIR), Osaka University, and also a PRESTO Researcher at the Japan Science and Technology Agency. At ISIR, his research focuses on the study of data mining, machine learning, and multidimensional databases. 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.

Takashi Washio received his Ph.D. 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 at the Institute of Scientific and Industrial Research (ISIR), Osaka University. At ISIR, his research focuses 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, the Journal Award of Computer Aided Chemistry in 2002, and 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.