GTRACE2: Improving Performance Using Labeled Union Graphs

Akihiro Inokuchi[1,2] and Takashi Washio[1]

[1] The Institute of Scientific and Industrial Research, Osaka University
Mihogaoka 8-1, Ibaraki, Osaka 567-0047, Japan
[2] PRESTO, Japan Science and Technology Agency
{inokuchi,washio}@ar.sanken.osaka-u.ac.jp

Abstract. The mining of a complete set of frequent subgraphs from labeled graph data has been studied extensively. Recently, much attention has been given to frequent pattern mining from graph sequences. In this paper, we propose a method to improve GTRACE which mines frequent patterns called FTSs (Frequent Transformation Subsequences) from graph sequences. Our performance study shows that the proposed method is efficient and scalable for mining both long and large graph sequence patterns, and is some orders of magnitude faster than the conventional method.

Keywords: Frequent Pattern, Graph Sequence, Labeled Union Graph.

1 Introduction

Studies on data mining have established many approaches for finding characteristic patterns from various structured data. Graph Mining [5,12,9], 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. Although the major methods 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 [4]. Accordingly, these conventional methods are not suitable for 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 where each human and each relationship between two humans correspond to a vertex and an edge, respectively. If a human joins (or leaves) the community in the human network, the numbers of vertices 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 given to frequent pattern mining from graph sequences [6,2,1,7]. Figure 1 (a) shows an example of a graph sequence consisting of 4 steps where each contains vertices denoted by 5 unique IDs.

M.J. Zaki et al. (Eds.): PAKDD 2010, Part II, LNAI 6119, pp. 178–188, 2010.

In [6], we proposed a new method, called GTRACE (Graph TRAnsformation sequenCE mining), for mining 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 unique IDs, it is intractable for graph sequences containing more steps and unique IDs than those in the Enron graph sequences.

In this paper, we propose a method to improve the efficiency of GTRACE mining frequent patterns called FTSs (Frequent Transformation Subsequences) from graph sequences. Our performance study shows that the proposed method is efficient and scalable for mining both long and large graph sequence patterns, and is some orders of magnitude faster than GTRACE. Although this paper focuses on undirected graphs where only the vertices have labels, the proposed method is applicable to both directed graph and undirected graphs where the edges also have labels without loss of generality.

2 Representation of Graph Sequences

In this section, we briefly review a compilation used to compactly represent graph sequences in GTRACE. Figure 1 (a) shows an example of a 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 [6], we proposed transformation rules to represent graph sequences compactly under the assumption that "the change over adjacent graphs is gradual". In other word, only a small part of the graph changes between two successive graphs $g^{(j)}$ and $g^{(j+1)}$ in a graph sequence, while the other parts remain unchanged. In the aforementioned human networks and the gene networks, these assumptions certainly hold, since most of the changes of the vertices are progressive over successive steps. The direct representation of a graph sequence is not compact, because many parts of a graph remain unchanged over several steps and are therefore redundant in the representation. On the other hand, a graph sequence is compactly represented by introducing a representation of graph transformation based on rules of insertion, deletion, and relabeling of vertices and edges under the assumption of gradual changes.

A labeled graph g is represented as $g = (V, E, L, f)$, where $V = \{v_1, \cdots, v_z\}$ 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 such that $f : V \to L$. $V(g)$, $E(g)$, and $L(g)$ are sets of vertices, edges and labels of g, respectively. A graph sequence is represented as $d = \langle g^{(1)}\ g^{(2)} \cdots g^{(n)} \rangle$, where the superscript integer of each g is the ordered step in

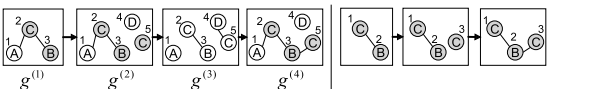

Fig. 1. Examples of a graph sequence and a mined frequent pattern

the graph sequence. We assume that each vertex v is mutually distinct from the others in any $g^{(j)}$ and keeps a unique ID $id(v)$ in d. We define the set of unique IDs to be $ID_V(d) = \{id(v) \mid v \in V(g^{(j)}), g^{(j)} \in d\}$ and the set of pairs of unique IDs to be $ID_E(d) = \{(id(v), id(v')) \mid (v, v') \in E(g^{(j)}), g^{(j)} \in d\}$.

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

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

Definition 1. *Given a graph sequence $d = \langle g^{(1)} \cdots g^{(n)} \rangle$, each graph $g^{(j)}$ in d is called an "interstate". Moreover, The differences between the graphs $g^{(j)}$ and $g^{(j+1)}$ 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 $g^{(j,m_j)} = g^{(j+1)}$. Each graph $g^{(j,k)}$ is called an "intrastate". The graph 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 a sequence. On the other hand, the order of intrastates is the order of graphs in the artificial interpolation, and there can be various interpolations between the graphs $g^{(j)}$ and $g^{(j+1)}$. We limit the interpolations to be compact and unambiguous by choosing one with the shortest length in terms of graph edit distance.

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

The transformation is represented by the following "transformation rule (TR)".

Definition 3. *A transformation rule (TR) which transforms $g^{(j,k)}$ to $g^{(j,k+1)}$ is represented by $tr^{(j,k)}_{[o_{jk}, l_{jk}]}$, where*

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

For the sake of simplicity, we denote the transformation rule by omitting the subscripts of o_{jk} and l_{jk} except in the case of ambiguity. We introduced five TRs defined in Table 1. In summary, we give the following definition of a transformation sequence.

Definition 4. *An intrastate sequence $s^{(j)} = \langle g^{(j,1)} \cdots g^{(j,m_j)} \rangle$ is represented by $seq(s^{(j)}) = \langle tr^{(j,1)}_{[o,l]} \cdots tr^{(j,m_j-1)}_{[o,l]} \rangle$. This is called an "intrastate transformation sequence". Moreover, a graph sequence $d = \langle g^{(1)} \cdots g^{(n)} \rangle$ is represented by an "interstate transformation sequence" $seq(d) = \langle seq(s^{(1)}) \cdots seq(s^{(n-1)}) \rangle$.* ■

Table 1. Transformation rules (TRs) to represent graph sequence data

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

Arguments l of the transformations of vertex deletion vd, edge insertion ei, and edge deletion ed are dummy and represented by '$\bullet$'.

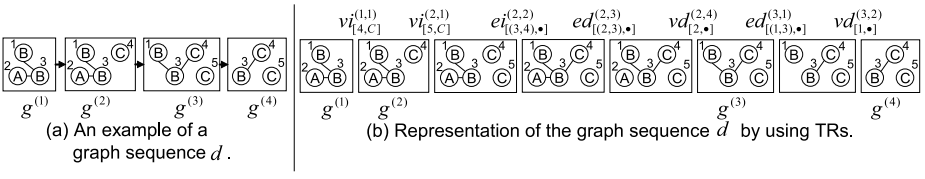

Fig. 2. A graph sequence and its TRs

The notation of the intrastate transformation sequence is far more compact than the original graph based representation, since only differences between two successive intrastates appear in the sequence. In addition, computing a sequence of TRs based on differences between two graphs is solvable in linear time, because all vertices have unique IDs.

Example 2. In Fig. 2 (a), a graph sequence is represented by a sequence of insertions and deletions of vertices and edges as shown in Fig. 2 (b). The sequence is compiled into $\langle vi^{(1,1)}_{[4,C]} vi^{(2,1)}_{[5,C]} ei^{(2,2)}_{[(3,4),\bullet]} ed^{(2,3)}_{[(2,3),\bullet]} vd^{(2,4)}_{[2,\bullet]} ed^{(3,1)}_{[(1,3),\bullet]} vd^{(3,2)}_{[1,\bullet]} \rangle$.

3 Mining Frequent Transformation Subsequences

In this section, we briefly review how GTRACE mines frequent transformation subsequences (FTSs) from a given set of graph sequences. To mine FTSs from a set of compiled graph sequences, we define an inclusion relation between transformation sequences. When a transformation sequence $seq(d)$ includes another transformation sequence $seq(d')$, it is denoted by $seq(d') \sqsubseteq seq(d)$ whose detail definition is provided in [6].

As mentioned in [6], to mine FTSs consisting of mutually relevant vertices only, we define the relevancy between unique IDs of vertices and edges as follows.

Definition 5. *Unique IDs in* $d = \langle g^{(1)} \cdots g^{(n)} \rangle$ *are relevant one another, and* d *is called a "relevant graph sequence", if the union graph* $g_u(d)$ *of* d *is a connected graph. We define the union graph of* d *to be* $g_u(d) = (V_u, E_u)$ *where*

$$V_u = \{id(v) \mid v \in V(g^{(j)}), g^{(j)} \in d\}, \text{and} \quad (1)$$

$$E_u = \{(id(v), id(v')) \mid (v, v') \in E(g^{(j)}), g^{(j)} \in d\}. \quad (2)$$

■

This union graph of the transformation sequence $seq(d)$ is also defined similar to Definition 5.

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

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

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 settings, we state our mining problem as follows.

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

To enumerate all rFTSs efficiently, GTRACE first generates a union graph for each graph sequence in DB based on the definition of a union graph. Subsequently, all connected frequent subgraphs in these union graphs are enumerated by using the conventional Graph Mining algorithm. At each time the algorithm outputs a connected frequent subgraph, an altered version of PrefixSpan [10] is called to mine rFTSs from transformation subsequences generated by the following projection.

Definition 6. *Given a graph sequence* $\langle id, d \rangle \in DB$ *and a connected graph* g*, we define a function "*$proj_1$*" to project* $seq(d)$ *to its subsequences.*

$$proj_1(\langle id, d \rangle, g) = \{\langle id', seq(d') \rangle \mid id = id', seq(d') \sqsubseteq seq(d) \text{ s.t. } g_u(d') = g\}. \quad ■$$

A data ID id' is attached to each transformation subsequence produced by the projection to calculate the exact support value of each rFTS, since multiple transformation subsequences are produced from a graph sequence $\langle id, d \rangle$ in the projection. Since the union graph of an rFTS is also frequent in the union graphs of all $\langle id, d \rangle \in DB$, we can enumerate all rFTSs from the projected transformation subsequences if all connected frequent subgraphs among the union graphs of all $\langle id, d \rangle$ are given.

Example 3. Given the graph sequence d in Fig. 3 (a), $seq(d)$ is represented by $seq(d) = \langle \underline{vi^{(1,1)}_{[3,C]} ei^{(1,2)}_{[(1,3),\bullet]} ei^{(1,3)}_{[(2,3),\bullet]}} vi^{(2,1)}_{[4,A]} ei^{(2,2)}_{[(1,4),\bullet]} \underline{ed^{(2,3)}_{[(1,3),\bullet]}} \rangle$, and its union graph $g_u(d)$ is depicted in Fig. 3 (b). Given a graph g which is a subgraph of $g_u(d)$ as shown in Fig. 3 (d), one of transformation sequences in $proj_1(\langle id, seq(d) \rangle, g)$ is $\langle id, seq(d') \rangle = \langle id, \langle vi^{(1,1)}_{[3,C]} ei^{(1,2)}_{[(1,3),\bullet]} ei^{(1,3)}_{[(2,3),\bullet]} ed^{(2,1)}_{[(1,3),\bullet]} \rangle \rangle$ as depicted in Fig. 3 (c), where this subsequence matches with the underlined rules in $seq(d)$.

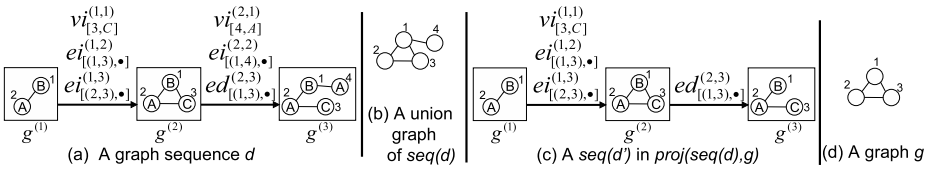

Fig. 3. An example of the projection

```
1) GTRACE(DB, σ′)
2)     G_u = {g_u(d) | ⟨id, d⟩ ∈ DB}
3)     for g =FrequentSubgraphMiner(G_u, σ′); untill g ≠ null{
4)         proj_1(DB, g) = ⋃_{⟨id,d⟩∈DB} proj_1(⟨id, d⟩, g)
5)         F′ = FTSMiner(proj_1(DB, g), σ′)
6)         F = F ∪ {α | α ∈ F′ ∧ g_u(α) = g}
7)     }
```

Fig. 4. Algorithm for mining rFTSs

Figure 4 shows an algorithm for enumerating all rFTSs F from DB. First, a set G_u of the union graphs of graph sequences DB is generated in Line 2. Assuming that the function call "FrequentSubgraphMiner" [8] repeatedly and exhaustively outputs connected frequent subgraphs g in G_u one at a time in Line 3, FTSMiner, which is the altered PrefixSpan [10,6], is called in Line 5 with the transformation sequences projected in Line 4 to mine rFTSs from $proj_1(DB, g)$. Finally, rFTSs mined from $proj_1(DB, g) = \bigcup_{\langle id,d\rangle \in DB} proj_1(\langle id, d\rangle, g)$, whose union graphs are isomorphic to g, are added to F in Line 6. These processes are continued until the connected frequent subgraph g is exhausted in FrequentSubgraphMiner. We have implemented FrequentSubgraphMiner using AcGM [8] which is one of the conventional Graph Mining methods.

4 Proposed Method: GTRACE2

Most of the computation time of GTRACE is used to run the altered PrefixSpan. The reason why the PrefixSpan used in GTRACE needs so much computation time is as follows. Let G_u and g be a set of union graphs of all $\langle id, d\rangle \in DB$ and a frequent connected subgraph mined by FrequentSubgraphMiner from G_u, respectively. The union graphs in G_u are often dense even if each interstate in graph sequences is sparse, since a union graph in G_u is generated by superimposing interstates in a graph sequence. In addition, since the union graph of a graph sequence is a graph with no labels, there exist many injective functions $V(g) \rightarrow V(g_u)$ between g and a dense union graph $g_u(d) \in G_u$ such that g is a subgraph of $g_u(d)$. Therefore, many projected transformation subsequences are produced from the graph g and one graph sequence d such that g is a subgraph of $g_u(d)$ according to the definition of projection.

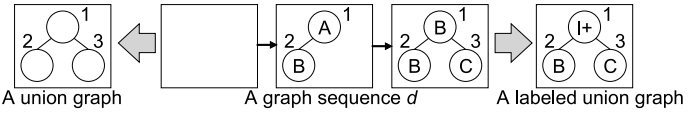

Fig. 5. Union Graph and Labeled Union Graph

To reduce the number of transformation subsequences produced by the projection, we redefine the union graph as follows:

Definition 7. *We redefine a union graph of* d *as* $g_u(d) = (V_u, E_u, L \cup \{l_+\}, f_u)$ *such that*

$$f_u(o)_{|o \in V_u} = \begin{cases} l \;\; \textit{if always } f(v) = l \textit{ for } v \in V(g^{(j)}) \\ \qquad \textit{such that } g^{(j)} \in d \textit{ and } id(v) = o \\ l_+ \;\; \textit{otherwise} \end{cases} \tag{3}$$

where V_u *and* E_u *are given by Eqs. (1) and (2), respectively.* L *is a set of vertex labels in* d*,* f *is a function to assign vertex label* $l \in L$ *to each vertex in interstates in* d*, and* $l_+ \notin L$*.* ■

The union graph defined here is a labeled graph, although the union graph defined in Section 3 is an unlabeled graph. So, we call the union graph we have defined here a labeled union graph. A label assigned to each vertex in the labeled union graph is determined by Eq. (3). If the vertices with unique ID o in interstates in d always have the identical label $l \in L$, a vertex o in the labeled union graph of d has the label l. Otherwise, the vertex o has a label l_+ such that $l_+ \notin L$.

Example 4. Figure 5 shows a union graph and a labeled union graph generated from the same graph sequence. Since two vertices with unique ID 1 in the graph sequence d have different labels, the corresponding vertex in the labeled union graph has a label l_+.

As mentioned in Section 3, GTRACE generates union graphs of all graph sequences in DB and mines all frequent connected subgraph patterns using AcGM. In this process, AcGM checks whether a pattern is included in each union graph. Since vertices with label l_+ in a labeled union graph should match any vertex in a pattern, the subgraph isomorphism test used in AcGM is altered as follows. Given two graphs $g(V, E, L, f)$ and $g'(V', E', L', f')$, g' is a subgraph of g, if there exists an injective function $\phi : V' \to V$ that satisfies the following conditions for $\forall v, v_1, v_2 \in V'$.

1. $(\phi(v_1), \phi(v_2)) \in E$, if $(v_1, v_2) \in E'$, and
2. $f(\phi(v)) = f'(v)$ or $f(\phi(v)) = l_+$.

By integrating the definition of the labeled union graph and the subgraph isomorphism test with GTRACE, we propose a new method called GTRACE2 to mine all rFTSs from graph sequences. According to the following lemma, we reduce the computation time to mine all rFTSs from graph sequences.

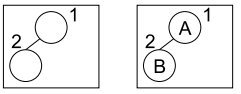

Fig. 6. Inputs of projection of GTRACE and GTRACE2

Lemma 1. *If g_1 is an unlabeled graph generated by removing all labels from a labeled graph g_2 to be used as input of projection in GTRACE2, then*

$$|\cup_{\langle id,d\rangle \in DB} proj_2(\langle id, d\rangle, g_2)| \leq |\cup_{\langle id,d\rangle \in DB} proj_1(\langle id, d\rangle, g_1)|,$$

where $proj_1$ and $proj_2$ are functions to project a graph sequence in GTRACE and GTRACE2, respectively. In addition, for $\langle id_2, seq(d_2)\rangle \in proj_2(\langle id_2, d\rangle, g_2)$, there must exists a transformation sequence $\langle id_1, seq(d_1)\rangle \in proj_1(\langle id_1, d\rangle, g_1)$ such that $id_1 = id_2$ and $seq(d_2) \sqsubseteq seq(d_1)$. Therefore, the average number of TRs in transformation sequences in $\cup_{\langle id,d\rangle \in DB} proj_2(\langle id, d\rangle, g_2)$ is less than or equal to the average number of TRs in transformation sequences in $\cup_{\langle id,d\rangle \in DB} proj_1(\langle id, d\rangle, g_1)$. ■

The proof of Lemma 1 is omitted due to the lack of space, but an example is given in Example 5. As shown in the experiments in [10], the computation time to run PrefixSpan is proportional to the number of sequences in its input, and it increases exponentially when the average number of items in the sequences increases. According to Lemma 1, since the number of transformation sequences generated by the projection in GTRACE2 usually decreases and the average number of TRs in the transformation sequences usually becomes less than in the original GTRACE, the computation time for running the altered PrefixSpan in GTRACE2 is reduced.

Example 5. The graph sequence $\langle 1, d\rangle$ at the center of Fig. 5 is represented as $\langle 1, \langle vi^{(1,1)}_{[1,A]} vi^{(1,2)}_{[2,B]} ei^{(1,3)}_{[(1,2),\bullet]} vi^{(2,1)}_{[3,C]} ei^{(2,2)}_{[(1,3),\bullet]} vr^{(2,3)}_{[1,B]}\rangle\rangle$, and its union graph and labeled union graph are shown in Fig. 5. Given the graph g_1 shown in Fig. 6 (a) as input of the projection in GTRACE, two vertices in g_1 correspond to vertices with unique IDs 1 and 2 or vertices with unique IDs 1 and 3 in the union graph $g_u(d)$. Therefore, $proj_1(\langle 1, d\rangle, g_1)$ is

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

On the other hand, given the graph g_2 shown in Fig. 6 (b) as input of the projection in GTRACE2, $proj_2(\langle 1, d\rangle, g_2)$ is

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

since two vertices with unique ID 1 and 2 in the input graph g_2 correspond to vertices with unique IDs 1 and 2 in the labeled union graph $g_u(d)$, respectively. This projected transformation sequence does not include $vr^{(2,3)}_{[1,B]}$ to satisfy $g_u(d) = g$ in Definition 8.

Table 2. Results for the Enron dataset

$\|ID_V(d)\|$	80	90	100	110	120	140	150	170	182
GTRACE comp. time	0.18	42.0	1509.5	–	–	–	–	–	–
comp. time for PrefixSpan	0.048	37.7	1407.7	–	–	–	–	–	–
# of subgraph patterns	4	7	10	–	–	–	–	–	–
avg. # of trans. seq.	809	19249	269726	–	–	–	–	–	–
avg. # of TRs	10.3	23.4	30.1	–	–	–	–	–	–
GTRACE2 comp. time	0.14	0.25	0.34	0.39	0.63	0.67	1.2	3.8	4.1
comp. time for PrefixSpan	0.015	0.062	0.078	0.11	0.22	0.20	0.45	2.5	2.7
# of subgraph patterns	22	26	28	29	36	39	45	50	52
avg. # of trans. seq.	197	415	585	656	813	892	1333	2078	2201
avg. # of TRs	4.7	6.2	6.5	6.4	6.7	6.7	7.0	7.9	7.9
σ', n	60%	50%	40%	20%	5%	2%	5	6	7
GTRACE comp. time	7.5	132.4	–	–	–	–	–	–	–
comp. time for PrefixSpan	1.7	60.9	–	–	–	–	–	–	–
# of subgraph patterns	4	7	–	–	–	–	–	–	–
avg. # of trans. seq.	41334	202432	–	–	–	–	–	–	–
avg. # of TRs	16.8	22.3	–	–	–	–	–	–	–
GTRACE2 comp. time	0.52	0.92	1.2	2.7	25.6	327.3	1.3	2.2	4.1
comp. time for PrefixSpan	0.03	0.11	0.17	1.4	24.0	325.5	0.50	0.97	2.7
# of subgraph patterns	13	24	29	46	81	124	44	47	52
avg. # of trans. seq.	3288	2906	3028	2424	1468	991	1940	2293	2201
avg. # of TRs	4.7	6.0	6.6	7.7	8.2	8.7	6.0	6.7	7.9

comp. time: computation time [sec],
comp. time for PrefixSpan: total computation time to run the altered PrefixSpan,
# of subgraph patterns: the number of frequent connected subgraphs mined by AcGM,
avg. # of trans. seq.: the average number of transformation sequences in $proj_i(DB, g)$,
avg. # of TRs: the average number of TRs in transformation sequences in $proj_i(DB, g)$
Default: minimum support σ' =15%, # of vertex labels $|L_v| = 8$, # of edge labels $|L_e| = 1$,
# of persons $|ID_V(d)| = 182$, # of interstates in a graph sequence n=7.

5 Experiment and Discussion

The proposed method was implemented in C++. The experiments were executed on an HP xw4600 with an Intel Core 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 both artificial and real world graph sequence data. Due to the lack of space, we report the experiments using the real world data.

To assess the practicality of the proposed method, it was applied to the Enron Email Dataset [3]. We assigned a unique 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 President} was assigned to each vertex. We then obtained a set of weekly graph sequence data DB. The total number of weeks, *i.e.*, number of sequences, was 123. We randomly sampled $|ID_V(d)|(= 1 \sim 182)$ persons to form DB.

Table 2 shows the computation times [sec] to run GTRACE and GTRACE2, total computation times [sec] to run the altered PrefixSpan, the numbers of frequent connected subgraphs mined by AcGM, the average numbers of transformation sequences in $proj_i(DB, g)$, and the average numbers of transformation sequences in $proj_i(DB, g)$ obtained for various numbers of unique IDs (persons) $|ID_V(d)|$, minimum support σ', and numbers of interstates n in each graph sequence of the dataset. All the other parameters were set to the default values indicated at the bottom of the table. Thus, the dataset with the default values

as contained 123 graph sequences each consisting of 182 persons (unique IDs) and 7 interstates. The parameter n =5, 6, or 7 indicates that each sequence d in DB consists of 5, 6, or 7 steps (interstates) from Monday to Friday, Saturday, or Sunday, respectively. When the required computation time exceeds two hours or a memory overflow occurs, the results are indicated by "-".

The upper, lower left, and lower right parts of the table show experimental results with regard to the number of persons (unique IDs), the minimum support threshold, and the number of interstates in graph sequences in the graph sequence database, respectively. The table indicates that GTRACE proved intractable for the graph sequence dataset generated from the default values, althought the proposed method GTRACE2 is tractable with respect to the database. In addition, the computation times for both GTRACE and GTRACE2 are exponential with respect to the increases of the number of $|ID_V(d)|$ and the number of interstates in the graph sequence database and with respect to the decrease of the minimum support threshold. The main reason that the computation time increases is the increase in the number of frequent patterns.

The computation times for GTRACE2 are much smaller than those for GTRACE, although the number of times to call the altered PrefixSpan increases. Most of computation time of GTRACE is used running the altered PrefixSpan. As shown in [10], the computation time of PrefixSpan is proportional to the number of sequences in its input and increase exponentially with respect to the average number of items in sequences in its input. The scalability of GTRACE2 comes from reducing the number of transformation sequences and the number of TRs in transformation sequences in the projected database by using the labeled union graph proposed in Section 4. GTRACE2 is practical, because it can be applied to graph sequences that are both longer and larger than those to which GTRACE can be applied.

6 Conclusion

In this paper, we proposed a method to improve GTRACE which mines a set of relevant frequent transformation subsequences (rFTSs) from given graph sequences by defining the labeled union graph. We developed a graph sequence mining program GTRACE2, and confirmed its efficiency and practical performance through computational experiments using artificial and real world datasets. Our performance study showed that the proposed method is some orders of magnitude faster than the conventional method, and is efficient and scalable for mining both long and large graph sequence patterns. Recently, we have proposed a method for mining another class of frequent patterns, called FRISSs (Frequent, Relevant, and Induced Subgraph Subsequences), from graph sequence [7]. The principle proposed in this paper using labeled graphs can be applied to the method. In real applications, it is hard to enumerate usefule and interesting FTSs which are exactly included in graph sequences. In future work, we plan to extend GTRACE2 to mine FTSs which are approximately included in graph sequences by using sliding windows and time constraints proposed in [11].

References

1. Berlingerio, M., et al.: Mining Graph Evolution Rules. In: Proc. of Euro. Conf. on Principles and Practice of Knowledge Discovery in Databases, pp. 115–130 (2009)
2. Borgwardt, K.M., et al.: Pattern Mining in Frequent Dynamic Subgraphs. In: Proc. of Int'l Conf. on Data Mining, pp. 818–822 (2006)
3. Enron Email Dataset, http://www.cs.cmu.edu/~enron/
4. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
5. Inokuchi, A., et al.: An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 13–23. Springer, Heidelberg (2000)
6. Inokuchi, A., Washio, T.: A Fast Method to Mine Frequent Subsequences from Graph Sequence Data. In: Proc. of Int'l Conf. on Data Mining, pp. 303–312 (2008)
7. Inokuchi, A., Washio, T.: Mining Frequent Graph Sequence Patterns Induced by Vertices. In: Proc. of SIAM Int'l Conf. on Data Mining (2010)
8. Inokuchi, A., et al.: A Fast Algorithm for Mining Frequent Connected Subgraphs. IBM Research Report, RT0448 (2002)
9. Nijssen, S., Kok, J.N.: A Quickstart in Frequent Structure Mining can Make a Difference. In: Proc. of Int'l Conf. on Knowledge Discovery and Data Mining, pp. 647–652 (2004)
10. Pei, J., et al.: PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In: Proc. of Int'l Conf. on Data Eng., pp. 2–6 (2001)
11. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Proc. of Int'l Conf. on Extending Database Technology, pp. 3–17 (1996)
12. Yan, X., Han, J.: gSpan: Graph-Based Substructure Pattern Mining. In: Proc. of Int'l Conf. on Data Mining, pp. 721–724 (2002)