Discovering Frequent Closed Partial Orders from Strings

Jian Pei, Haixun Wang, *Member*, *IEEE Computer Society*, Jian Liu, Ke Wang, Jianyong Wang, *Member*, *IEEE Computer Society*, and Philip S. Yu, *Fellow*, *IEEE*

Abstract—Mining knowledge about ordering from sequence data is an important problem with many applications, such as bioinformatics, Web mining, network management, and intrusion detection. For example, if many customers follow a partial order in their purchases of a series of products, the partial order can be used to predict other related customers' future purchases and develop marketing campaigns. Moreover, some biological sequences (e.g., microarray data) can be clustered based on the partial orders shared by the sequences. Given a set of items, a total order of a subset of items can be represented as a string. A string database is a multiset of strings. In this paper, we identify a novel problem of *mining frequent closed partial orders from strings.* Frequent closed partial orders capture the nonredundant and interesting ordering information from string databases. Importantly, mining frequent closed partial orders can discover meaningful knowledge that cannot be disclosed by previous data mining techniques. However, the problem of mining frequent closed partial orders is challenging. To tackle the problem, we develop *Frecpo* (for Frequent closed partial order), a practically efficient algorithm for mining the *complete set* of frequent closed partial orders from large string databases. Several interesting pruning techniques are devised to speed up the search. We report an extensive performance study on both real data sets and synthetic data sets to illustrate the effectiveness and the efficiency of our approach.

Index Terms—Frequent patterns, closed patterns, partial orders, strings, data mining.

1 Introduction

Mining ordering information from sequence data is an important data mining task. Sequential pattern mining [3] can be regarded as mining frequent segments of total orders from sequence data. However, sequential patterns are often insufficient to concisely capture the general ordering information.

Example 1 (Motivation). Suppose MapleBank in Canada wants to investigate whether there are some orders which customers will follow to open their accounts. A database DB in Table 1 about four customers' sequences of opening accounts in MapleBank is analyzed.

Although there does not exist a "global template" for customers' behavior, finding the frequent patterns may help to capture their habits. Given a support threshold min_sup, a sequential pattern is a sequence s which appears as subsequences of at least min_sup sequences. For example, let $min\_sup = 3$. The following four sequences are sequential patterns since they are subsequences of three sequences, 1, 2, and 4, in DB:

$$\text{CHK} \rightarrow \text{MMK} \rightarrow \text{MORT} \rightarrow \text{RESP};$$
$$\text{CHK} \rightarrow \text{MMK} \rightarrow \text{MORT} \rightarrow \text{BROK};$$
$$\text{CHK} \rightarrow \text{RRSP} \rightarrow \text{MORT} \rightarrow \text{RESP};$$
$$\text{CHK} \rightarrow \text{RRSP} \rightarrow \text{MORT} \rightarrow \text{BROK}.$$

Sequential patterns capture the frequent account opening patterns shared by customers. However, the four sequential patterns cannot completely capture the ordering shared by customers 1, 2, and 4. It is easy to see that a partial order R as shown in Fig. 1 is shared by the three account opening sequences.

Moreover, we can make the following two interesting and stimulating observations.

- The partial order R summarizes the four sequential patterns—the four sequential patterns are paths in partial order R.
- The partial order R provides more information about the ordering than the sequential patterns. For example, R indicates that some customers often open money market accounts and RRSP accounts in any order, but those two accounts are often opened before the mortgage account. One possible business explanation for this ordering is that, according to Canadian revenue regulation, a person can use an amount from her/his RRSP account free of income tax to purchase her/his first house if some prerequisites are satisfied. Such information is not presented in the sequential patterns explicitly.

Partial order R is shared by a good number of customers and is meaningful in business. For example, after a customer opens a checking account in MapleBank,

- J. Pei and K. Wang are with the School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada V5A 1S6. E-mail: {jpei, wangk}@cs.sfu.ca.
- H. Wang and P.S. Yu are with the IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532. E-mail: {haixun, psyu}@us.ibm.com.
- J. Liu is with Efficient Frontier Inc., 321 Castro St., Ste. 201, Mountain View, CA 94041-1205. E-mail: jian.liu@efrontier.com.
- J. Wang is with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China. E-mail: jianyong@mail.tsinghua.edu.cn.

Manuscript received 26 Aug. 2005; revised 9 May 2006; accepted 30 May 2006; published online 19 Sept. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0335-0805.

TABLE 1
A Database DB of Sequences of Account Opening

Account codes and explanation

Account code	Account type
CHK	Checking account
MMK	Money market
RRSP	Retirement Savings Plan
MORT	Mortgage
RESP	Registered Education Savings Plan
BROK	Brokerage

Customer Records

Cid	Sequence of account opening
1	CHK → MMK → RRSP → MORT → RESP → BROK
2	CHK → RRSP → MMK → MORT → RESP → BROK
3	MMK → CHK → BROK → RESP → RRSP
4	CHK → MMK → RRSP → MORT → BROK → RESP

a customer representative from the bank may call the customer to introduce the money market account and the RRSP account. However, at this point, the bank may not want to mail the customer brochures on RESP accounts.

This example motivates the idea of using frequent partial orders to effectively summarize sequential patterns and provide more general and more concise ordering information.

Given a set of items (or events), a total order of a subset of items can be represented as a string. A string database is a multiset of strings. The knowledge about ordering, especially the frequent partial orders in string databases, has many applications. Here, we list four of them.

Application 1: Bioinformatics. Ordering information is often important in the analysis of biological experiment data. For example, to discover patterns in gene expression matrices, one promising approach [5] is to look for order-preserving submatrices (OPSMs). That is, in an n by m gene expression matrix for n genes and m experiments, each element $v_{i,j}$ gives the expression level of a gene g_i in an experiment e_j. A submatrix is order-preserving if the expression levels of all genes in the submatrix induce the same (linear or partial) ordering of the experiments. As indicated in [5], such a pattern may arise if the experiments in the order-preserving submatrix represent distinct stages in the progress of a disease or in a cellular process, and the expression levels of all genes in the submatrix vary across the stages in the same way. Moreover, [15] also shows that a partial order of conditions shared by a group of genes may indicate that the genes form a coexpressed group and they respond to a sequence of environment stimuli.

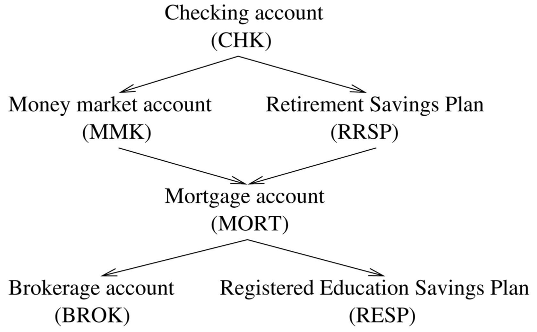


Fig. 1. A frequent partial order R in Example 1.

Application 2: Process model mining, Web mining, and market basket analysis. The workflow paradigm has been extensively used to specify how business processes should be conducted. It is often desirable to construct process models from logs of past, unstructured executions of a given process [1], [25].

In Web mining and market basket analysis, a critical task is to identify groups of customers in which all customers' sequences of purchases induce the same ordering of a series of products. In previous studies, sequential patterns are often used for this purpose. However, as shown in Example 1, sequential patterns may not be able to concisely capture the general ordering information. Instead, a partial order can model the customers' purchase behavior better. Thus, it can be more informative and more effective to use frequent partial orders in place of sequential patterns in many cases. Moreover, selected frequent partial orders can be used as signatures of customer behavior in classification and clustering analysis.

Application 3: Network management and intrusion detection. In network management, it is important to characterize network traffic. Frequent partial orders obtained from network packet scheduling data may disclose frequent routing paths and identify possible bottlenecks of networks.

Moreover, it is important to discover the signatures (i.e., distinct features) of normal network access and intrusions. Consider misuse detection, where a training data set containing both labeled normal activities and intrusions is available. We can mine from the training data set partial orders which are frequent in the subset of intrusions and are rare in the subset of normal activities. Such frequent partial orders can be used to identify malicious activities in the future. On the other hand, in anomaly detection, frequent partial orders can be used to characterize the major patterns of network accesses. If an activity does not follow any frequent partial orders observed so far, then it can be a candidate of anomaly.

Application 4: Preference-based service. Preferences can be modeled as partial orders. It is interesting to study common preferences from a large collection of data, such as marketing survey and product evaluation. For example, a customer may be asked to rank a set of products in a marketing survey. The preference of a customer can be derived from her/his ranking. Then, it is interesting to mine the common preferences as frequent partial orders from the ranking data. Moreover, customer segmentation and marketing campaigns may be developed based on such ordering information.

The problem of mining partial orders in sequence data has been studied before from angles different from our study. Two major categories of previous work exist. The first batch of studies only look at some specific kinds of frequent partial orders. Particularly, a seminal piece [17] is on mining serial and parallel frequent episodes. As indicated in Section 3, the expression power of serial and parallel episodes may not be sufficient enough to capture the general partial orders shared by sequences.

Another category of research is on mining global partial orders, i.e., finding a single or a set of partial order(s) to fit the whole sequence data set. However, while global orders may be desirable in some applications, they may not meet the requirements in some other situations. There can be multiple different trends existing in a sequence data set. Therefore, in some applications, such as mining preferences or network packet scheduling data, a single or a set of partial order(s) may not fit the whole data set well. Moreover, developing efficient approximation algorithms for finding a partial order globally fitting a large data set well is still an open issue.

Simultaneously to this study, Casas-Garriga [7] also proposed a similar idea to use closed partial orders to summarize sequential data. However, [7] focuses on the concept and does not provide an efficient algorithm for the mining. As shown in our experimental results, the method is not efficient for large data sets. Therefore, mining frequent partial orders from sequence data remains a challenging problem.

In this paper, we study the problem of *mining frequent closed partial orders from strings* and make the following contributions:

First, we comprehensively model the problem. We identify the problem of mining frequent partial orders. There can be many frequent partial orders in a large string database. It is desirable to devise a concise and nonredundant representation for frequent closed partial orders. Technically, we develop the concept of *frequent closed partial orders* (FCPO) that captures the interesting and nonredundant information about frequent orders. The major idea is that we only present the frequent partial orders which do not have a proper superset with the same support.

We show the relationship among several types of frequent patterns which can be mined from sequence data, including frequent itemsets, frequent closed itemsets, sequential patterns, closed sequential patterns, frequent graph patterns, frequent closed graph patterns, and frequent closed partial orders. We also investigate the computational complexity of the problem.

Moreover, by a systematic survey of previous work, we show that most of the previous studies focus on series-parallel orders (SPO) [24]. However, in general, the expression power of series-parallel orders is insufficient enough to capture the closed partial orders shared by strings.

Second, we propose practically efficient algorithms to mine frequent closed partial orders. The first method is to represent every string as the set of edges in its transitive closure, and mine frequent sets of edges. It can be shown that a frequent closed set of edges is a frequent partial order in transitive closure. This leads to our first algorithm *TranClose*. Unfortunately, even though *TranClose* avoids the costly mining of frequent graph patterns [28] directly, it is still inefficient in many cases since it has to compute the closures of strings and orders, which can be much larger in size than their transitive reduction.

To tackle the problem, we propose algorithm *Frecpo*, which mines the string database directly and never computes the transitive closure. Moreover, it identifies the frequent closed partial orders in the form of transitive reduction, which is the minimal representation. It explores several interesting pruning techniques to speed up the mining process.

Last, we conduct an extensive performance study on real and synthetic data sets to examine the effectiveness of mining frequent closed partial orders and the efficiency of our methods. It shows that *Frecpo* can mine large string databases, and the mining results are interesting and provide knowledge that cannot be discovered by previous data mining techniques.

While the basic concepts and notions of frequent closed partial order mining are briefly introduced in [21], in this paper, we provide a substantially more thorough treatment of the conceptual issues and the algorithms, and report a comprehensive performance study.

The remainder of the paper is organized as follows: In Section 2, we formulate the problem of mining frequent closed partial orders from strings. We also thoroughly examine the relationship between frequent partial orders and other frequent patterns which can be mined from string databases, and prove the complexity of the problem. We systematically review related work in Section 3 and discuss why the popularly used series-parallel orders are insufficient to capture the closed partial orders shared by strings in general. The mining algorithms are developed in Section 4. We report an extensive performance study in Section 5. The paper is concluded in Section 7.

2 Problem Definition

In this section, we comprehensively model the problem of mining frequent partial orders from strings. We propose the concept of frequent closed partial order and examine the relationship among several types of frequent patterns that can be mined from sequence data. We also show the complexity of the problem.

2.1 Preliminaries

A *partial order* is a binary relation that is reflexive, antisymmetric, and transitive. A *total order* (also called linear order) is a partial order R such that for any two items x and y, if $x \neq y$, then either $R(x,y)$ or $R(y,x)$ holds.

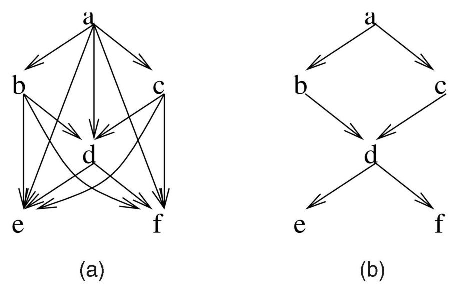


Fig. 2. A partial order and its transitive reduction. (a) A partial order. (b) The transitive reduction.

A partial order R can be expressed in a directed acyclic graph (DAG for short): the items are the vertices in the graph and $x \rightarrow y$ is an edge if and only if $(x,y) \in R$ and $x \neq y$. We also write an edge $x \rightarrow y$ as (x,y) or xy. For example, Fig. 2a shows a partial order R, which has 13 edges.

Since a partial order is transitive, some edges can be derived from the others and thus are redundant. For example, in Fig. 2a, edge $a \rightarrow d$ is redundant given edges $a \rightarrow b$ and $b \rightarrow d$. Generally, an edge $x \rightarrow y$ is *redundant* if there is a path from x to y that does not contain the edge. For a partial order R, the *transitive reduction* of R can be drawn in a *Hasse diagram*: For $(x,y) \in R$ and $x \neq y$, x is positioned higher than y; edge $x \rightarrow y$ is drawn if and only if the edge is not redundant. Fig. 2b shows the transitive reduction of the same partial order R in Fig. 2a. The transitive reduction has only six edges. For an order R, the transitive reduction may have much fewer edges.

In this paper, we draw a partial order in a Hasse diagram, i.e., its transitive reduction, and omit the isolated vertices. For example, Fig. 3 shows four partial orders R_1, R_2, R_3, and R_4, and R_1 is further a total order.

Let V be a set of items, which serves as the domain of our string database. A *string* defines a global order on a subset of V. In this paper, we focus on strings instead of general sequences and assume that each item appears in a string at most once, but not necessarily every item appears in a string.

A string can be written as $s = x_1 \cdots x_l$, where

$$x_1, \ldots, x_l \in V.$$

l is called the *length* of string s, i.e., $len(s) = l$. For strings $s = x_1 \cdots x_l$ and $s' = y_1 \cdots y_m$, s is called a *superstring* of s' and s' a *substring* of s if 1) $m \leq l$ and 2) there exist integers $1 \leq i_1 < \cdots < i_m \leq l$ such that $x_{i_j} = y_j (1 \leq j \leq m)$. We also say s contains s'. For a string database SDB, the *support* of a string s, denoted by $sup(s)$, is the number of strings in SDB that are superstrings of s.

The total order defined by string s can be written in the *transitive closure* of s, denoted by

$$\mathcal{C}(s) = \{(x_i, x_j) | 1 \leq i < j \leq l\}.$$

Please note that, in the transitive closure, we omit the trivial pairs (x_i, x_i). For example, for string $s = abcd$, $len(s) = 4$. The transitive closure is

$$\mathcal{C}(s) = \{(a,b),(a,c),(a,d),(b,c),(b,d),(c,d)\}.$$

Here, we omit the trivial pairs (a,a), (b,b), (c,c), and (d,d).

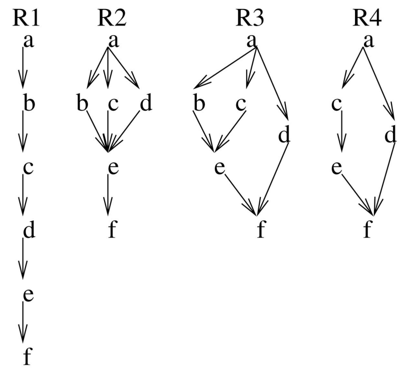


Fig. 3. Four orders $R_1 \supset R_2 \supset R_3 \supset R_4$.

The *order containment relation* is defined as, for two partial orders R_1 and R_2, if $R_1 \subset R_2$, then R_1 is said to be *weaker* than R_2 and R_2 is *stronger* than R_1. By intuition, a partially ordered set (or poset for short) satisfying R_2 will also satisfy R_1. For example, in Fig. 3, $R_4 \subset R_3 \subset R_2 \subset R_1$. Please note that R_4 covers fewer items than the other three partial orders. Trivially, we can add the missing items into the DAG as isolated vertices so that every DAG covers the same set of items. To keep the DAG simple and easy to read, we omit such isolated items.

2.2 Frequent Closed Partial Orders (FCPO)

A *string database SDB* is a multiset of strings. For a partial order R, a string s is said to *support* R if $R \subseteq \mathcal{C}(s)$. The *support of R in SDB*, denoted by $sup(R)$, is the number of strings in SDB that support R. Given a minimum support threshold min_sup, a partial order R is called *frequent* if $sup(R) \geq min\_sup$.

Following the related definitions and the order containment relation, we have the following result.

Property 2.1 (Antimonotonicity of frequent partial orders). *For a string database SDB and partial orders R and R' such that $R' \subset R$, $sup(R') \geq sup(R)$. Therefore, if R is frequent, then R' is also frequent.*

To avoid the triviality, instead of reporting all frequent partial orders, we can mine the representative ones only.

Example 2 (Frequent closed partial orders). Let us consider string database DB in Table 1 again. The four sequential patterns discussed in Example 1 can be regarded as frequent partial orders which are supported by strings 1, 2, and 4. As discussed before, given that the partial order R in Fig. 1 is also supported by strings 1, 2, and 4, the four sequential patterns as frequent partial orders are redundant.

There does not exist another partial order R' such that R' is stronger than R in Fig. 1 and is also supported by strings 1, 2, and 4. In other words, R is the strongest one among all frequent partial orders supported by strings 1, 2, and 4. Thus, the partial order R is not redundant and can be used as the representative of the frequent partial orders supported by strings 1, 2, and 4. Technically, R is a frequent closed partial order.

A partial order R is *closed* in a string database SDB if there exists no partial order $R' \supset R$ such that $sup(R) = sup(R')$. A partial order R is a *frequent closed partial order* if it is both frequent and closed.

Problem Definition. The problem of mining frequent closed partial orders from strings is to find the complete set of frequent closed partial orders in a given string database SDB with respect to a minimum support threshold min_sup.

2.3 Various Types of Frequent Patterns from Strings

For a string database SDB and a minimum support threshold min_sup, in addition to frequent closed partial orders, we can mine some other types of frequent patterns as follows:

Frequent itemsets [2] and frequent closed itemsets [18]. If the ordering information in a string is ignored, a string can be treated as a set of items. For a set of items $X \subseteq I$, $sup(X)$ is the number of strings in SDB in which X appears. X is a *frequent itemset* if $sup(X) \geq min\_sup$. A frequent itemset X is a *frequent closed itemset* if there exists no $X' \supset X$ such that $sup(X) = sup(X')$.

Sequential patterns [3] and closed sequential patterns [29]. For string s, $sup(s)$ is the number of strings in SDB which contain s as a substring. s is a *sequential pattern* if $sup(s) \geq min\_sup$. In other words, a sequential pattern is a frequent total order on a subset of items. A sequential pattern s is a *closed sequential pattern* if there exists no proper supersequence s' of s such that $sup(s') = sup(s)$.

Graph patterns [13] and closed graph patterns [28]. Since a string defines a total order, its transitive closure can be viewed as a DAG. For a DAG G, $sup(G)$ is the number of graphs in which G is an embedded subgraph. G is a *frequent graph pattern* if $sup(G) \geq min\_sup$. A frequent graph pattern G is a *frequent closed graph pattern* if there exists no graph G' such that $sup(G) = sup(G')$ and G is an embedded subgraph of G'.

Then, *what are the relationships among the above types of frequent patterns?* We have the following results based on the related definitions.

Corollary 2.1 (FPO, frequent itemsets, and sequential patterns). *The set of items in a frequent partial order is a frequent itemset. Moreover, if R is a frequent partial order, then every path s in R is a sequential pattern.*

In a directed acyclic graph (DAG), a vertex v is a *sink* if no edge leaves v. A vertex v is a *source* if no edge enters v.

Theorem 1 (FCPO and closed sequential patterns). *Let R be a frequent closed partial order, and $s_1, \ldots, s_k$ be all the paths in R's transitive closure graph such that each path is from a source to a sink. Then, a string s supports R if and only if it simultaneously supports $s_1, \ldots, s_k$.*

Proof. By Corollary 2.1, we have s simultaneously supports $s_1, \ldots, s_k$. To show the other direction, we only need to notice the fact that every pair $(x, y) \in R$ must be in some path from a source to a sink. That is, (x, y) must be contained in some s_i. Hence, s supports every pair in R. That is, s supports R. □

Theorem 2 (CPO and transitive closure graph patterns). *A partial order R is a frequent (closed) partial order in a string database if and only if it is a frequent (closed) graph pattern in the corresponding database of transitive closures of strings.*

Proof. We only show the case of frequent partial orders. The case of frequent closed partial orders can be proved similarly. Let SDB be a string database and GDB be the database of transitive closures of strings.

Consider a string s in SDB such that s supports R. It is easy to see that R (in transitive closure) is a subgraph of $\mathcal{C}(s)$, the transitive closure of s. Thus, $sup(R)$ in GDB is no less than that in SDB.

On the other hand, since every item appears in a string at most once, every graph pattern in GDB defines a partial order. Suppose R_G is a frequent graph pattern in GDB, and the partial order defined by R_G is R. Consider a graph G in GDB that is a supergraph of R_G. Let G be the transitive closure of s in SDB. R must be supported by s. Hence, we have $sup(R)$ in SDB is no less than that in GDB.

Based on the above, we have R is a frequent partial order in SDB if and only if its transitive closure is a frequent graph pattern in GDB. The claim on the relationship between frequent closed partial orders and frequent closed graph patterns can be proved similarly. □

2.4 Complexity Analysis

We investigate the complexity of the frequent closed partial order mining problem in two steps. First, we show that counting the number of frequent closed partial orders (called the *counting problem*) is #P-Complete. This means that the corresponding decision problem (i.e., whether there are n frequent closed partial orders in a given string database with respect to a given support threshold) is NP-hard. Second, we show that, under a polynomial time transformation, the frequent closed partial order mining problem can also be reduced to some typical frequent pattern mining problems that have been studied before.

Theorem 3 (Complexity). *The problem of counting the number of frequent closed partial orders from strings is #P-Complete.*

Proof. We construct a polynomial time parsimonious transformation from the problem of counting the number of frequent closed itemsets, which is known to be #P-Complete [6], [12], [30].

As shown in Theorem 2, the problem of mining frequent closed partial orders is equivalent to mining frequent closed graph patterns in the transitive closure graphs. We reduce the problem of mining frequent closed itemsets into mining frequent closed graph patterns in the transitive closure graphs as follows.

For a transaction database TDB, we construct a string database SDB as follows: For each transaction (i.e., itemset) in TDB, we sort all the items alphabetically and make up a string. It can be shown that every frequent partial order in SDB is a total order. Moreover, an itemset X is a frequent closed itemset in TDB if and only if the alphabetical order of items in X is a frequent closed partial order in SDB. That means the transformation is parsimonious. Clearly, the reduction is of polynomial time. □

Immediately following from Theorem 3, the corresponding decision problem, determining whether there are n frequent closed partial orders in a string database with respect to a given support threshold, is NP-hard.

Interestingly, we can also reduce the problem of mining frequent closed graph patterns in the transitive closure graphs to the frequent closed itemset mining problem as follows: For each string, its transitive closure graph can be uniquely represented as the set of edges in the graph. We treat each edge as an item. Then, a transitive closure graph becomes an itemset. It can be shown that a subgraph G is a frequent closed graph pattern if and only if its edge set is a frequent closed itemset in the transformed transaction database. Clearly, the reduction is of polynomial time. By Theorem 2, we can reduce the problem of frequent closed partial order mining problem to the frequent itemset mining problem as well.

The above shows that the two problems are in the same class of polynomial time equivalent problems. As shown in [6], [12], [30], a whole set of frequent pattern mining problems, including mining frequent (closed/maximal) itemsets, (closed/maximal) sequential patterns, and (closed/maximal) graph patterns, are in the same class, their counting problems are #P-Complete. The problem of mining frequent (closed) partial orders is also in this class. The above result is consistent with the result in [5], which shows that determining whether there exists a total order on n items of support k is NP-Complete. It is a special case of the search of frequent partial orders studied in this paper.

3 Related Work

In this section, we systematically review related work. Particularly, we address why series-parallel orders, which are extensively studied in previous work, are insufficient for capturing the ordering information shared by strings. Moreover, we review the two-step method in [7] and point out its inefficiency.

3.1 Sequential Pattern Mining

As sequence data is available in many applications, mining sequence data has been investigated extensively. There are intensive studies on mining sequential patterns [3]. Several efficient algorithms were proposed, such as GSP [22], PrefixSpan [19], SPADE [32], SPAM [4], and DISC [8].

There can be many sequential patterns. To improve the effectiveness and remove the redundant sequential patterns, closed sequential patterns can be mined [23], [26], [29]. Another approach to improve the effectiveness is to specify constraints. In [10], [20], various constraints on sequential patterns were investigated.

Sequence data can be noisy. Yang et al. [31] studied the problem of mining long patterns from noisy sequence data. In [14], the problem of mining approximate sequential patterns was addressed.

In [17], Mannila et al. considered mining frequent episodes from event sequences (basically, strings). In principle, an episode can be any partial order. However, due to the computational complexity consideration, algorithms on only series and parallel episodes were given. An episode is parallel if the partial order is trivial (i.e., $x \not\leq y$ for all $x \neq y$). An episode is serial if the partial order is a total order (i.e., for any x and y, either $x \leq y$ or $y \leq x$). It coincides with sequential pattern mining in general.

As illustrated in Section 1, mining frequent partial orders is a generalization of mining sequential patterns.

3.2 Mining Partial Orders

Recently, two interesting studies investigated the problem of mining a small set of partial orders globally fitting data best [11], [16]. Particularly, [16] addressed sequence data. However, very different from the problem studied here, [16] tried to find one or a (small) set of partial orders that fit the *whole* data set as well as possible, which is an optimization problem. An implicit assumption is that the whole data set somehow follows a global order.

Moreover, [1], [25] studied the problem of reconstructing a workflow model from a set of executions of the model, such as records in a log file. In process model mining, it is also assumed that a global workflow template exists and the mining wants to reconstruct the template as much as possible from the executions of the template. The assumption of a global template is feasible and useful in some applications, such as scheduling jobs and students taking courses.

For some other applications, such as the DNA microarray data analysis and network packet routing, there is usually no nontrivial order that can be expected globally. This paper addresses such situations. That is, we want to find the partial orders that are frequent in a database, but do not necessarily dominate the database. Some partial orders found may even conflict with each other.

There is another important difference between the work [16] and this paper. In [16], due to the complexity consideration, only series-parallel orders [24] are considered, whose definition is recalled as follows.

The *minimal series parallel* (MSP) DAG is defined as follows:

1. The DAG having a single vertex and no edges is MSP.
2. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two MSP DAGs, so is either of the DAGs constructed by the following operations:
    a. *Parallel composition*: $G_p = (V_1 \cup V_2, E_1 \cup E_2)$.
    b. *Series composition*:
    $$G_s = (V_1 \cup V_2, E_1 \cup E_2 \cup (N_1 \times R_2)),$$
    where N_1 is the set of sinks of G_1 and R_2 is the set of sources of G_2.

A partial order can be represented as a DAG. A partial order is a *series parallel order* if the transitive reduction of its DAG is an MSP DAG.

Intuitively, a series parallel order is formed by assembling objects using parallelism and serialism. An important property is that a series parallel order can be represented in a *binary decomposition tree* [24]. Then, many search problems can be solved efficiently by dynamic programming.

In [16], Mannila and Meek tried to find series parallel orders that globally fit a data set as well as possible. However, for mining frequent partial orders, series parallel

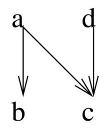


Fig. 4. The frequent partial order R is shared by $abdc$ and $dacb$.

orders may not be sufficient, since they cannot always capture all the partial orders shared by sequences.

Example 3 (Dimension 2 nonseries parallel order). Consider two strings $abdc$ and $dacb$. A partial order R shared by them is shown in Fig. 4. Since R is exactly the forbidden subgraph of MSP DAG [24], R is not series parallel. In other words, only using the series parallel orders cannot cover all frequent partial orders.

Moreover, it is shown [24] that any series parallel order is the intersection of two total orders, i.e., the dimensionality of any series parallel order is 2. For partial orders that are frequent in multiple sequences (i.e., they are the intersection of the corresponding total or partial orders), the dimensionality is likely more than 2. In such cases, the frequent partial orders may not be series parallel.

In fact, as discussed in Section 5, we did find nonseries parallel orders that are frequent in real data sets. Fig. 10 presents some examples (except for Fig. 10b). Such nonseries parallel orders cannot be identified by any previous methods.

There are several interesting studies on applications of ordering information. For example, to discover local structures in gene expression data, Ben-Dor et al. [5] looked for local patterns that manifest themselves simultaneously on a subset G of genes and a subset T of experiments. Specifically, they searched for order-preserving submatrices (OPSMs), in which the expression levels of all genes induce the same linear ordering (i.e., total order on a subset) of the experiments. They showed that the OPSM search problem is NP-hard. They defined a probabilistic model in which an OPSM is hidden within an otherwise random matrix. Guided by this model, they developed an efficient algorithm to find hidden OPSMs in a random matrix. Please note that their method cannot find the *complete set* of linear orders. Instead, our methods here find the complete set. In [15], Liu and Wang proposed a sequential pattern mining method to find the complete set of linear orders, i.e., substrings. However, their approach is not concerned with partial orders in general.

3.3 A Two-Step Method

Simultaneously to this study, Casas-Garriga [7] also proposed a similar idea to use closed partial orders to summarize sequential data. A two-step algorithm was proposed. First, the complete set of closed sequential patterns is mined. Then, the combinations of closed sequential patterns are enumerated. For a set of closed sequential patterns $s_1, \ldots, s_n$, let R be a partial order generated as follows: $(x, y) \in R$ if and only if $x \neq y$ and for each sequential pattern $s_i (1 \leq i \leq n)$, x appears before y in s_i. If $R \neq \emptyset$, then a closed partial order is identified. It checks the support of the closed partial orders identified in the previous step and removes all redundant frequent closed partial orders. That is, each frequent closed partial order should be output only once.

However, the above algorithm is far from efficient. First, mining the complete set of closed sequential patterns is nontrivial. Second, there often exist a large number of closed sequential patterns from large string databases. Last, enumerating the combinations of closed sequential patterns and removing redundant frequent closed partial orders in the last step can be very expensive. Only some patterns found from two small data sets are reported in [7]. The two data sets used in the experiments in [7] are small, containing only 1,000 transactions and 607 transactions, respectively. No experimental results on the efficiency and the scalability of the algorithm are shown. As shown in our experimental results, the method is not efficient for large data sets. Therefore, mining frequent partial orders from sequence data remains a challenging problem.

4 Algorithms

In this section, we present two algorithms for frequent closed partial order mining. First, we describe algorithm *TranClose* (for Transitive Closure), which transforms a string database into a database of the transitive closures of the strings, and reduces the problem to mining frequent closed itemsets. However, it has to enlarge the database substantially.

Second, we propose an efficient algorithm, *Frecpo* (for Frequent closed partial order), which mines the string database directly and never computes any transitive closure. Moreover, it identifies the frequent closed partial orders in the form of transitive reduction, which is the minimal representation. It exploits several interesting pruning techniques to speed up the mining process.

4.1 TranClose: A Rudimentary Method

Here, we describe *TranClose*, a method more efficient than the two-step algorithm in [7].

As shown in Section 2.4, the problem of mining frequent closed partial orders can be reduced to mining frequent closed graph patterns from the transitive closure DAGs of the strings. However, mining graph patterns can be very costly, since the bottleneck, many isomorphism tests to determine whether a graph is a subgraph in another graph, can be very expensive [28].

To tackle the problem, we can further reduce the problem to mining frequent closed itemsets. That is, every transitive closure DAG can be uniquely represented as the set of edges in the DAG. Then, mining frequent closed graph patterns in the DAG database can be accomplished by mining frequent closed edge-sets in the transformed transaction database, as illustrated in the following example.

Example 4 (*TranClose*). Consider a string database SDB as shown in the first two columns of Table 2. Suppose the minimum support threshold is 2.

TranClose mines the complete set of frequent closed partial orders in three steps.

In the first step, we expand the strings to their transitive closures. A transitive closure is denoted by the set of edges. The third column of Table 2 shows the transfor-

TABLE 2
String Database SDB as the Running Example

Sid	String	Transitive closure $\mathcal{C}(s)$
1	$abcdef$	$ab, ac, ad, ae, af, bc, bd, be, bf, cd, ce, cf, de, df, ef$
2	$acbde$	$ac, ab, ad, ae, cb, cd, ce, bd, be, de$
3	$dabce$	$da, db, dc, de, ab, ac, ae, bc, be, ce$
4	$dcabe$	$dc, da, db, de, ca, cb, ce, ab, ae, be$

mation. The set of edges in the transitive closure of each string becomes a transaction so a transaction database TDB is created. Please note that the edges are directed. That is, edges bc and cb are different.

In the second step, we mine frequent closed edge-sets from the transformed transaction database TDB *(i.e., the third column in* Table 2) with support threshold $min\_sup = 2$. Each frequent closed edge-set corresponds to the transitive closure of a frequent closed partial order.

In the last step, for each frequent closed edge-set, we compute its transitive reduction. In this example, there are six patterns found. They are shown in Fig. 5 together with their transitive reduction DAGs for examination.

Algorithm *TranClose* is summarized in Fig. 6. We can use any frequent closed itemset mining algorithm such as CHARM [33] and CLOSET+ [27] to mine frequent closed edge-sets from the transformed transaction database, and derive the frequent closed partial orders from the frequent closed edge-sets.

The bottleneck of *TranClose* is that it has to handle a very much enlarged transitive closure database: For a string of length l, its transitive closure has $\frac{l(l-1)}{2}$ edges.

In our small running example, there are totally 41 edges in all the frequent closed edge-sets, while only 27 edges in their transitive reduction DAGs. In other words, more than one-third of the edges in the transitive closures are redundant and will be removed in the transitive reductions. For large string databases where there are long strings, the redundancy may be even bigger. As a well accepted fact, mining long patterns is often very costly. *To improve the efficiency, we have to avoid computing transitive closure in mining frequent closed partial orders*.

4.2 Algorithm Frecpo

In this section, we develop algorithm *Frecpo* which mines frequent closed partial orders in the form of transitive reductions directly from string databases and avoids computing transitive closures.

4.2.1 General Idea and Framework

In order to efficiently mine the complete set of frequent closed partial orders, we have to address the following two issues.

- *The correctness and completeness issue*. We have to find a systematic way to enumerate all the frequent closed partial orders without duplicate. This will guarantee that the mining result is correct and complete.

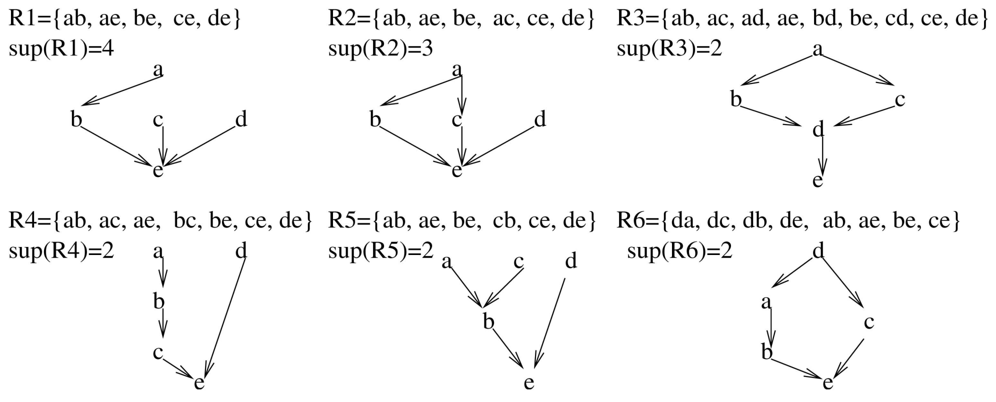

R1={ab, ae, be, ce, de}
sup(R1)=4
R2={ab, ae, be, ac, ce, de}
sup(R2)=3
R3={ab, ac, ad, ae, bd, be, cd, ce, de}
sup(R3)=2
R4={ab, ac, ae, bc, be, ce, de}
sup(R4)=2
R5={ab, ae, be, cb, ce, de}
sup(R5)=2
R6={da, dc, db, de, ab, ae, be, ce}
sup(R6)=2

Fig. 5. The frequent closed itemsets and the transitive reductions of the corresponding frequent CPOs.

Input: a string database SDB and a minimum support threshold min_sup;
Output: the complete set of frequent closed partial orders;
Method:

```
1: create a transaction database $TDB$ by transforming each string in $SDB$ into its transitive
   closure and make the set of edges in the transitive closure a transaction;
2: mine frequent closed edge-sets from $TDB$ with support threshold $min\_sup$;
3: for each frequent closed edge-set, compute its transitive reduction as a frequent closed
   partial order;
```

Fig. 6. The *TranClose* algorithm.

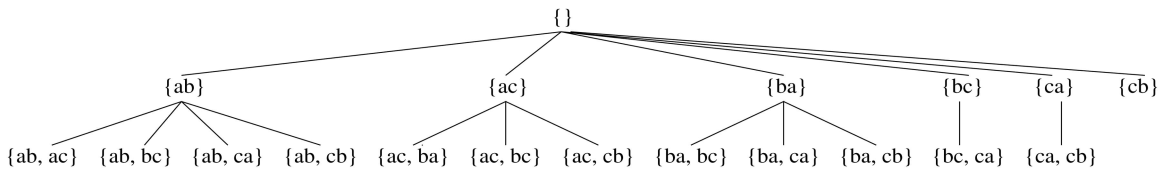


Fig. 7. The set enumeration tree of the transitive reductions of all possible partial orders on items a, b, and c.

- *The efficiency and scalability issue*. We must have an efficient method to extract frequent closed partial orders and prune futile search branches.

To address the correctness and completeness issue, *Frecpo* searches a set enumeration tree of transitive reductions of partial orders in a depth-first manner.

In principle, a partial order can be uniquely represented as the set of edges in its transitive reduction. Moreover, all edges in a set can be sorted in the dictionary order[1] and, thus, it can be written as a list. Therefore, we can enumerate all partial orders in the dictionary order. A set enumeration tree of partial orders can be formed: For orders R_1 and R_2, R_1 is an ancestor of R_2 and R_2 is a descendant of R_1 in the tree if and only if the list of edges in R_1 is a prefix of the list of edges in R_2.

For example, consider a set of items $\{a, b, c\}$. The transitive reductions of all possible partial orders on the three items can be enumerated in a set enumeration tree shown in Fig. 7.

By a depth-first search of the set enumeration tree of transitive reductions of partial orders, *Frecpo* will not miss any frequent partial order. *Frecpo* employs depth-first search instead of breadth-first search because there are many previous studies (e.g., [8], [15], [19], [27], [28], [29], [32], [33]) strongly suggesting that a depth-first search with appropriate pseudoprojection techniques often achieves a better performance than a breadth-first search when mining large databases.

To address the efficiency and scalability issue, *Frecpo* prunes the futile branches and narrows the search space as much as possible. Basically, three types of techniques are used:

- *Pruning infrequent items, edges and partial orders*. According to Property 2.1, if a partial order R in the set enumeration tree is infrequent, then the partial orders in the subtree rooted at R, which are stronger than R, cannot be frequent. The subtree can be pruned. Hence, *Frecpo* often does not have to search the complete set enumeration tree. Instead, only the upper part of the tree which contains all the frequent partial orders is searched. Moreover, only frequent closed partial orders will be output.
- *Pruning forbidden edges*. Not every edge can appear in the transitive reduction of a partial order. For example, if every string containing ac also contains ab and bc, then edge ac should not appear in the transitive reduction of any frequent closed partial order. Edge ac is called a forbidden edge. Removing the forbidden edges can also reduce the search space.
- *Extracting transitive reductions of frequent partial orders directly*. In *Frecpo*, we develop an efficient method to identify frequent closed partial orders and also extract their transitive reductions from various subsets of strings. Thus, *Frecpo* does not need to compute the transitive reductions.

Algorithm *Frecpo* is shown in Fig. 8. In the following subsections, we will explain the technical details.

4.2.2 Pruning in Frecpo

Our first rule of pruning is a corollary of Property 2.1.

Lemma 4.1 (Pruning by support). *An infrequent item or an infrequent edge cannot appear in any frequent partial order.*

The lemma is used in *Frecpo* in two ways. First, at the beginning of the algorithm, the database is scanned so that frequent items and frequent edges are identified. Infrequent items and infrequent edges are pruned. Second, in the recursive depth-first search, for any frequent closed partial order R, only the edges that frequently appear together with R in the string database should be used to expand R to form R's children. Technically, all the strings supporting R form the R-projected database $SDB|_R = \{s \in SDB | R \subseteq \mathcal{C}(s)\}$. Only the frequent edges in the R-projected database and satisfying the requirement of the enumeration tree should be used to expand R to R's children in the set enumeration tree. The items and edges infrequent in the projected database will be removed.

Our second rule of pruning is based on the observation that not every frequent edge can appear in the transitive reduction of a frequent closed partial order. An edge xy is called a *forbidden edge* in a string database SDB if there exists an item z such that for every string s in SDB which contains xy, s also contains xzy. In such a case, for any frequent closed partial order R which contains (x, y), R also contains (x, z) and (z, y), which disqualify (x, y) in R's transitive reduction.

Lemma 4.2 (Pruning forbidden edges). *A forbidden edge cannot appear in the transitive reduction of any frequent closed partial order.*

Frecpo uses a *detection matrix* to identify both frequent edges and forbidden edges, as illustrated in the following example.

Example 5 (*Frecpo*—Part 1). Let us consider again mining frequent closed partial orders from the string database SDB in Table 2 with respect to minimum support threshold $min\_sup = 2$.

1. In fact, any global order on the edges works. For the sake of convenience, we choose dictionary order as an example here.

```
Input: a string database $SDB$ and a minimum support threshold $min\_sup$;
Output: the complete set of frequent CPOs;
Method:
1:  scan database once, find frequent items; // Lemma 4.1
2:  scan database again, find global feasible edges; // Lemmas 4.1 and 4.2
    // if the total number of items in $SDB$ is not large, the first two scans can be combined.
3:  let $R$ be the set of global feasible edges with support $|SDB|$;
4:  if $R \neq \emptyset$ then output $R$ as a frequent CPO; // Lemma 4.3
5:  let $L = e_1, \ldots, e_n$ be the list of global feasible edges with support less than $|SDB|$;
6:  for each edge $e_i$ in $L$ do
7:    if $R \cup \{e_i\}$ does not contain any redundant edge and there exists no FCPO $R'$ found
      before such that $R' \supset (R \cup \{e_i\})$ and $sup(R') = sup(e_i)$ then
8:      form $R \cup \{e_i\}$-projected database $SDB|_{R\cup\{e_i\}}$;
9:      recursively mine $SDB|_{R\cup\{e_i\}}$;
```

Fig. 8. The *Frecpo* algorithm.

By scanning the database only once, *Frecpo* computes the supports of the items. Following Lemma 4.1, infrequent items are pruned, such as f in our running example $(sup(f) = 1)$.

To prune the infrequent edges and the forbidden edges, *Frecpo* scans the database again and fills in a matrix $\{cnt[x, y]\}$, where x and y are both frequent items, and $cnt[x, y]$ registers both $sup(xy)$ and the list of items that appear between x and y in all strings having been scanned so far that contain xy. The list is called the anchor list. The matrix is called the *detection matrix* and is shown in Fig. 9.

From the detection matrix, we can immediately prune the infrequent edges (those with support less than 2, such as ca). An edge is a forbidden edge if its anchor list is not empty. Following Lemma 4.2, forbidden edges ad, ae, and db can be pruned as well.

In this example, SDB contains six different items. There are $6 \times 5 = 30$ possible different edges. Twenty different edges appear in the database. Only 11 edges survive from the pruning.

Clearly, the length of an anchor list monotonically decreases as the scan goes on. If the strings are scanned in an arbitrary order, the initial length of any anchor list for any edge is bounded by the maximum number of frequent items in any string. More often than not, the length of a string is much shorter than the total number of items in the whole database. Moreover, as a heuristic, we can scan the short strings before the long ones. Then, the initial length of any anchor list for any edge xy is bounded by the number of frequent items in the shortest string which contains xy.

If the number of items is not very large and a detection matrix for all items can be held in main memory, *Frecpo* can scan the database only once to prune infrequent items, infrequent edges and forbidden edges by using a detection matrix holding all items instead of only the frequent ones.

	a	b	c	d	e
a		$4, \emptyset$	$3, \emptyset$	$2, \{b, c\}$	$4, \{b\}$
b	$0, \emptyset$		$2, \emptyset$	$2, \emptyset$	$4, \emptyset$
c	$1, \emptyset$	$2, \emptyset$		$2, \emptyset$	$4, \emptyset$
d	$2, \emptyset$	$2, \{a\}$	$2, \emptyset$		$4, \emptyset$
e	$0, \emptyset$	$0, \emptyset$	$0, \emptyset$	$0, \emptyset$	

Fig. 9. The matrix detecting infrequent edges and edges not in transitive reduction.

4.2.3 Extracting Frequent Closed Partial Orders

Example 6 (*Frecpo*—Part 2). As shown in Example 5, only the edges ab, ac, bc, bd, be, cb, cd, ce, da, dc, and de can be used to construct the transitive reduction of frequent closed partial orders. They are called the *global feasible edges*. Among them, ab, be, ce, and de have support 4, i.e., they appear in every string in SDB. The four edges form a frequent closed partial order, i.e., order R_1 in Fig. 5. In other words, the set of global feasible edges that appear in every string forms a frequent CPO. Interestingly, the set is in fact the transitive reduction, since any redundant edge in the set is identified as a forbidden edge by the detection matrix.

The observation in Example 6 leads to the following.

Lemma 4.3 (EXTRACTING TRANSITIVE REDUCTION OF FCPO). *In a string database SDB, the set of global feasible edges that have support $|SDB|$ is the transitive reduction of the frequent closed partial order R of support $|SDB|$.*

Proof. There exists only one FCPO whose support is $|SDB|$. Otherwise, if there are two FCPOs R_1 and R_2 whose support are $|SDB|$, both R_1 and R_2 $(R_1 \neq R_2)$ are supported by every string in the database. That means $R_1 \cup R_2$ is supported by every string in the database and thus is also a frequent partial order with support $|SDB|$. That leads to a contradiction to the assumption that R_1 and R_2 are closed.

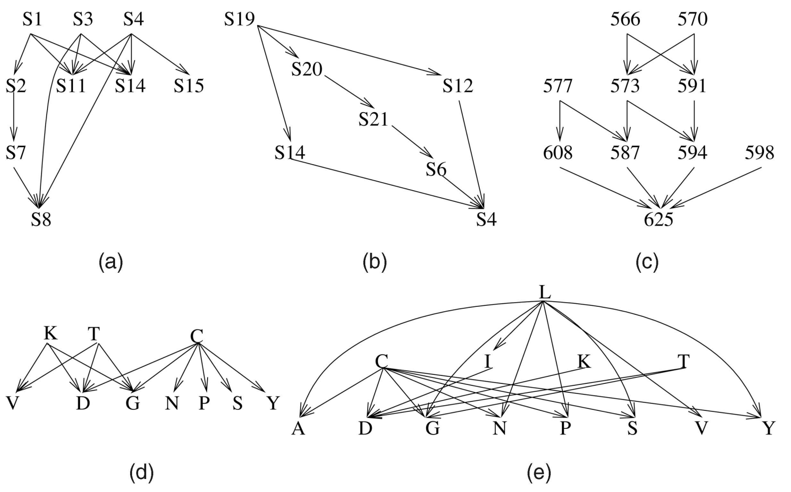


Fig. 10. Some frequent closed partial orders found in real data sets. (a) A pattern in data set Yeast (support=80). (b) A pattern in data set BreastCancer (support=224). (c) A pattern in data set Gazelle (support=10). (d) A pattern in data set Snake (support=107). (e) Another pattern in data set Snake (support=80).

We denote the FCPO with support $|SDB|$ by R. The set specified in the lemma is a superset of the transitive reduction of R. On the other hand, if there exists a redundant edge, the edge will be identified by the detection matrix. Hence, the set is exactly the transitive reduction of R. □

Lemma 4.3 enables *Frecpo* to identify the transitive reduction of frequent partial orders directly. In other words, *Frecpo* prunes redundant edges using a detection matrix. It never has to explicitly compute transitive reduction for any frequent closed partial order.

4.2.4 Recursively Depth-First Searching

Once a frequent closed partial order R is found, *Frecpo* expands R to its children. Following the similar reasoning in Lemmas 4.1, 4.2 and 4.3, only frequent, nonforbidden edges in the $R\text{-projected}$ database should be used to expand R to its children in the enumeration tree.

Example 7 (*Frecpo*—Part 3). Let us continue the mining process in Example 6. Frequent closed partial order R_1 is the order shared by all strings. Thus, any other frequent closed partial order will be stronger than R_1.

The other frequent closed partial orders in transitive reduction can be partitioned into the following subsets according to the dictionary order of the remaining global feasible edges (i.e., ac, bc, bd, cb, cd, da, and dc): 1) the ones having edge ac in their transitive reduction, 2) the ones having edge bc but no ac in their transitive reduction, etc., and 7) the one having dc but no other edges in its transitive reduction (if it is a frequent closed partial order). These subsets can be mined one by one in a depth-first search manner.

We first consider the subset of frequent closed partial orders having edge ac in their transitive reductions. They also contain R_1. The strings in SDB that are superstrings of ac, namely, strings 1, 2, and 3, are collected as the $(R_1 \cup \{ac\})\text{-projected}$ database.

We prune the local infrequent items, infrequent edges and forbidden edges by scanning the $(R_1 \cup \{ac\})\text{-projected}$ database once and filling in the local detection matrix. The feasible edges in this projected database are bc, bd, and cd. Since each feasible edge has support 2, which is less than the number of strings in the projected database, we extract $R_2 = R_1 \cup \{ac\}$ as the transitive reduction of a frequent closed partial order as shown in Fig. 5. Any frequent partial order having ac must be stronger than R_2.

Since we have three local feasible edges in the $(R_1 \cup \{ac\})\text{-projected}$ database, the remaining frequent closed partial orders having ac in their transitive reduction can be further partitioned into three subsubsets: the ones having ac and bc, the ones having ac and bd but no bc, and the ones having ac and cd but no bc nor bd.

R_2 has an edge ab, and any frequent partial order having ac is a superset of R_2. Clearly, edges ab, ac, and bc cannot stay together in a transitive reduction, since ac is redundant in such a case. Thus, we immediately determine that the first subsubset is empty without checking the database at all.

The remaining frequent closed partial orders can be found recursively.

4.2.5 Summary

In implementation, we use the pseudoprojection technique which was first proposed in [19] and later has become popular in depth-first search frequent pattern mining. That is, if a database or a projected database can fit into main memory, instead of deriving a copy of strings for every projected database, we use hyperlinks (implemented as pointers) to link the strings in the projected database together. The recursive projected databases can share the same physical database storage. Scanning and deriving projected databases are efficient with the help of hyperlinks. As discussed in [19], [26], if a database is large and cannot fit into main memory, the physical projections should be generated. Once a projected database can be held into main memory, the recursion is switched to pseudoprojection.

The correctness of algorithm *Frecpo* can be justified based on our previous discussion. Comparing to algorithm

TranClose and other rudimentary methods, *Frecpo* has three distinct advantages.

Advantage 1: Mining in transitive reduction to avoid substantial space and I/O overhead. *Frecpo* never explicitly unfolds strings into transitive closures. As discussed before, the transitive closures of strings can be much larger than the strings themselves. Thus, mining the strings directly avoids the substantial space overhead and also the I/O cost. *Frecpo* does examine combinations of items for each string. However, such tests are conducted on the fly in main memory. It does not involve any space or I/O overhead, which is the bottleneck of mining large databases.

Advantage 2: Directly extracting frequent closed partial orders in transitive reduction. *Frecpo* computes detection matrices and extracts frequent closed partial orders in transitive reduction directly (Lemma 4.3). It avoids the postprocessing of computing transitive reductions. Transitive reduction is the minimum representation of a partial order. Using this minimum representation makes the mining more effective and efficient.

Advantage 3: Aggressively and progressively pruning futile branches in recursive depth-first search. *Frecpo* aggressively prunes infrequent items and edges and forbidden edges that are impossible to appear in transitive reductions of frequent closed partial orders. Thus, the search space shrinks dramatically in the recursive depth-first search. Moreover, only frequent items and local feasible edges in the current projected database will be used to expand the current frequent closed partial order into stronger ones. This pattern-growth approach makes the search more focused.

5 Experimental Results

In this section, we report a systematic performance study on both real data sets and synthetic data sets. The experiments were conducted on an IBM T41 laptop computer with a Pentium M 1.4 G CPU and 512 M main memory, running Microsoft Windows XP operating system. We implemented algorithms *TranClose* and *Frecpo* in C++. We also compared with *BIDE* [26], the so far fastest closed sequential pattern method which can serve as the first step of the two-step method in [7]. The original executable from the authors of [26] was used. We used both real data sets and synthetic data sets to test the algorithms. Here, as representative results, we only report the results on real data sets *Yeast*, *BreastCancer*, *Snake*, *Connect-4*, and *Gazelle*, and synthetic data sets generated by the IBM sequence data generator [3]. In all our experiments, the data sets can be held into main memory.

5.1 Interesting FCPOs in Real Data Sets

To test the effectiveness of mining frequent closed partial orders, we first show some interesting patterns from real data sets *Yeast*, *BreastCancer*, *Snake*, and *Gazelle*.

Data sets *Yeast* and *BreastCancer* are microarray data sets. Data set *Yeast* contains the expression levels of 2,884 genes under 17 conditions. Data set *BreastCancer* contains 3,226 genes under 22 conditions. In these two data sets, we sorted the samples in each row (i.e., per gene) in expression ascending order, as suggested in [5]. The *Snake* data set contains 175 Toxin-Snake protein sequences and 20 unique items. It records a family of eukaryotic and viral DNA binding proteins. This data set is also used in [26] for evaluation of frequent closed sequential pattern mining. Each record is a string. The average string length is 67 and the maximal string length is 121. As preprocessing, we removed the duplicate items in a string and only kept the first occurrences. That is, an item appears at most once in a string after the preprocessing. *Gazelle* contains 29,369 Web click-stream sequences from customers. These data sets have been popularly used as benchmarks for the performance of frequent pattern mining in previous studies.

Some interesting frequent closed partial orders in those real data sets are shown in Fig. 10. Some of those patterns cannot be found by any previous frequent pattern mining methods, since they may be hidden in multiple frequent itemsets or sequential patterns and thus cannot be identified in whole. For example, the pattern in Fig. 10d is also very interesting from the graph theory point of view. First, it is a bipartite graph. Moreover, it contains forbidden subgraphs of MSP DAG [24]. The subgraph induced on vertex set $\{C, G, N, T\}$ is such an instance, while there are several other instances. Therefore, this pattern is not series parallel and, thus, cannot be found by the previous methods on mining (frequent) episodes [17], [16], [11]. In fact, the patterns in Figs. 10a, 10c, and 10e are not series parallel, either.

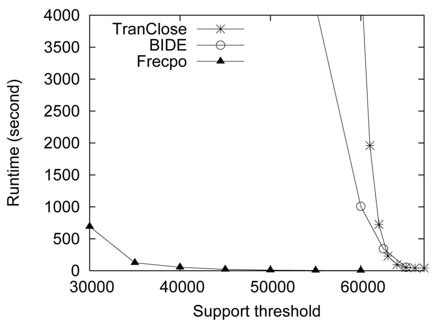
5.2 Mining Long Strings in Connect-4

The *Connect-4* data set contains all legal 8-ply positions in the game of Connect Four in which neither player has won yet, and in which the next move is not forced. The data set has 67,557 strings, and each string has 42 nominal items. Each item natively appears in each string at most once. Thus, it does not need any preprocessing. *Connect-4* is a well-known dense data set and has been used extensively in the studies of mining frequent (closed) itemsets and frequent (closed) sequential patterns. We used this data set to test the scalability of the algorithms on mining long strings.

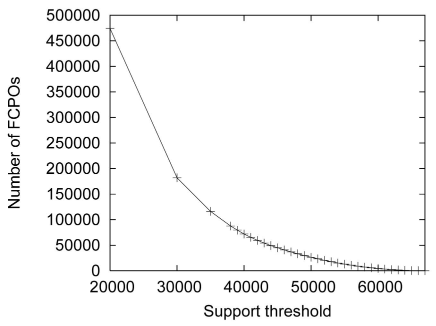
Fig. 11a shows the runtime of *TranClose*, *BIDE*, and *Frecpo* with respect to the minimum support threshold. *BIDE*, the so far fastest closed sequential mining method, can serve as the first step of the two-step method in [7]. When the minimum support threshold goes down, the number of FCPOs increases substantially, as shown in Fig. 11b. Thus, the runtime of all methods increases accordingly. However, *Frecpo* is clearly more scalable than the other two, since *Frecpo* does not unfold the strings onto their transitive closures. Instead, it keeps the transitive reductions of the FCPOs and exploits effective techniques to prune unfruitful branches. This set of experiments clearly show that *Frecpo* is substantially more efficient and more scalable than *TranClose* and the two-step method in [7].

5.3 Results on Synthetic Data Sets: Scalability

To further test the scalability of the algorithms, we used the renowned IBM sequence data generator to generate synthetic data sets. Preprocessing is used to enforce that each item appears only once in a string. The IBM data generator takes quite a few parameters in generating synthetic data sets. Here, we took most of the default values for the command options provided by the data generator and focused on testing the effect of the following two factors: the average length of the strings in the database and the number of strings in the database.

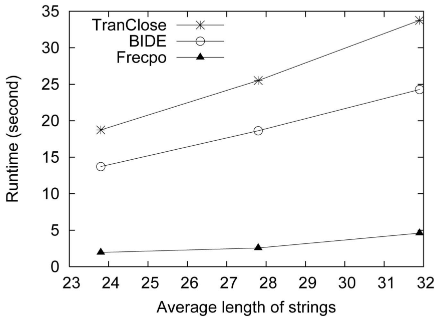


(a)

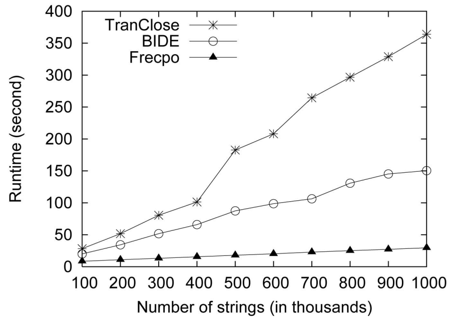


(b)

Fig. 11. The results on real data set *Connect-4*. (a) Runtime versus support threshold. (b) Number of FCPOs.



(a)



(b)

Fig. 12. The results on systhetic data sets. (a) Scalabilty with respect to average length. (b) Scalability with respect to database size.

Fig. 12a shows the scalability on average length of strings in the database. Here, we fixed the number of strings to 600,000, the number of items in the database to 500, and the support threshold to 2,000 (i.e., 0.33 percent), and varied the average length of the strings. As can be seen, *TranClose*, *BIDE* and *Frecpo* are scalable, but *Frecpo* is clearly the most efficient. As the average length increases, the number of FCPOs and their size also increase. Maintaining the transitive closures for large FCPOs is costly.

As the last experiment, we tested the scalability of the mining algorithms on the database size up to 1 million strings. The number of items in the database was set to 500 and the average number of items per string was 15.59. We fixed the support threshold to 1 percent. The results are shown in Fig. 12b. As can be seen, the three algorithms are linearly scalable, and *Frecpo* is consistently the most efficient.

6 Discussion

In this section, we briefly discuss two possible extensions of the *Frecpo* algorithm to mine other types of data and patterns.

6.1 Mining Ordering Databases

In this paper, we focus on strings. A string specifies a total order on a subset of items. Interestingly, algorithm *Frecpo* can be extended to mine *ordering databases*, where each entry is a partial order on a subset of items.

Conceptually, algorithm *Frecpo* can be applied on ordering databases. However, comparing to mining string databases, mining ordering databases incurs a major cost to determine whether an edge is contained in a partial order in the database. This is a common operation in determining global feasible edges.

While a partial order can be represented as its transitive reduction, searching the graphs directly can be costly. Here, we propose an easy way to improve the efficiency. For each vertex in the transitive reduction of a partial order, we can assign a *level number* as follows: 1) each source vertex (i.e., such an item does not have any ancestor in the order) has level 0 and 2) the level number of each other vertex is the length of the longest path from a source to the vertex. (a, b) is not contained in partial order R if the level number of a is not smaller than that of b in R.

As preprocessing, we can compute the level numbers for every partial order in the database. Then, to determine whether (a, b) is an edge in R, we first check the level numbers of a and b. Only when a has a smaller level number than b will we search the graph for edge (a, b). In other words, we can use the level numbers to filter out many impossible edges early.

6.2 Mining Emerging Order Patterns

In Section 1, we mention that, to support misuse detection, we can mine partial orders that are frequent in the subset of intrusions and are rare in the subset of normal activities. Although we can apply *Frecpo* on the subsets of intrusions and normal activities, respectively, and compare the frequent closed partial orders found, it is more efficient to mine the two sets jointly.

We can borrow the idea of emerging patterns [9]. In the database, each string carries a label: either intrusion or normal. Instead of mining all frequent closed partial orders, we monitor the class distribution in the projected databases. We only output a frequent closed partial order if most activities in its projected database are normal. If a projected database is relatively pure, i.e., either there are very few intrusions or there are very few normal activities, we can prune the recursive mining since the current order already distinguishes the two subsets.

7 Conclusions

In this paper, we studied a novel problem of mining frequent closed partial orders from strings. We showed that the problem is interesting and has broad applications, such as in bioinformatics, network management, and intrusion detection. However, our theoretical analysis showed that the problem is challenging. We developed an efficient algorithm, *Frecpo*, to tackle the problem. *Frecpo* mines the string database directly without explicitly unfolding the strings onto their transitive closures. Moreover, *Frecpo* directly extracts the frequent closed partial orders in transitive reduction and prunes the search space aggressively and progressively by depth-first search.

As future work, we will investigate several interesting problems inspired by this study. First, it is important to exploit the frequent closed partial order mining techniques in various applications, such as clustering and classification, and tackle the problems of constraint-based or preference-based mining. Second, how to extend the frequent pattern mining techniques in general to mine other kinds of advanced frequent patterns for emerging applications is always an attractive topic.

Acknowledgments

The authors thank Dr. Bongki Moon, the associate editor, and the anonymous reviewers for their insightful and constructive comments, especially their suggestions on the complexity analysis and the pointers to related work on workflow reconstruction. The research of Jian Pei and Jian Liu is supported in part by the NSERC Grants 312194-05 and 614067, and US National Science Foundation Grant IIS-0308001. The research of Ke Wang is supported in part by NSERC grants. The research of Jianyong Wang is supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 60573061. All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," *EDBT '98: Proc. Sixth Int'l Conf. Extending Database Technology,* pp. 469-483, 1998.

[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. 1993 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '93),* pp. 207-216, May 1993.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 1995 Int'l Conf. Data Eng. (ICDE '95),* pp. 3-14, Mar. 1995.

[4] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential Pattern Mining Using a Bitmap Representation," *Proc. 2002 ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02),* pp. 429-435, July 2002.

[5] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, "Discovering Local Structure in Gene Expression Data: The Order-Preserving Submatrix Problem," *Proc. Sixth Ann. Int'l Conf. Computational Biology,* pp. 49-57, 2002.

[6] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino, "On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets," *Proc. Symp. Theoretical Aspects of Computer Science,* pp. 133-141, 2002.

[7] G. Casas-Garriga, "Summarizing Sequential Data with Closed Partial Orders," *Proc. 2005 SIAM Int'l Conf. Data Mining,* Apr. 2005.

[8] D.Y. Chiu, Y.H. Wu, and A.L.P. Chen, "An Efficient Algorithm for Mining Frequent Sequences by a New Strategy without Support Counting," *Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04),* pp. 275-286, 2004.

[9] G. Dong and J. Li, "Efficient Mining of Emerging Patterns: Discovering Trends and Differences," *Proc. 1999 Int'l Conf. Knowledge Discovery and Data Mining (KDD '99),* pp. 43-52, Aug. 1999.

[10] M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints," *Proc. 1999 Int'l Conf. Very Large Data Bases (VLDB '99),* pp. 223-234, Sept. 1999.

[11] A. Gionis, T. Kujala, and H. Mannila, "Fragments of Order," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 129-136, 2003.

[12] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R.S. Sharma, "Discovering All Most Specific Sentences," *ACM Trans. Database Systems,* vol., 28, no. 2, pp. 140-174, 2003.

[13] A. Inokuchi, T. Washio, and H. Motoda, "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data," *Proc. 2000 European Symp. Principle of Data Mining and Knowledge Discovery (PKDD '00),* pp. 13-23, Sept. 2000.

[14] H.C.M. Kum, J. Pei, and W. Wang, "Approxmap: Approximate Mining of Consensus Sequential Patterns," *Proc. 2003 SIAM Int'l Conf. Data Mining,* May 2003.

[15] J. Liu and W. Wang, "Op-Cluster: Clustering by Tendency in High Dimensional Space," *Proc. Third IEEE Int'l Conf. Data Mining (ICDM '03),* Nov. 2003.

[16] H. Mannila and C. Meek, "Global Partial Orders from Sequential Data," *Proc. 2000 ACM SIGKDD Int'l Conf. Knowledge Discovery in Databases (KDD '00),* pp. 150-160, Aug. 2000.

[17] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery,* vol. 1, pp. 259-289, 1997.

[18] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," *Proc. Seventh Int'l Conf. Database Theory (ICDT '99),* pp. 398-416, Jan. 1999.

[19] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," *Proc. 2001 Int'l Conf. Data Eng. (ICDE '01),* pp. 215-224, Apr. 2001.

[20] J. Pei, J. Han, and W. Wang, "Constraint-Based Sequential Pattern Mining in Large Databases," *Proc. 2002 Int'l Conf. Information and Knowledge Management (CIKM '02),* Nov. 2002.

[21] J. Pei, J. Liu, H. Wang, K. Wang, P.S. Yu, and J. Wang, "Efficiently Mining Frequent Closed Partial Orders," *Proc. Fifth IEEE Int'l Conf. Data Mining (ICDM '05),* pp. 753-756, IEEE, Nov. 2005

[22] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96),* pp. 3-17, Mar. 1996.

[23] P. Tzvetkov, X. Yan, and J. Han, "TSP: Mining Top-k Closed Sequential Patterns," *Proc. Third IEEE Int'l Conf. Data Mining (ICDM '03),* Nov. 2003.

[24] J. Valdes, R.E. Tarjan, and E.L. Lawler, "The Recognition of Series Parallel Digraphs," *Proc. 11th Ann. ACM Symp. Theory of Computing,* pp. 1-12, 1979.

[25] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, pp. 1128-1142, Sept. 2004.

[26] J. Wang and J. Han, "BIDE: Efficient Mining of Frequent Closed Sequences," *Proc. 20th IEEE Int'l Conf. Data Eng.,* pp. 79-90, 2004.

[27] J. Wang, J. Han, and J. Pei, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '03),* 2003.

[28] X. Yan and J. Han, "Closegraph: Mining Closed Frequent Graph Patterns," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '03),* 2003.
[29] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining Closed Sequential Patterns in Large Databases," *Proc. 2003 SIAM Int'l Conf. Data Mining,* May 2003.
[30] G. Yang, "The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns," *Proc. 10th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '04),* 2004.
[31] J. Yang, P.S. Yu, W. Wang, and J. Han, "Mining Long Sequential Patterns in a Noisy Environment," *Proc. 2002 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '02),* Jun. 2002.
[32] M.J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning,* vol. 42, nos. 1-2, pp. 31-60, 2001.
[33] M.J. Zaki and C.J. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemset Mining," *Proc. 2002 SIAM Int'l Conf. Data Mining,* pp. 457-473, Apr., 2002.

Jian Pei received the PhD degree in computing science from Simon Fraser University, Canada, in 2002. He is currently an assistant professor of computing science at Simon Fraser University, Canada. His research interests can be summarized as developing effective and efficient data analysis techniques for novel data intensive applications. Particularly, he is currently interested in various techniques of data mining, data warehousing, online analytical processing, and database systems, as well as their applications in bioinformatics, privacy preservation, and education. His current research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the US National Science Foundation (NSF). He has published prolifically in refereed journals, conferences, and workshops, has served extensively in the organization committees and the program committees of many international conferences and workshops, and has been a reviewer for the leading academic journals in his fields. He is a member of the ACM, the ACM SIGMOD, and the ACM SIGKDD.

Haixun Wang received the BS and MS degrees, both in computer science, from Shanghai Jiao Tong University in 1994 and 1996, respectively, and the PhD degree in computer science from the University of California, Los Angeles in 2000. He is currently a research staff member at the IBM T.J. Watson Research Center. He has published more than 60 research papers in referred international journals and conference proceedings. He is a member of the ACM, the ACM SIGMOD, the ACM SIGKDD, and the IEEE Computer Society. He has served in program committees of international conferences and workshops, and has been a reviewer for some leading academic journals in the database field.

Jian Liu received the BS and MS degrees in computer science from Peking University, China, in 2000 and 2003, and the MS degree in computer science from the State University of New York at Buffalo in 2005. He is currently a software engineer at Efficient Frontier Inc. based in Mountain View, California. He has been working in the area of data mining since 2003. His research interests include data mining, data warehousing, and databases.

Ke Wang received the PhD degree from the Georgia Institute of Technology. He is currently a professor in the School of Computing Science, Simon Fraser University. Before joining Simon Fraser, he was an associate professor at National University of Singapore. He has taught in the areas of database and data mining. Professor Wang's research interests include database technology, data mining and knowledge discovery, machine learning, and emerging applications, with recent interests focusing on the end use of data mining. This includes explicitly modeling the business goal (such as profit mining, bio-mining and Web mining) and exploiting user prior knowledge (such as extracting unexpected patterns and actionable knowledge). He is interested in combining the strengths of various fields such as database, statistics, machine learning and optimization to provide actionable solutions to real-life problems.

Jianyong Wang received the PhD degree in computer science in 1999 from the Institute of Computing Technology, the Chinese Academy of Sciences. Since then, he has worked as an assistant professor in the Department of Computer Science and Technology, Peking University, in the areas of distributed systems and Web search engines, and visited the School of Computing Science at Simon Fraser University, the Department of Computer Science at the University of Illinois at Urbana-Champaign, and the Digital Technology Center and Department of Computer Science and Engineering at the University of Minnesota, mainly working in the area of data mining. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is a member of the IEEE Computer Society and the ACM SIGKDD.

Philip S. Yu received the BS degree in electrical engineering from National Taiwan University, the MS and PhD degrees in electrical engineering from Stanford University, and the MBA degree from New York University. He is with the IBM T.J. Watson Research Center and currently manager of the Software Tools and Techniques group. His research interests include data mining, Internet applications and technologies, database systems, multimedia systems, parallel and distributed processing, and performance modeling. Dr. Yu has published more than 450 papers in refereed journals and conferences. He holds or has applied for more than 250 US patents. He is a fellow of the ACM and a fellow of the IEEE. He is an associate editor of the *ACM Transactions on the Internet Technology* and the *ACM Transactions on Knowledge Discovery in Data*. He is a member of the IEEE Data Engineering steering committee and is also on the steering committee of the IEEE Conference on Data Mining. He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* (2001-2004), an editor, advisory board member, and also a guest co-editor of the special issue on mining of databases. He had also served as an associate editor of *Knowledge and Information Systems*. In addition to serving as a program committee member on various conferences, he will be serving as the general chair of the 2006 ACM Conference on Information and Knowledge Management and the program chair of the 2006 joint conferences of the Eighth IEEE Conference on E-Commerce Technology (CEC '06) and the Third IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE' 06). He was the program chair, cochair, general chair, and general cochair of several conferences and workshops. He has received several IBM honors including two IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, two Research Division Awards, and the 86th plateau of Invention Achievement Awards. He received an Research Contributions Award from the IEEE International Conference on Data Mining in 2003 and also an IEEE Region 1 Award for "promoting and perpetuating numerous new electrical engineering concepts" in 1999. Dr. Yu is an IBM Master Inventor.