Available online at www.sciencedirect.com

ScienceDirect

Data & Knowledge Engineering 60 (2007) 5–29

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

On compressing frequent patterns ☆

Dong Xin, Jiawei Han *, Xifeng Yan, Hong Cheng

Department of Computer Science, University of Illinois at Urbana-Champaign, Rm 2132, Siebel Center for Computer Science, 201 N. Goodwin Avenue, Urbana, IL 61801, USA

Available online 3 March 2006

Abstract

A major challenge in frequent-pattern mining is the sheer size of its mining results. To compress the frequent patterns, we propose to cluster frequent patterns with a tightness measure δ (called δ-cluster), and select a *representative pattern* for each cluster. The problem of finding a minimum set of representative patterns is shown NP-Hard. We develop two greedy methods, RPglobal and RPlocal. The former has the guaranteed compression bound but higher computational complexity. The latter sacrifices the theoretical bounds but is far more efficient. Our performance study shows that the compression quality using RPlocal is very close to RPglobal, and both can reduce the number of closed frequent patterns by almost two orders of magnitude. Furthermore, RPlocal mines even faster than FPClose [G. Grahne, J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, in: Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)], a very fast closed frequent-pattern mining method. We also show that RPglobal and RPlocal can be combined together to balance the quality and efficiency.

Keywords: Data mining; Frequent pattern mining

1. Introduction

Frequent-pattern (or itemsets) mining has been a focused research theme in data mining due to its broad applications in mining association [2], correlation [5], causality [18], sequential patterns [3], episodes [14], multi-dimensional patterns [13], max-patterns [9], partial periodicity [11], and many other important data mining tasks.

The problem of frequent-itemsets mining can be defined as follows. Given a transaction database, let $\mathcal{O} = \{o_1, o_2, \ldots, o_d\}$ be the set of items that appear in the database, $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ be the transaction set, and $I(t_i) \subseteq \mathcal{O}$ be the set of items in transaction t_i. For any itemset P, let $T(P) = \{t \in \mathcal{T} | P \subseteq I(t)\}$ be the corresponding set of transactions. We say P is the *expression* of pattern P, and $|T(P)|$ is the *support* of

☆ The work was supported in part by the US National Science Foundation NSF IIS-03-08215/05-13678. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

* Corresponding author. Fax: +1 217 265 6494.
E-mail address: hanj@cs.uiuc.edu (J. Han).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.01.006

pattern P. An itemset P is *frequent* if $|T(P)| \geqslant min\_sup$, where *min_sup* is a user-specified threshold. The task of frequent-itemsets mining is to *find all the frequent itemsets*. Several extensions have been made to the original frequent itemsets problem. A frequent itemset P is *closed* if there is no itemset P' such that $P \subset P'$ and $T(P) = T(P')$, a frequent itemset P is *maximal* if there is no frequent itemset P' such that $P \subset P'$.

There have been many scalable methods developed for frequent-pattern mining [12]. However, the real challenge in frequent-pattern mining is the sheer size of its mining results. In many cases, a high *min_sup* threshold may discover only commonsense patterns but a low one may generate an explosive number of output patterns, which severely restricts its usage. To solve this problem, it is natural to explore how to "*compress*" the patterns, i.e., find a concise and succinct representation that describes the whole collection of patterns.

Two major approaches have been developed in this direction: lossless compression and lossy approximation. The former is represented by the *closed frequent itemsets* [16] and *non-derivable frequent itemsets* [6]. Their compression is lossless in the sense that the complete set of original frequent patterns can be recovered. However, the methods emphasize too much on the *supports* of patterns so that its compression power is quite limited. The latter is represented by the *maximal frequent itemsets* [9], as well as *boundary cover sets* proposed recently [1]. These methods only consider the *expressions* of patterns, while the *support* information in most of the itemsets is lost.

To achieve high-quality pattern compression, it is desirable to build up a pattern compression framework that concerns both the *expressions* and *supports* of the patterns. A motivation example is shown as follows.

Example 1. Table 1 shows a subset of frequent itemsets on *accidents* data set [8], where 39, 38, 16, 18, 12, 17 are the names of individual items. The closed itemsets cannot get any compression on this subset. The maximal itemsets will only report the itemset P_3. However, we observe that itemsets P_2, P_3 and P_4 are significantly different w.r.t. their supports, and the maximal itemset totally loses this information. On the other hand, the two pairs (P_1, P_2) and (P_4, P_5) are very similar w.r.t. both expressions and supports. We suggest a high-quality compression as P_2, P_3 and P_4.

A general proposal for high-quality compression is to cluster frequent patterns according to certain similarity measure, and then select and output only a *representative pattern* for each cluster. However, there are three crucial problems that need to be addressed: (1) how to measure the similarity of the patterns, (2) how to define quality guaranteed clusters where there is a representative pattern best describing the whole cluster, and (3) how to efficiently discover these clusters (and hence the representative patterns)? This paper addresses these problems.

First, we propose a distance measure between two frequent patterns, and show it is a valid distance metric. Second, we define a clustering criterion, with which, the distance between the representative pattern and every other pattern in the cluster is bounded by a threshold δ. The objective of the clustering is to minimize the number of clusters (hence the number of representative patterns). Finally, we show the problem is equivalent to set-covering problem, and it is NP-hard w.r.t. the number of the frequent patterns to be compressed. We propose two greedy algorithms: RPglobal and RPlocal. The former has bounded compression quality but higher computational complexity; whereas the latter sacrifices the theoretical bound but is far more efficient. Our performance study shows that the quality of the compression using RPlocal is very close to RPglobal, and both can reduce the number of patterns generated by about two orders of magnitude w.r.t. the original collection of

Table 1
A subset of frequent itemsets

Pattern ID	Itemsets	Support
P_1	$\{38, 16, 18, 12\}$	205227
P_2	$\{38, 16, 18, 12, 17\}$	205211
P_3	$\{39, 38, 16, 18, 12, 17\}$	101758
P_4	$\{39, 16, 18, 12, 17\}$	161563
P_5	$\{39, 16, 18, 12\}$	161576

closed patterns. Moreover, RPlocal directly mines representative patterns from database and runs even faster than FPClose [10], a fast closed frequent-itemset mining algorithm. We also show that RPglobal and RPlocal can be integrated together to balance the quality and efficiency.

The remainder of the paper is organized as follows. In Section 2, we formally introduce the problem. The NP-hardness is proved in Section 3. Section 4 proposes the RPglobal and RPlocal methods. Our performance study is presented in Section 5. Section 6 discusses the related work. The potential extensions is presented in Section 7, and we conclude the study in Section 8.

2. Problem statement

In this section, we first introduce a new distance measure on closed frequent patterns, and then discuss the clustering criterion.

2.1. Distance measure

Definition 1 (*Distance measure*). Let P_1 and P_2 be two closed patterns. The *distance* of P_1 and P_2 is defined as

$$D(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}$$

Example 2. Let P_1 and P_2 be two patterns: $T(P_1) = \{t_1, t_2, t_3, t_4, t_5\}$ and $T(P_2) = \{t_1, t_2, t_3, t_4, t_6\}$, where t_i is a transaction in the database. The distance between P_1 and P_2 is $D(P_1, P_2) = 1 - \frac{4}{6} = \frac{1}{3}$.

Theorem 1. *The distance measure D is a valid distance metric, such that*:

(1) $D(P_1, P_2) > 0, \forall P_1 \neq P_2$
(2) $D(P_1, P_2) = 0, \forall P_1 = P_2$
(3) $D(P_1, P_2) = D(P_2, P_1)$
(4) $D(P_1, P_2) + D(P_2, P_3) \geqslant D(P_1, P_3), \forall P_1, P_2, P_3$

Proof. It is easy to verify that the first three properties are true. We prove the fourth statement is true.

To simplify the presentation, we define the following variables: $|T(P_1)| = a$, $|T(P_2)| = b$, $|T(P_3)| = c$, $|T(P_1) \cap T(P_2)| = b_1$, $|T(P_2) - T(P_1) \cap T(P_2)| = b_2$, $|T(P_1) \cap T(P_3)| = c_1$, $|T(P_3) - T(P_1) \cap T(P_3)| = c_2$, $|T(P_1) \cap T(P_2) \cap T(P_3)| = d_1$, and $|T(P_2) \cap T(P_3) - T(P_1) \cap T(P_2) \cap T(P_3)| = d_2$. The meanings of the variables are shown in Fig. 1.

Since $(T(P_1) \cap T(P_2)) \cup (T(P_1) \cap T(P_3)) \subseteq T(P_1)$, we have

$$|T(P_1) \cap T(P_2)| + |T(P_1) \cap T(P_3)| - |T(P_1) \cap T(P_2) \cap T(P_3)| \leqslant |T(P_1)| \Rightarrow b_1 + c_1 - d_1 \leqslant a \tag{1}$$

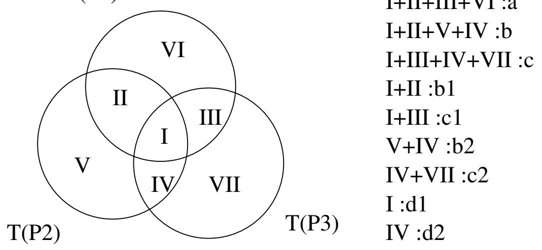


Fig. 1. Meanings of variables.

Plug in all the variables into the distance definition,

$$D(P_1,P_2)+D(P_2,P_3)\geqslant D(P_1,P_3)\iff \frac{b_1}{a+b_2}+\frac{c_1}{a+c_2}\leqslant 1+\frac{d_1+d_2}{b_1+b_2+c_1+c_2-d_1-d_2}$$

Using Eq. (1), we have:

$$\begin{aligned}
&1+\frac{d_1+d_2}{b_1+b_2+c_1+c_2-d_1-d_2}\\
&\quad\geqslant 1+\frac{d_1}{b_1+b_2+c_1+c_2-d_1}\quad (d_2\geqslant 0)\\
&\quad\geqslant 1+\frac{d_1}{a+b_2+c_2}=\frac{a+d_1+b_2+c_2}{a+b_2+c_2}\quad (\text{Eq. }(1))\\
&\quad\geqslant \frac{b_1+c_1+b_2+c_2}{a+b_2+c_2}=\frac{b_1+c_2}{a+b_2+c_2}+\frac{c_1+b_2}{a+b_2+c_2}\quad (\text{Eq. }(1))\\
&\quad\geqslant \frac{b_1}{a+b_2}+\frac{c_1}{a+c_2}\quad (a+b_2\geqslant b_1, c_2\geqslant 0, a+c_2\geqslant c_1, b_2\geqslant 0)
\end{aligned}$$

Thus the fourth statement is true. □

Remark. The distance measure can be extended to the general frequent patterns except that for non-closed patterns, we may have $D(P_1,P_2)=0$ for some $P_1\neq P_2$. This happens when P_1 and P_2 share the same support transactions set.

2.2. Clustering criterion

By defining the distance on the set of transactions, the *support* information of patterns are well incorporated. We further consider the *expressions* of the patterns. Given two patterns A and B, we say B can be *expressed* by A if $O(B)\subset O(A)$. Following this definition, assume patterns $P_1,P_2,\ldots,P_k$ are in the same cluster. The representative pattern P_r of the cluster should be able to *express* all the other patterns. Clearly, we have $\bigcup_{i=1}^{k}O(P_i)\subseteq O(P_r)$.

Using the distance measure defined in Section 2.1, we can simply apply a clustering method, such as k-means, on the collection of frequent patterns. However, it introduces two problems. First, the quality of the clusters cannot be guaranteed; and second, it may not be able to find a representative pattern for each cluster (i.e., the pattern P_r may not belong to the same cluster). To overcome these problems, we introduce the concept of δ-cluster, where δ $(0\leqslant\delta\leqslant 1)$ is the tightness measure of a cluster.

Definition 2 (δ-*cluster*). A pattern P is δ-*covered* by another pattern P' if $O(P)\subseteq O(P')$ and $D(P,P')\leqslant\delta$. A set of patterns form a δ-*cluster* if there exists a representative pattern P_r such that for each pattern P in the set, P is δ-covered by P_r.

Remark. First, in δ-cluster, one pattern can belong to multiple clusters. Second, using δ-cluster, we only need to compute the distance between each pattern and the representative pattern in a cluster. Since a pattern P is δ-covered by a representative pattern P_r only if $O(P)\subseteq O(P_r)$, we can simplify the distance calculation by only considering the supports of the patterns: $D(P,P_r)=1-\frac{|T(P)\cap T(P_r)|}{|T(P)\cup T(P_r)|}=1-\frac{|T(P_r)|}{|T(P)|}$. Finally, if we extend the distance definition to non-closed patterns, it is easy to verify that a non-closed pattern must be δ-covered by a representative pattern if its corresponding closed pattern is covered. We have the following lemma with the proof omitted.

Lemma 1. *Given a transaction database, a minimum support M and a cluster quality measure δ, if a representative pattern set R δ-covers all the closed frequent patterns, then R δ-covers all the frequent patterns.*

In the remaining of the paper, when we refer to frequent patterns, we mean *closed* frequent patterns. For simplicity, we use *cover* and *cluster* to represent δ-cover and δ-cluster, respectively.

If we restrict the representative pattern to be frequent, then the number of representative patterns (i.e., clusters) is no less than the number of maximal frequent patterns. This is because a maximal frequent pattern can only be covered by itself. In order to achieve more succinct compression, we relax the constraints on representative patterns, i.e., allow the supports of representative patterns to be less than *min_sup*, M. For any representative pattern P_r, assume its support is k. Since it has to *cover* at least one frequent pattern (i.e., P) whose support is at least M, we have

$$\delta \geqslant D(P, P_r) = 1 - \frac{|T(P_r)|}{|T(P)|} \geqslant 1 - \frac{k}{M}$$

That is, $k \geqslant (1 - \delta)M$. This is the *min_sup* for a representative pattern. To simplify the notation, we use $\widehat{M}$ to represent $(1 - \delta)M$. The pattern compression problem is defined as follows.

Definition 3 (*Pattern compression problem*). Given a transaction database, a *min_sup* M and the cluster quality measure δ, the *pattern compression problem* is to find a set of representative patterns R, such that for each frequent pattern P (w.r.t. M), there is a representative pattern $P_r \in R$ (w.r.t. $\widehat{M}$) which *covers* P, and the value of $|R|$ is minimized.

3. NP-hardness

We show that the problem defined above is NP-hard.

Theorem 2. *The problem of finding the minimum number of representative patterns is NP-hard.*

Proof. We show that the pattern compression problem can be reduced from the set-covering problem.

First, for any pattern compression problem, we can construct a corresponding set-covering problem. There are two *min_sup*s in the pattern compression problem: *min_sup* M and the representative pattern's *min_sup* $\widehat{M}$. We denote the set of frequent patterns (w.r.t. M) as $FP(M)$, the set of frequent patterns (w.r.t. $\widehat{M}$) as $FP(\widehat{M})$. For each pattern $\widehat{P} \in FP(\widehat{M})$, we generate a set whose elements are all the patterns $P \in FP(M)$ which are *covered* by $\widehat{P}$. The set-covering problem is to find a minimum number of sets which cover all the elements, where each set corresponds to a representative pattern.

We then show that for any set-covering problem, we can construct a corresponding pattern compression problem. Let the given set-covering problem contain N elements $\{e_1, e_2, \ldots, e_N\}$ and K sets $\{S_1, S_2, \ldots, S_K\}$. Each set S_i contains n_i elements $\{e_i^1, e_i^2, \ldots, e_i^{n_i}\}$. We assume that (1) there exist no two sets S_i and S_j such that $S_i \subseteq S_j$; and (2) there exists no single set covering all the elements.

By considering these elements as individual items, we first construct α ($\geqslant 0$) transactions, each of which contains N items $\{e_1, e_2, \ldots, e_N\}$; then construct β ($\geqslant 1$) transactions for each individual set (i.e., for each set S_i, there will be β transactions $\{e_i^1, e_i^2, \ldots, e_i^{n_i}\}$). Now we have a database containing $\alpha + \beta K$ transactions. If there is any individual item e_i whose support is less than $\alpha + \beta K$, we further insert transactions with only one item (e_i) until its support reaches $\alpha + \beta K$. Let $M = \alpha + \beta K$ and $\delta = \frac{\beta K - 1}{\alpha + \beta K}$, then $\widehat{M} = \alpha + 1$. Since α and β can be chosen arbitrarily, the value of δ can be selected as any value in $(0, 1)$.

Now we have a database where the longest pattern $\{e_1, e_2, \ldots, e_N\}$ is not frequent w.r.t. $\widehat{M}$. It will not be considered as a representative pattern. Each set in the original set-covering problem corresponds to an itemset whose support is at least $\alpha + \beta \geqslant \widehat{M}$ (we denote the set of all of these itemsets as RP) and can be considered as representative patterns. Each element in the set-covering problem corresponds to an item whose support is exactly M and has to be covered. We show that the solution of the pattern compression problem will only choose representative patterns from RP. This is because RP is the maximal pattern set w.r.t. $\widehat{M}$. If there is a representative pattern $P \notin RP$, we can always find a $P' \in RP$, such that $\mathrm{O}(P) \subset \mathrm{O}(P')$. Since all the frequent patterns covered by P have supports at most M, they can also be covered by P'. We conclude that the optimal selection of the representative patterns corresponds to the solution of the original set-covering problem. □

In the rest of the paper, we treat the following terms as equivalent: *element* vs. *frequent pattern* (*w.r.t. M*); *set* vs. *frequent pattern* (*w.r.t.* $\widehat{M}$); and *set-cover* vs. *set of representative patterns*. For any frequent pattern P (w.r.t. $\widehat{M}$), we denote the set of patterns which can be covered by P as *set*(P).

4. Discovering representative patterns

In this section, we describe algorithms for computing representative patterns.

4.1. The RPglobal method

Generally, the size of the frequent patterns is quite large. It is undesirable to enumerate all the combinations to find the optimal selection. Since the problem is equivalent to the set-covering problem, it is natural to consider some approximate algorithms available in the set-covering problem. The well-known one is the greedy algorithm [7] which iteratively finds the current largest set. The pseudo-code for the pattern compression problem is shown in Fig. 2. Since the precondition for this method is to collect the complete information over the elements and sets, we refer it as *global method* (in contrast to the *local method* to be discussed in the next section).

The code is self-explanatory. Following the result of greedy set-covering [7], the ratio between the number of the representative patterns selected by RPglobal and that of the optimal one is bounded.

Theorem 3. *Given a collection of frequent patterns $\mathcal{F}$, let the set of representative patterns selected by* RPglobal *be C_g, the set of optimal* (i.e., *minimal number*) *representative patterns be C^*, then $|C_g| \leqslant |C^*| \times H(\max_{P\in\mathcal{F}}|set(P)|)$, where $H(n) = \sum_{k=1}^{n}\frac{1}{k}$.*

Proof. See [7]. □

The RPglobal method contains two steps. The first one is to collect the complete coverage information (i.e., find all the frequent patterns Q that can cover P), and the second one is to find the set-covers (i.e., find the set of representative patterns). The greedy set-covering step can be implemented in time complexity of $O(\sum_{P\in\mathcal{F}}|set(P)|)$ [7]. The computational challenge comes from finding the pattern coverage information. Note this coverage problem is different from closedness checking, which can be handled more efficiently because of the following reasons. First, closedness checking only needs to find *one* super-pattern which subsumes the query pattern, whereas the coverage checking has to find *all* super patterns that can cover it. Second, the closedness checking can utilize transaction ID-based hash functions to do fast checking [20], while the coverage checking cannot benefit from it since there is a δ tolerance between the support transaction sets. To facilitate the coverage search, we use an FP-tree-like structure [12] to index all the frequent patterns (w.r.t. $\widehat{M}$). An example of FP-tree is shown in Fig. 3. The FP-tree has a head table associated with it. Single items are stored in the head table. The entry for an item also contains the head of a list that links all the nodes with the same name.

Algorithm (RPglobal) Compute Representative Patterns by Greedy Set-Covering.

Input: (1) A collection of frequent patterns FP w.r.t. $\hat{M}$, (2) a minimum support, M, and (3) a quality measure for clustering, δ.

Output: The set of representative patterns.

```
BEGIN
  for each  $P \in FP$ s.t. $support(P) \geq M$
    Insert $P$ into the set $E$;
    for each $Q \in FP$, s.t. $Q$ covers $P$
      Insert $P$ into $set(Q)$;
  while $E \neq \phi$
    Find a $RP$ that maximizes $|set(RP)|$;
    for each $Q \in set(RP)$
      Remove $Q$ from $E$ and the remaining sets;
    Output $RP$;
END
```

Fig. 2. The RPglobal algorithm.

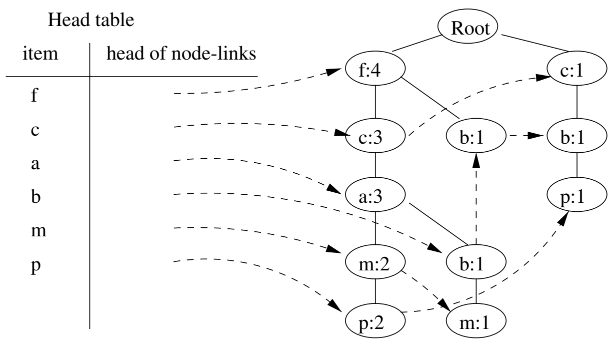


Fig. 3. A sample FP-tree.

The construction of the index tree is similar to FP-tree, except that in FP-tree, the counts of the nodes are updated by summing the counts of the inserted itemsets, while here, the counts of the nodes are updated by choosing the maximum count over the inserted itemsets. To differentiate from traditional FP-tree, we call our index tree as RP-tree (representative pattern tree). The coverage checking using RP-tree works as follows. Suppose the current pattern is Q, $\mathrm{O}(Q) = \{o_1, o_2, \ldots, o_k\}$ (items are ordered as in the RP-tree head table), and its support is C. The support region for a valid representative pattern is $[C \times (1-\delta), C]$. Following the linked list of o_k in RP-tree, for each node n in the list, we test whether (1) the count is within the support region; and (2) the query itemset is a subset of the ancestors of n.

The worst case computation complexity for coverage checking could be $\mathrm{O}(|\mathscr{F}|^2)$. The RPglobal method works well when $|\mathscr{F}|$ is not large. However, when the number of frequent patterns to be compressed increases, the method does not scale well. It is necessary to develop an alternative method which discovers the set of representative patterns efficiently, while still preserves the high quality of the results.

4.2. The RPlocal method

In this subsection, we introduce the idea of a *local method* and show how this method can be efficiently incorporated into the frequent-pattern mining process.

4.2.1. Local greedy method

Computing the complete coverage information is necessary for RPglobal, since the method needs to find a globally maximum set at each step. To develop a scalable method, this expensive computational requirement has to be relaxed. Our objective is to report the representative patterns by an almost linear scan of the whole collection of patterns, without knowing the complete coverage information. The intrinsic relationship among the nearby patterns (according to the order generated by frequent pattern mining algorithms) can be utilized for this purpose.

Most frequent pattern mining algorithms conduct depth-first enumerations in the pattern space (Fig. 4). It starts from an *empty* pattern set, recursively calls the pattern-growth routine to expand the pattern set. Since the individual items are sorted, at any stage of the algorithm, all the single items can be partitioned into three disjoint sets: the *conditional set* (the items appearing in the current pattern), the *todo-set* (the items to be expanded based on the current pattern) and the *done-set* (all the other items).

Example 3. Fig. 10 shows a search space with five single items a, b, c, d, e. At the time when the depth-first search reaches pattern $\{a, c\}$, the conditional set is (a, c), the todo-set is (d, e) and the done-set is (b).

The depth-first search scans each pattern twice: the first visit from its parent, and the second visit after finishing the calls to its children. One can verify that after a pattern is visited in its second time, all the patterns that can possibly cover it have been enumerated. The future patterns are not able to cover it.

We output a pattern in its second visit. The *local greedy method* sequentially scans the output patterns, at any time when an uncovered pattern (called *probe pattern*) is found, the algorithm finds the current largest set

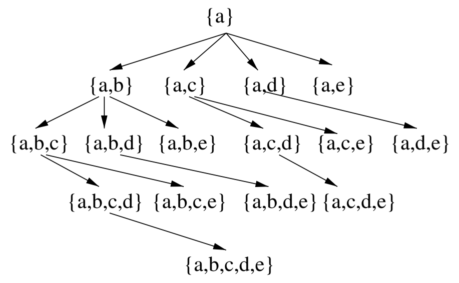


Fig. 4. Depth-first search in pattern space.

(i.e., a representative pattern) which covers it. Here the current largest set has the same meaning as it has in the global greedy method (i.e., the already covered pattern does not count for the set size). The following theorem shows a bound of the local method.

Theorem 4. *Given a collection of frequent patterns $\mathscr{F}$, let the set of representative patterns selected by the local method be C_l, the set of optimal representative patterns be C^*. Assume the minimum number of patterns that cover all the probe patterns be T. Then* $|C_l| < |C^*| \times (\sqrt{2T \times \max_{P\in\mathscr{F}}|set(P)|} + 1)$.

Proof. For the simplicity of presentation, we prove the theorem in the set-covering framework. Let the sequence of probe elements be $e_1, e_2, \ldots, e_l$ ($l = |C_l|$), the sets selected by the local method be $S_1, S_2, \ldots, S_l$. Similar to the approaches in [7], we assign a cost 1 to each set which is selected by the local method, distribute this cost evenly over the elements covered for the first time. If e is covered for the first time by S_i, then $c_e = \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$, and $|C_l| = \sum c_e$. The cost assigned to the optimal cover is $\sum_{S\in C^*}\sum_{e\in S} c_e$. Since each element is in at least one set in the optimal cover, we have

$$|C_l| = \sum c_e \leqslant \sum_{S\in C^*}\sum_{e\in S} c_e = \sum_{S\in C^*} K_S$$

where $K_S = \sum_{e\in S} c_e = \sum_{S'\in C_l}\frac{|R(S')\cap S|}{|R(S')|}$ and $R(S_i) = S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})$.

Let the minimum sets covering all the probe elements be $\{M_1, M_2, \ldots, M_T\}$. We further assume that set M_i, $i \in \{1, 2, \ldots, T\}$, covers probe elements $E_i = \{e_{i_1}, e_{i_2}, \ldots, e_{i_p}\}$ ($i_1 < i_2 < \cdots < i_p$), where $E_i \cap E_j = \phi$, ($\forall i \neq j$), and $\bigcup_{i=1}^{T} E_i = \{e_1, e_2, \ldots, e_l\}$.

Let the set (selected by the local method) associated with probe element e_{i_1} be S_{i_1}. Since S_{i_1} is one of the current largest sets which cover probe pattern e_{i_1}, we have $|R(S_{i_1})| \geqslant |R(M_i)|$. Because $e_{i_1}, e_{i_2}, \ldots, e_{i_p}$ are in the order of probe elements, these elements must have not been covered at the time when e_{i_1} is selected as probe element. Thus, $|R(M_i)| \geqslant p$, and we conclude $|R(S_{i_1})| \geqslant p$. Similarly, we have $|R(S_{i_2})| \geqslant p-1, |R(S_{i_3})| \geqslant p-2, \ldots, |R(S_{i_p})| \geqslant 1$.

For all $S' \in C_l$, assume $S'_1, S'_2, \ldots, S'_l$ is in ascending order of $|R(S')|$. K_S achieves the maximum value if we distribute the elements of S first in set S'_1 fully, then $S'_2, \ldots$, until S'_{k+1}. Since distributing fully on S' means $\frac{R(S')\cap S}{R(S')} = 1$, we have $K_S \leqslant k+1$.

Evenly distribute the first k S' into T buckets, and assign minimum $|R(S')|$ value for them, we have,

$$|S| \geqslant |R(S'_1)| + |R(S'_2)| + \cdots + |R(S'_k)| \geqslant T \times \sum_{i=1}^{\frac{k}{T}} i = T \times \frac{\frac{k}{T} \times \left(\frac{k}{T} + 1\right)}{2} > \frac{k^2}{2T}$$

We have $K_S \leqslant k + 1 < \sqrt{2T|S|} + 1$, thus

$$|C_l| \leqslant |C^*| \times \left(\max_{S\in C^*}(K_S)\right) < |C^*| \times \left(\sqrt{2T \times \max_{S}|S|} + 1\right) \qquad \square$$

The difference between the local method and the global method is the selections of the probe patterns. Clearly, if the probe patterns are selected as patterns in the current largest set, the local method is identical to the global method. Since the complete coverage information is not available, the bound on the local method is worse than the global method. However, in our experiments, we found that the performance of the local method is very close to that of the global method. This is because in the pattern compression problem, the layout of the patterns and their coverage is not arbitrary. Instead, most frequent patterns are strongly connected if they are in the same cluster, i.e., a pattern P is covered by the representative pattern P_r if and only if P_r subsumes P and the distance between P and P_r is within δ. As a result, for each pattern P_r, $set(P_r)$ preserves local compactness, that makes the selections of probe patterns not a dominant factor for compression quality. Meanwhile, in most pattern compression problem, the sizes of sets are somewhat balanced. It is unlikely to have a very large set which is selected by the global method but missed by the local method, thus leads to a significant performance difference.

The local method relaxes the requirement of global comparison of sets sizes. However, it still needs to find the current largest set at each step, which involves the coverage checking for the future patterns. We further relax the method by finding a *reasonably large* set, instead of the *largest* set. The reasonably large set is expected to cover more future patterns. Intuitively, the candidate set should be as long as possible, since longer patterns generally have larger coverage. The candidate set should also contain more items within the *probe pattern's todo-set*. This is because items in todo-set are expected to appear more in the future.

These intuitions are well justified by the real experimental statistics. Fig. 5 (*connect* data set [8] with $min\text{-}sup = 0.8 \times \#transactions$, $\delta = 0.1$) shows the future coverage w.r.t. a probe pattern which is output at position 5043. The future coverage counts the number of coverable patterns which are output after the probe pattern. We ignore the patterns which cannot cover the probe pattern (i.e., the future coverage is 0). The values on the y-axis are normalized w.r.t. the largest coverage. The x-axis is the order in which patterns are output. The probe pattern is first visited at position 4952. Fig. 6 shows the corresponding pattern length (normalized w.r.t. the longest length). We observe that the largest future coverage appears between the first visit and second visit of probe pattern, and it also has the longest length within the same region. Based on the above observations, we select the *reasonably large* set as the longest pattern, which can cover the probe pattern, among all the patterns between the first visit and second visit of the probe pattern (i.e., patterns expanded by the probe pattern).

Since the local method only requires the knowledge on already-discovered patterns, we further integrate it into frequent pattern mining process in order to improve the computational efficiency. The new method is called RPlocal.

4.2.2. The algorithm

We develop an FP-growth-like algorithm [12,10] to discover representative patterns. Since FP-growth is a well-known method, we omit the detailed description here due to the limited space.

The RPlocal algorithm is described in Fig. 7. We explain the algorithm line by line. Line 1 picks an item to be expanded from the head table. Line 2 pushes the item onto a global stack IS which keeps track of the itemsets along the path from the root in the pattern space. Each entry of IS has the following fields: *item name*,

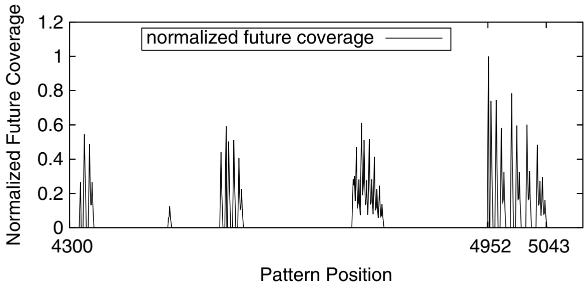


Fig. 5. Patterns' positions and their coverage.

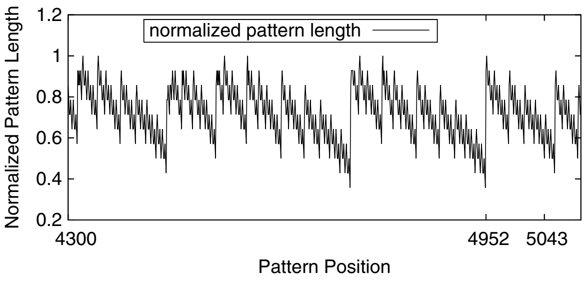


Fig. 6. Patterns' positions and their length.

```
Algorithm (RPlocal) Compute Representative Patterns by Local Search.
Input: (1) a transaction database $D$, (2) a minimum support, $M$ and (3) a
quality measure for clustering, $\delta$.
Output: The set of representative patterns.
BEGIN
    $IS = \phi$; // global stack to keep itemsets
    scan $D$, create FP-tree $F$;
    Initiate an empty RP-tree $R$;
    call $FPrepresentative(F, R)$;
END

procedure $FPrepresentative(F, R)$ {
1.  for each $item$ in $F.headtable$ {
2.      push $item$ into $IS$;
3.      if $(closed\_pruning(IS(top)) = true)$ continue;
4.      $set\_representative()$;
5.      if $F.headtable[item].count < M$
            or $coverage\_checking(IS(top), R) = false$;
6.          $IS[top].covered = true$;
        else
7.          $IS[top].covered = false$;
8.      $Todo - set$ = {frequent (w.r.t. $\hat{M}$) items based on
            the current conditional set };
9.      Build a new FP-tree $F_{new}$ based on $Todo - set$;
10.     Initiate $F_{new}$'s RP-tree, $R_{new}$;
11.     call $FPrepresentative(F_{new}, R_{new})$;
12.     if $((RP = get\_representative()) \neq NULL)$
13.         Insert $RP$ into $R$ and its predecessor RP-trees;
14.         Output $RP$;
15.     pop item from IS;
16.     $item\_reordering()$;
    }
}
```

Fig. 7. The RPlocal algorithm.

counts, *covered*, and *cover pattern* (among its children). Line 3 checks whether closed pruning can be applied on *IS*(*top*). We discuss the challenge and our solution to the closed pruning problem in Section 4.2.3. Line 4 traces all the itemsets in the *IS* stack to check whether the current itemset can cover the itemsets in the stack. If yes, and the current pattern is longer than the one stored in the *cover pattern* field, then the *cover pattern* field is updated by the current pattern. Line 5 checks current itemset's support. If it is less than *M*, then it is not required to be covered. It also checks whether it is covered by a previous representative pattern (the previous representative patterns are indexed in RP-tree R, as shown in RPglobal). Lines 8–11 are the same as the FP-growth algorithm. It collects the todo-set based current itemset, constructs a conditional FP-tree and a conditional RP-tree for the coverage checking, then recursively calls the *FPrepresentative* routine with the new trees. The details of coverage checking and the conditional RP-tree are discussed in Section 4.2.4. Line 12 checks whether the current itemset can be a probe pattern. If it is, then the *cover pattern* stored in the *IS* stack is selected as a new representative pattern. Meanwhile, the *covered* fields of all the other itemsets in the *IS* stack are set as true if they can be covered by the new representative pattern. The new representative pattern is inserted into all RP-trees for future coverage checking. Finally, line 16 is a pruning technique that we will discuss in Section 4.2.5.

4.2.3. Prune non-closed patterns

Here we discuss the implementation of *closed_pruning* in the RPlocal algorithm. Assume a pattern P is not closed, and the related closed pattern is P_c. There are two possibilities for P_c. One is $(O(P_c) - O(P)) \cap$ *done-set* $\neq \phi$, then P_c is a pattern discovered before the first visit of P. The other is $(O(P_c) - O(P)) \cap$ *done-set* $= \phi$, then P_c is a pattern expanded by P (i.e., P_c is discovered between the first visit and the second visit of P).

The second case is intrinsically handled by the RPlocal algorithm at line 8, where items with the same frequency as P are directly merged into the conditional set, and the non-closed itemsets are skipped. The first case is more interesting w.r.t. the computation pruning. The following lemma has been widely used in most of the closed frequent pattern mining methods [20], and we state it without proof.

Lemma 2. *In the RPlocal method, for any pattern P, if there exists a pattern P_c which was discovered before the first visit of P, s.t. $O(P) \subset O(P_c)$ and $|T(P)| = |T(P_c)|$, then all the patterns being expanded by P (i.e., patterns within the first and second visits of P) are not closed.*

We call this pruning technique as *closed pruning*. The function *closed_pruning* is to check whether the pattern is not closed w.r.t. a previously discovered pattern. The challenge for closed pruning in the RPlocal algorithm is that only representative patterns are kept, and generally it is a small subset of closed patterns. It is not possible to check the closedness of a pattern using the previous outputs. Keeping all the closed patterns is one option. However, an interesting observation from our experiments shows that even without closed pruning, RPlocal runs faster than the closed frequent pattern mining algorithm. This is because the *coverage checking* in RPlocal is much more efficient than the *closedness checking* in closed frequent pattern mining since the number of representative patterns to be checked with is significantly less than the number of closed frequent patterns in closedness checking. Keeping all the closed patterns obviously will degrade the performance of RPlocal.

Instead of checking the closedness with the previous output, RPlocal uses a closedness checking method which tries to memorize the information of items in *done-set*. Since we only need to know whether an item is present or not in the closed pattern, we use 1 bit to represent an item's presence. If there are N single frequent items, we use a N-bit array (referred as *closed_index*) for each pattern. The *closed_index* of a pattern is computed by the bit-and operation between the *closed_indices* of all the related transactions. An example on how to use *closed_index* is shown as follows.

Example 4. Given a database having 5 transactions: $\{f,c,a,m,p\}$, $\{f,c,a,b,m\}$, $\{f,b\}$, $\{c,b,p\}$, $\{f,c,a,m,p\}$, we use $N = 6$ bits for the *closed_index*. Items f,c,a,b,m,p are assigned to the 1st to 6th bits, according to the computation order. The *closed_indices* of transactions 1 and 5 are 111011, and the *closed_index* of transaction 2 is 111110. The *closed_index* of pattern $\{c,a\}$ is 111010. Since item f is in the done-set of pattern $\{c,a\}$ and f's bit is 1. We conclude that the closed pruning can be applied on pattern $\{c,a\}$.

To efficiently compute the *closed_index* for each pattern, we attach *closed_index* to each node in the FP-tree. The *closed_index* can be aggregated along with the count measure, except that the count is updated by sum, while the *closed_index* is updated by bit-and. Since the *closed_index* is attached to every node, the method is limited by the memory requirement. Fortunately, our task is not to identify all closed pruning. Instead, we aim to prune as much as possible, and the unpruned non-closed patterns will go to the coverage checking. The problem turns out: Given a fixed number of k for *closed_index*, if the total number of single frequent items is larger than k, how to select k items from them, and how much pruning can be achieved?

It is natural to choose the first k items according to the computation order because the closed pruning checks patterns with items in *done-set*. The experimental statistics on *pumsb_star* [8] data set is shown in Fig. 8, where we collect the percentage of closed pruning achieved, by setting k as 32 and 64. We observe this simple optimization works quite well. With only one integer ($k = 32$) as *closed_index*, the method misses less than 2% and 15% closed pruning when the number of frequent items are 2 and 6 times of k, respectively. Using two integers ($k = 64$) as *closed_index*, the method misses less than 1% of the closed pruning. The similar phenomena are also observed on all the other data sets (e.g., mushroom, connect, accidents, chess) in our experimental evaluations. It is interesting to see that these limited k bits achieve good pruning percentages. We give a detailed explanation in the rest of this subsection.

Assume there are totally n independent frequent items, whose computation order is $o_1, o_2, \ldots, o_k, \ldots, o_n$ (for simplicity, we assume the order of items keeps unchanged). We leave the first k for *closed_index*, and $r = n - k$ as left items. The percentage of the closed pruning by *closed_index* is defined as function $h(k, r)$.

For any pattern, let the conditional set be $\{o_{i_1}, o_{i_2}, \ldots, o_{i_m}\}$, where $i_1 < i_2 < \cdots < i_m$, we say m is the *length* of the pattern and i_m is the *span* of the pattern. The position j is called a *hole* if $j < i_m$ and $j \notin \{i_1, i_2, \ldots, i_m\}$. If the set of holes is not empty (i.e., the done-set is not empty), then this pattern is possible to be subsumed by a previously output pattern (i.e., the closed pruning is possible to be applied on). A hole is *active* if the closed pruning takes effect.

The items in the conditional set are distributed into two parts: the first k items set and the rest r items set. Let the number of items falling in the rest r items set be v, and the number of holes falling in the rest r items set be u (referred as (u, v)-*configuration*). To estimate the percentage of closed pruning for a (u, v)-configuration (defined as $g(k, u, v)$), we need to further define two parameters: the expect number of active holes c and the maximal pattern length l.

Assume items are independent, every *hole* has an equal opportunity to be *active*. If there is *one* hole, which exists in the first k items, then the closed pruning is caught by the *closed_index*, otherwise, it misses. Since there are at most $l - v$ items falling into the first k items set, for each $0 \leqslant i \leqslant m = \max(l - v, k)$, there are $\binom{k}{i}$ different patterns. For each pattern, the number of all different placements for c active holes is $\binom{k - i + u}{c}$, and the number of placements that all c active holes falling in the rest r items set is $\binom{u}{c}$. Thus the pruning percentage by *closed_index* is

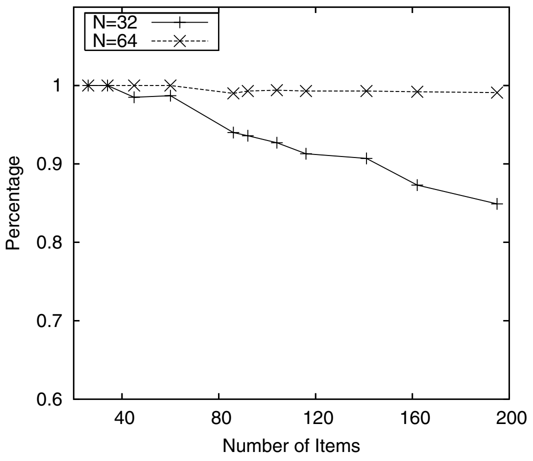


Fig. 8. Percentage of closed pruning achieved.

$$1 - \frac{\binom{u}{c}}{\binom{k-i+u}{c}},$$

we have

$$g(k,u,v) = \frac{\sum_{i=0}^{m}\left(1 - \frac{\binom{u}{c}}{\binom{k-i+u}{c}}\right) \times \binom{k}{i}}{\sum_{i=0}^{m}\binom{k}{i}}.$$

Now we examine the value of $h(k,r)$. Among all patterns, there are two cases which are divided evenly. First, the last item is not in the conditional set. In this case, the pruning percentage is same as $h(k,r-1)$. Second, the last item is in the conditional set. In this case, we enumerate all possible (u,v)-configurations. There are at most $l-1$ bits to be placed within the latter r items (since the last item is already in, there are $r-1$ selections). For each $0 \leqslant i \leqslant m = \max(l-1, r-1)$, there are $\binom{r-1}{i}$ different (u,v) *configurations* $(u = r-1-i, v = i+1)$, and for each configuration, the pruning percentage is $g(k, r-1-i, i+1)$, we have:

$$h(k,r) = \frac{1}{2}h(k,r-1) + \frac{\sum_{i=0}^{m}\binom{r-1}{i} \times g(k,r-1-i,i+1)}{2\sum_{i=0}^{m}\binom{r-1}{i}}$$

The base case is $h(k,0) = 1$. We run simulations by setting $k = 32$ and varying r from 0 to 64. We observe that, in most cases, the maximal pattern length is approximately proportional to the number of frequent items. Typically, we select the ratio as $\frac{1}{3}$, which is close to the experiments in *pumsb_star* data set. The value of expected active closed bits c is varying from 1 to 4.

The simulation result in Fig. 9 shows that the percentage of pruning increases as c increases. This is because when c is large, the probability that at least one active holes are caught by *closed_index* is high. Typically, when $c = 4$, the simulation curve is close to the real experimental results. Note $c = 1$ is the base line where

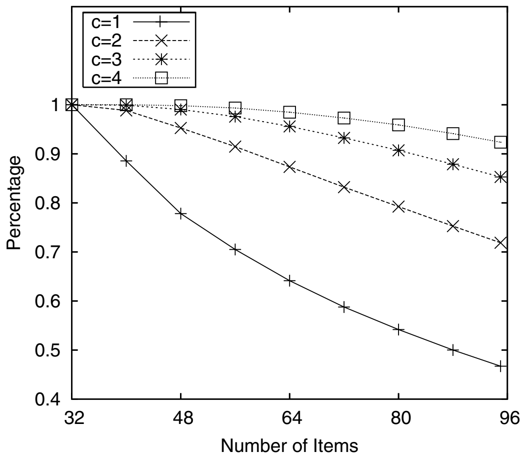


Fig. 9. Simulation results: percentage of closed pruning achieved.

the corresponding curve represents the lower bound of closed pruning. We believe the *closed_index* method has practical usage in real applications. The reason is as follows. The number of frequent patterns is exponentially explosive w.r.t. the number of items. Within the current computational limit, if the number of items is large, either the maximal length of the pattern is small or the pattern length is not small but the value of c is reasonably large (thus the output closed patterns can be kept in a reasonable size). In the former case, the effect of closed pruning (using whatever methods) to the whole computational efficiency is limited; while in the latter case, our study shows that *closed_index* can achieve considerable improvement. Furthermore, the *closed_index* approach only involves bit operations, which is very efficient.

4.2.4. Coverage checking

The task of coverage checking is to check whether an itemset is covered by a previous output representative pattern. We have made two efforts to improve the efficiency: First, we use conditional RP-tree to check the coverage; Second, we use *coverage_index* to speedup the subset checking. The idea of building a conditional RP-tree for each FP-tree has been used for closed checking in [10]. Since the FP-tree is conditional (i.e., all the itemsets generated by the current FP-tree share the same conditional items), it is natural to construct a conditioned RP-tree associated with the current FP-tree, such that all the representative patterns stored in the conditional RP-tree share the same conditional items. We can expect that the conditioned RP-tree would be smaller than the whole RP-tree.

The function *coverage_checking* works as we described in RPglobal algorithm. The major part of the computational cost is to traverse upword in the RP-tree to verify whether the query pattern is subsumed. We improve it by adopting a similar technique used in Section 4.2.3. Every node in RP-tree is attached with a *coverage_index*, which summarizes whose ancestor items. We allocate M bits for the *coverage_index*, and each item is assigned to one bit. In our experiments, we set $M = 32$ (one integer), this is because the coverage checking runs heavily in the whole mining procedure, and we want this part as efficient as possible. A bit is set to 1 iff the corresponding item appears in the ancestors. The following rule is used to reduce the comparisons: *Let the coverage_index of the query pattern be Q, the coverage_index in the node be N, if $Q \,\&\, N \neq Q$, where & is bit-and operation, then the query pattern cannot be a subset of the ancestors of the node.*

There are several differences in the *coverage_index* and *closed_index*: First, the bit assignment in closed checking is global (i.e., an item is assigned to the same bit from scratch to the end), while in coverage checking, it is local, (i.e., bits are dynamically reassigned in each RP-tree for the remaining frequent items). Second, if the number of remaining items is larger than M, we assign not only bits for the first M items, but also multiple items to one bit by distributing all items evenly on the M bits. Finally, the *closed_index* is updated when new itemsets are inserted, while the *coverage_index* of each node is fixed by all its ancestors.

4.2.5. Item reordering

We now discuss the *item_reordering* routine on line 16 of the algorithm. This pruning method is based on the following lemma. Due to the limited space, we omit the proof.

Lemma 3. *Let P_r be a representative pattern, if pattern P_u is covered by P_r, then for any pattern P, such that* $O(P_u) \subseteq O(P) \subseteq O(P_r)$, *$P_r$ covers P.*

For example, if itemset $\{a\}$ is covered by a representative pattern $\{a,b,c,d\}$, so are patterns $\{a,b\}$, $\{a,c\}$, $\{a,d\}$, $\{a,b,c\}$, $\{a,b,d\}$, and $\{a,c,d\}$, thus all these patterns can be safely pruned. However, the pruning technique is non-trivial because the prunable space is embedded in the whole search space, removing this space would leave the whole search space unconnected.

Example 5. Take Fig. 10 as an example. Assume itemset $\{a,b,c,d\}$ covers $\{a\}$, and $\{a,b,c,d\}$ has been selected as a representative pattern. According to Lemma 3, the search space containing $\{a,b,d\}$, $\{a,c\}$, $\{a,d\}$, and $\{a,c,d\}$ can be pruned. However, if we remove them, there are several patterns which contain item e (i.e., $\{a,b,d,e\}$, $\{a,c,d,e\}$, $\{a,c,e\}$, and $\{a,d,e\}$) will be unreachable. To overcome this problem, we dynamically reorder the search space as part (b) in Fig. 10, where we search item e first, items c, d will be skipped under node $\{a\}$ and $\{a,b\}$, but will go back to search space after item e is considered. The search space within the dashed lines in part (b) of Fig. 10 are pruned.

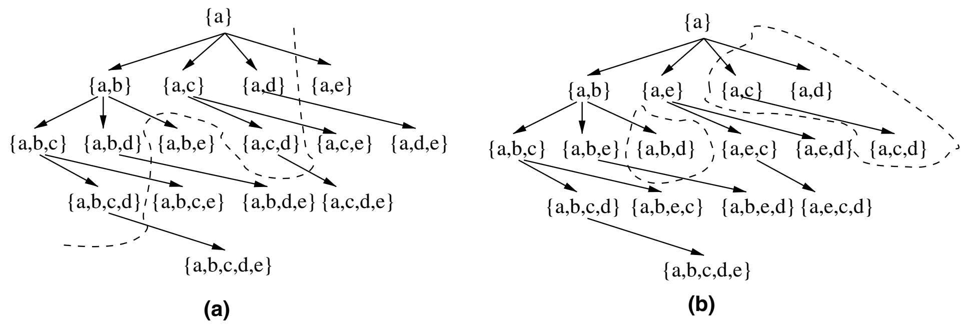


Fig. 10. Reorder items (c, d).

We refer items (c, d) as the set of *prunable-items*. There will be at most one prunable-items set directly under one node. If there are two prunable sets, we have to make choice to select one. For example, if both $\{a, b, c, d\}$ and $\{a, b, c, e\}$ are selected as representative patterns under node a, we know that both (c, d) and (c, e) can serve as prunable-items set. However, if we reorder all of them (i.e., (c, d, e)) to the tail part, the itemset $\{a, c, d, e\}$ are not reachable, but $\{a, c, d, e\}$ has not yet been examined for coverage. In this case, we select the prunable-items set which contains more items in hope to prune more search space.

4.3. Combining RPglobal and RPlocal

We have developed two algorithms for the pattern compression problem. The RPglobal method has guaranteed compression bound but is worse on scalability, whereas the RPlocal method is efficient but worse on the compression. In this subsection, we discuss a combined method: RPcombine.

The main idea of RPcombine is to first use RPlocal to get a small subset of candidate representative patterns, then use RPglobal to find the final results. To ensure that all the frequent patterns (w.r.t. M) are δ-covered, we need to choose the parameters for each step carefully.

Assume the quality measures for RPlocal and RPglobal are δ_l and δ_g, respectively. Any frequent pattern P must be δ_l-covered by a candidate representative pattern P_l, which is further δ_g-covered by a final representative pattern P_g. An obvious result from Theorem 1 shows if we choose $\delta_l + \delta_g = \delta$, then P is guaranteed to be covered by P_g. Here we exploit a better assignment. We have:

$$D(P, P_l) = 1 - \frac{|T(P_l)|}{|T(P)|} \leqslant \delta_l, \quad D(P_l, P_g) = 1 - \frac{|T(P_g)|}{|T(P_l)|} \leqslant \delta_g$$

The constraint given by the problem is

$$D(P, P_g) = 1 - \frac{|T(P_g)|}{|T(P)|} \leqslant \delta$$

To achieve more compression, we would like the values of δ_l and δ_g to be as large as possible. This implies:

$$(1 - \delta_l) \times (1 - \delta_g) = 1 - \delta$$

We use λ to control the tradeoff between δ_l and δ_g, such that $\delta_l = \lambda\delta$ and $\delta_g = \frac{(1-\lambda)\delta}{1-\lambda\delta}$. We further discuss the selection of *min_sup* for RPlocal and RPglobal: M_l and M_g. Since the original compression problem has parameter M and δ, we have $\widehat{M} = (1 - \delta) \times M$ for representative patterns. The RPlocal step needs to keep the same *min_sup* for representative patterns. Thus,

$$M_l = \frac{\widehat{M}_l}{1 - \delta_l} = \frac{\widehat{M}}{1 - \delta_l} = \frac{(1 - \delta)M}{1 - \lambda\delta}$$

All the frequent patterns (w.r.t. M) are δ_l-covered in the RPlocal step. To ensure that they are δ-covered finally, the RPglobal step needs to δ_g-cover all the patterns that possibly δ_l-cover the frequent patterns (w.r.t. M). Thus,

$$M_g = (1 - \delta_l) \times M = (1 - \lambda\delta) \times M$$

In conclusion, RPcombine takes three parameters: M, δ, λ. The RPlocal step uses parameters M_l, δ_l on the database, and the RPglobal step uses parameters M_g, δ_g on the outputs of the first step.

5. Performance study

We first demonstrate the quality and computational performance using the benchmark data sets in the *frequent itemset mining dataset repository* [8], then show a case study on a real text document data. All the algorithms were implemented in C++, and all the experiments were conducted on an Intel Pentium-4 2.6 GHz system with 1 GB RAM. The system ran Linux with the 2.6.1 kernel and gcc 3.3.2. The methods to be compared are summarized as follows. In the FPClose method, we generate all the closed frequent patterns w.r.t. M (we use FPClose package [10], which is the winner of FIMI workshop 2003 [8]). In the RPglobal method, we first use FPClose to get all the closed frequent itemsets with *min_sup* $\widehat{M} = M \times (1 - \delta)$, then use RPglobal to find a set of representative patterns covering all the patterns with *min_sup* M. In the RPlocal method, we directly compute all the representative patterns from database.

5.1. Number of presentative patterns

The first set of experiments compare three algorithms w.r.t. the number of output patterns. We select *accidents*, *chess*, *connect* and *pumsb_star* data sets [8]. For each data, we vary the value of *min_sup* as the percentage of the number of total transactions and fix $\delta = 0.1$ (we think it is a reasonably good compression quality). The results are shown from Figs. 11–14. We have the following observations: First, both RPglobal and RPlocal are able to find a subset of representative patterns, which is almost two orders of magnitude less than the whole collection of the closed patterns; Second, although RPlocal outputs more patterns than RPglobal, the performance of RPlocal is very close to RPglobal. Almost all the outputs of RPlocal are within two times of RPglobal. The results of RPglobal are partial in that when minimum support becomes low, the number of closed patterns grows very fast, the running times of RPglobal exceed the time limit (30 min).

5.2. Running time

The corresponding running time of the three methods are shown from Figs. 15–18. The times for RPglobal include FPClose procedure. The results show that RPglobal does not scale well w.r.t. the number of patterns, and is much slower than RPlocal. Comparing FPClose and RPlocal, we observe that although RPlocal examines more patterns than FPClose (i.e., RPlocal examines the patterns with *min_sup* $\widehat{M}$, while FPClose only examines the patterns with *min_sup* M), RPlocal runs faster than FPClose, especially when *min_sup* is low.

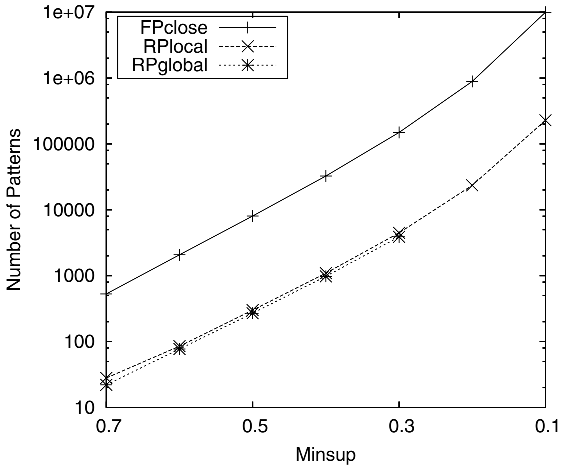


Fig. 11. Number of output patterns w.r.t. *min_sup*, accidents data set.

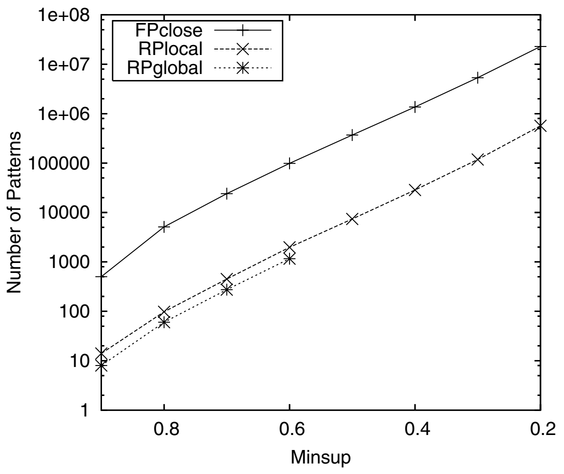


Fig. 12. Number of output patterns w.r.t. *min_sup*, chess data set.

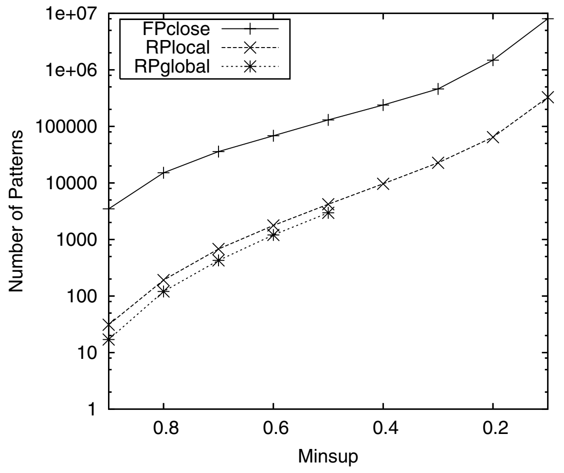


Fig. 13. Number of output patterns w.r.t. *min_sup*, connect data set.

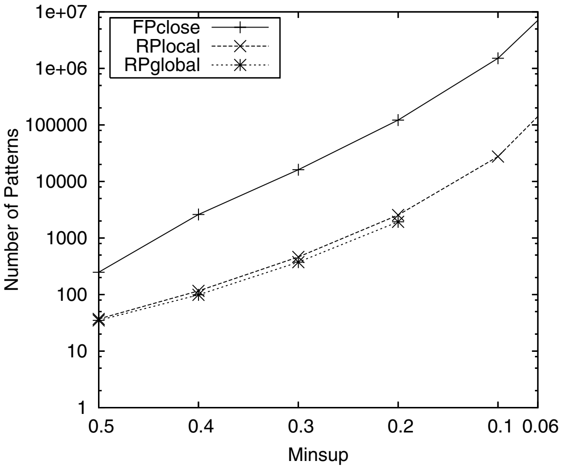


Fig. 14. Number of output patterns w.r.t. *min_sup*, Pumsb_star data set.

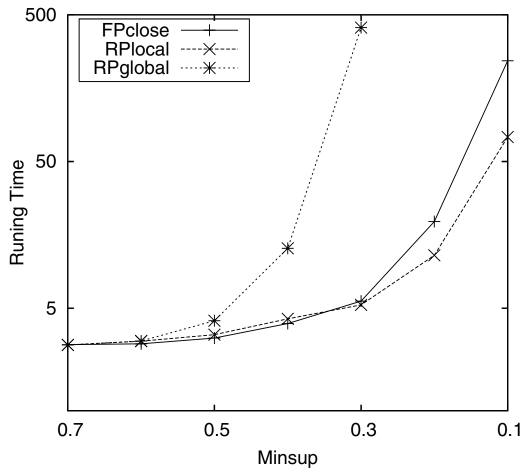


Fig. 15. Running time w.r.t. *min_sup*, accidents data set.

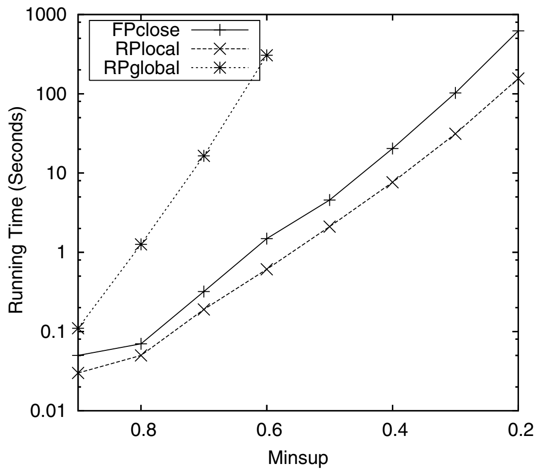


Fig. 16. Running time w.r.t. *min_sup*, chess data set.

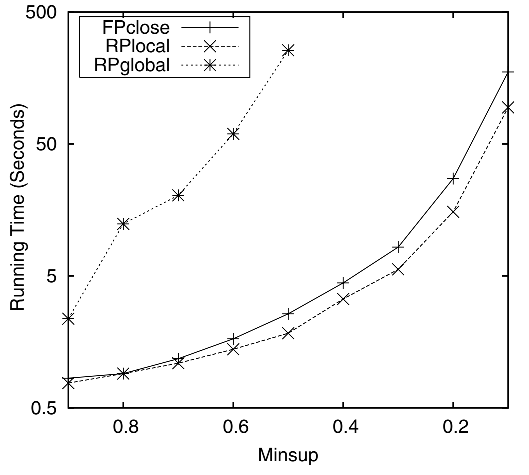


Fig. 17. Running time w.r.t. *min_sup*, connect data set.

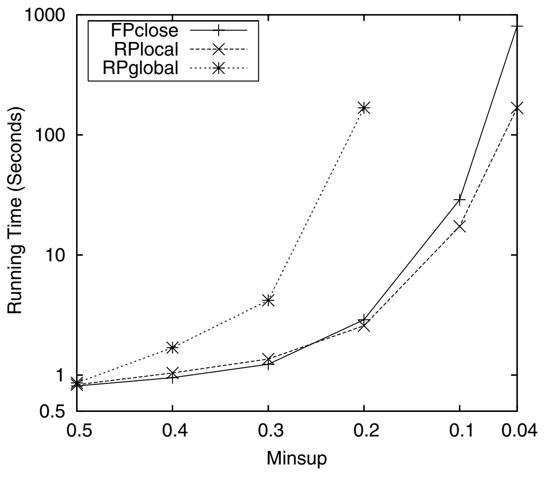


Fig. 18. Running time w.r.t. *min_sup*, Pumsb_star data set.

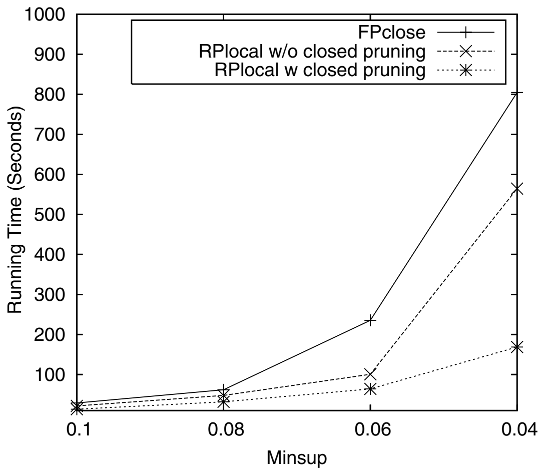


Fig. 19. Running time w.r.t. closed pruning, Pumsb_star data set.

We further investigate the benefit of *closed pruning*. Fig. 19 shows the results on *pumsb_star* data set, with three configurations: FPClose, RPlocal ($\delta = 0.1$) with and without *closed pruning*. We observe that even without *closed pruning*, RPlocal is more efficient than FPClose. This is because in RPlocal, the number of representative patterns is much less than the number of closed patterns. As a result, both the construction and query on RP-trees are more efficient. We use two integers as *closed_index* and the improvement by applying *closed pruning* is significant. When $M = 0.04$, the *closed pruning* version runs three times faster than the version without *closed pruning*, and four times faster than FPClose. At that time, the number of frequent items is 173.

5.3. Distribution of representative patterns

The distributions of representative patterns w.r.t. pattern lengths and pattern supports are shown from Figs. 20–23. We use *accidents* data set, with $min\_sup = 0.4$. In order to get a high-level summary of support distributions, we group supports into 10 buckets. The bucket id is computed by $\lfloor \frac{10 \times support}{\#transactions} \rfloor$. Fig. 20 shows the distributions w.r.t. the pattern lengths for three methods: FPClose, RPglobal and RPlocal ($\delta = 0.1$). We observe that the overall shape of the distributions of RPglobal and RPlocal are similar to the shape of closed patterns. RPglobal and RPlocal have certain shifts to longer length because the nature of the compression problem

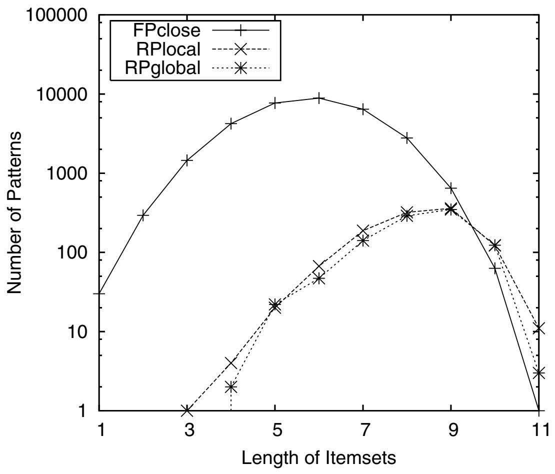


Fig. 20. Distribution of patterns w.r.t. pattern length, accident data set.

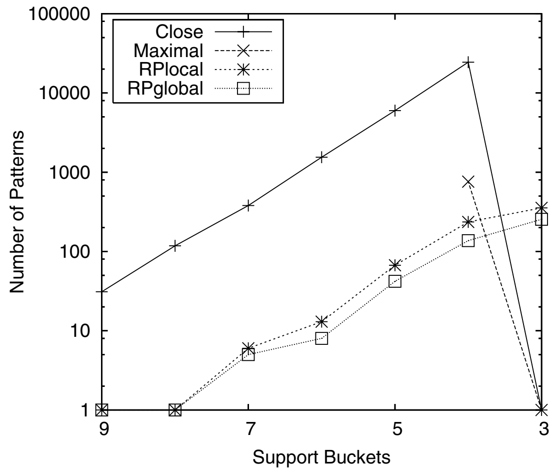


Fig. 21. Distribution of patterns w.r.t. support buckets, accident data set.

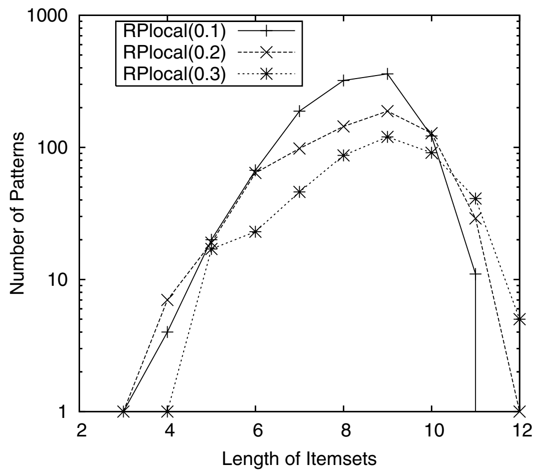


Fig. 22. Distribution of patterns w.r.t. pattern length, accident data set.

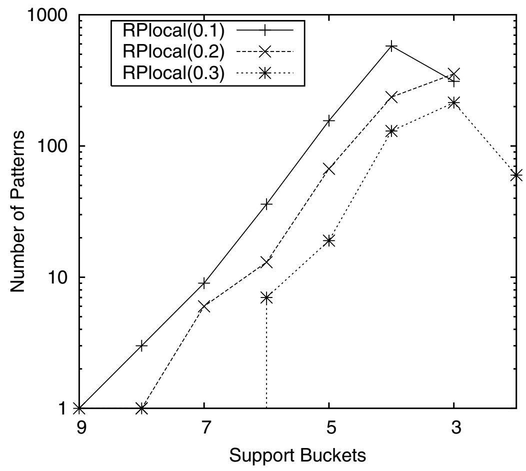


Fig. 23. Number of output patterns w.r.t. support buckets, accidents data set.

favors larger itemsets. Fig. 21 compares the distributions (w.r.t. pattern supports) of close itemsets, maximal itemsets, and representative itemsets by RPglobal and RPlocal ($\delta = 0.2$). While the maximal itemsets catch the boundary supports only, RPglobal and RPlocal are able to get a reasonable distribution which is similar to the original closed itemsets. These suggest that both RPglobal and RPlocal achieve high quality compressions.

We also run RPlocal with different δ from 0.1 to 0.3. Figs. 22 and 23 show the pattern distributions w.r.t. lengths and supports. As we expected, the number of representative patterns decreases when the value of δ increases, because a larger value of δ enables a representative pattern to cover more patterns. Increasing the value of δ also shifts the distributions of the patterns to longer and lower support patterns.

5.4. Additional tests

We examine the performance of RPcombine w.r.t. the different values of λ. Fig. 24 shows the final number of representative patterns by RPglobal, RPlocal and RPcombine on *chess* data set (with $M = 0.6$, $\delta = 0.1$). Fig. 25 shows the corresponding running time. The times of RPcombine are the sums of local and global steps. The λ for RPcombine is varied from 0.01 to 0.4. When λ is small (i.e., 0.01), the local step reports more candidates and the global step takes more time, but the compression quality is better. The compression quality

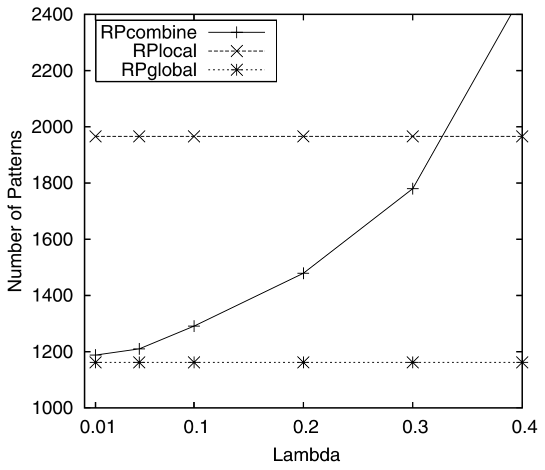


Fig. 24. Number of output patterns w.r.t. λ, chess data set.

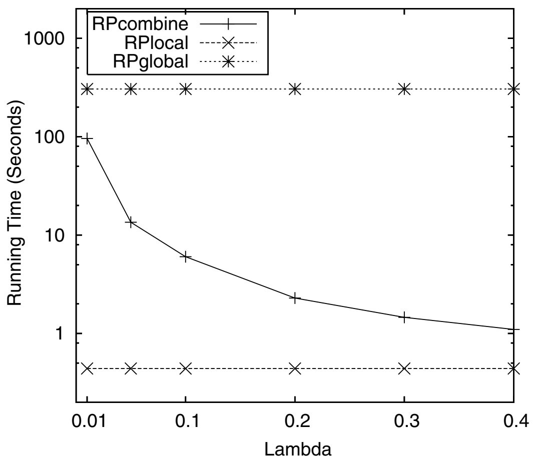


Fig. 25. Running time w.r.t. λ, chess data set.

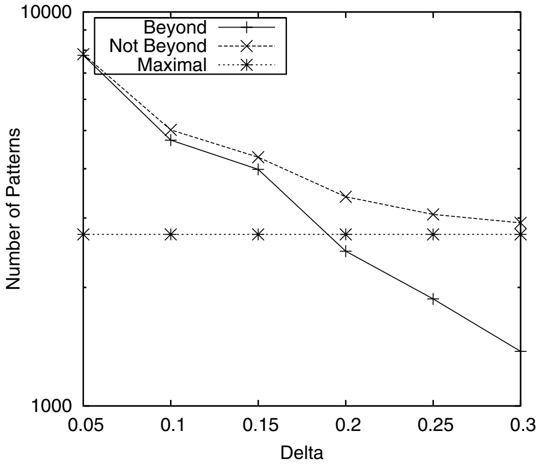


Fig. 26. Number of output patterns, accidents data set.

degrades as λ increases, the number of representative patterns is even larger than RPlocal when $\lambda = 0.4$. This is because at that time, M_g decreases a lot in order to guarantee all the original frequent patterns are covered, and δ_g also decreases. As a result, the global step needs to cover more pattern with tighter quality measure. In most applications, we suggest to choose $\lambda = 0.1$.

The final experiment is designed to verify the benefit to allow the support of representative patterns to be beyond of *min_sup M*. We compare the number of output patterns with three different options: (1) the *Beyond* case where the *min_sup* of representative patterns is $\widehat{M}$; (2) the *Not Beyond* case where the *min_sup* of representative patterns is M; and (3) maximal patterns with *min_sup M*. We use the *accident* data set, varying δ from 0.05 to 0.3, while fixing M as 0.3. The results in Fig. 26 show that the *beyond* case gets fewer number of representative patterns, especially in the case when δ is large, while the *not beyond* case has maximal patterns as its lower bound.

5.5. Document theme extraction: A case study

Theme discovery uses knowledge about the meaning of words in a text to identify broad topics covered in a document [15]. One way to find themes from text document is to extract the frequent patterns of term occurrence. For example, a frequent pattern of "database management..." indicates that the document might be

Table 2
Top-5 document themes

FPClose	RPglobal	RPlocal
permission make digital copy personal grant without fee distribute	permission make digital copy personal grant without fee distribute	permission make digital copy personal grant without fee distribute
permission make digital copy personal distribute commercial full citation	thailand authority lack sufficient refrigerate unit stop corpse rot	color lcd display hour battery life apple click wheel
database manage database application mine algorithm keyword	volunteer specialist account firm pricewaterhousecooper work unite nation	thailand authority lack sufficient refrigerate unit stop corpse rot
database manage database mine algorithm keyword	state department spokesman comment intend estimate long military operate	grow frustrate slow process whittle down list
database manage database application mine algorithm	dna test carry confirm dead bid clear misidentification	tv radio station broke normal programming broadcast

related to a collection of database papers, whereas a frequent pattern like "red cross..." might identify the topic of the documents as aid and relief. In this case study, we show how to apply our methods to discover term occurrence patterns efficiently and understand the underlying text data effectively.

The document collection is constructed by a mixture of documents of four topics: 386 news articles about Tsunami, 367 research papers about data mining, 350 research papers about bioinformatics, and 347 blog articles about iPod Nano. A document is broken into sentences as transactions. For all three algorithms (FPClose, RPglobal and RPlocal), we use *min_sup* as 0.0002. The δ value for RPglobal and RPlocal is set as 0.5. Since it is not possible to show all the mining results in the paper, we report a subset of top significant patterns (Table 2). A pattern's significance is modeled by a *tf-idf* scoring function similar to the Pivoted Normalization weighting based document score [19]. Specifically, given a theme pattern $p = w_1 \cdots w_t$, the significance is defined by

$$S(p) = \sum_{i=1}^{t} \frac{1 + \ln(1 + \ln(tf_i))}{(1 - s) + s\frac{dl}{avdl}} \cdot \ln \frac{N + 1}{df_i}$$

where tf_i equals the support of the pattern p and df_i is the inverse document frequency of word w_i in the whole transaction set.

Without considering redundancy, the top-5 results returned by FPClose only consist of two valuable themes (themes 1 and 3), and all the others are redundant. RPglobal and RPlocal report five significantly different themes. The third theme appears at 17th (12th) in RPglobal (RPlocal) results. This is because both RPglobal and RPlocal allow the frequencies of representative patterns to be less than *min_sup*, and they can discover more significant patterns w.r.t. *tf-idf* score. Since RPglobal and RPlocal use different heuristics, their results are not identical. However, both of them cover patterns discovered by FPClose with $\delta = 0.5$.

6. Related work

Lossless methods have been proposed to reduce the output size of frequent itemset patterns. Ref. [16] developed the concept of closed frequent patterns, and [6] proposed mining non-derivable frequent itemsets. These kinds of patterns are concise in the sense that all of the frequent patterns can be derived from these representations. However, they emphasize too much on the supports of patterns so that the compression power is limited.

Our work belongs to the family of Lossy compression methods. Previous works in this direction include maximal patterns [9], error-tolerant patterns [17], δ-free itemsets [4] and boundary cover sets [1]. Typically, our work is close to error-tolerant patterns and δ-free itemsets. Our work is different in that we define a new distance measure and formulate the problem as set-covering. Furthermore, we allow extended (i.e., longer) patterns to represent the compressed patterns, and it leads to stronger compression.

7. Discussions

In this section, we discuss several related issues. First, to approximate a collection of frequent patterns, people always favor the more succinct compression. However, the explosive output pattern size restricts the application of most advanced algorithms. The RPlocal method can be used as sampling procedure as we did in RPcombine, since it is efficient and achieves considerable compression. Second, the compressed pattern sets generated by our method can be used for queries of finding approximate supports. We can construct an RP-tree with all the representative patterns. The query process is similar to the *coverage checking*, except that in *coverage checking*, the query pattern comes with its support, while here, the support is unknown. Among all the patterns which subsume the query pattern, we report the maximum support C. The support of the query pattern is bounded in $[C, \frac{C}{1-\delta}]$. Finally, although our algorithms are developed for frequent itemset problems, the methodology can be easily applied on frequent sequential patterns and graph patterns. This is because our method does not rely on the structure of the patterns, but only the containment relationship. As long as the mining procedures follow the pattern growth principle, we can always integrate RPlocal into the depth-first search.

8. Conclusions

We have considered the problem of compressing frequent patterns. The problem was shown to be NP-Hard. Several methods have been proposed. The RPglobal method has theoretical bound, and works well on small collections of frequent patterns. The RPlocal method is quite efficient, and preserves reasonable compression quality. We also discuss a combined approach, RPcombine, to balance the quality and efficiency.

Acknowledgement

We are grateful to Professor Chengxiang Zhai and his student Qiaozhu Mei for providing us the text dataset and its processing tools.

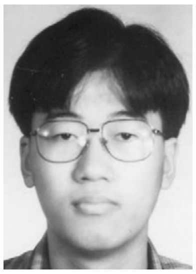
References

[1] F. Afrati, A. Gionis, H. Mannila, Approximating a collection of frequent sets, in Proc. KDD'04, pp. 12–19.
[2] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proc. SIGMOD'93, pp. 207–216.
[3] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proc. ICDE'95, pp. 3–14.
[4] J. Boulicaut, A. Bykowski, C. Rigotti, Free-sets: a condensed representation of Boolean data for the approximation of frequency queries, Data Mining and Knowledge Discovery Journal 7 (1) (2003) 5–22.
[5] S. Brin, R. Motwani, C. Silverstein, Beyond market basket: generalizing association rules to correlations, in: Proc. SIGMOD'97, pp. 265–276.
[6] T. Calders, B. Goethals, Mining all non-derivable frequent itemsets, in: Proc. PKDD'02, pp. 74–85.
[7] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, 2001.
[8] Frequent Itemset Mining Dataset Repository, Available from: <http://fimi.cs.helsinki.fi/data/>.
[9] D. Gunopulos, H. Mannila, R. Khardon, H. Toivonen, Data mining, hypergraph transversals, and machine learning, in: Proc. PODS'97, pp. 209–216.
[10] G. Grahne, J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, in: Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03).
[11] J. Han, G. Dong, Y. Yin, Efficient mining of partial periodic patterns in time series database, in: Proc. ICDE'99, pp. 106–115.
[12] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proc. SIGMOD'00, pp. 1–12.
[13] B. Lent, A. Swami, J. Widom, Clustering association rules, in: Proc. ICDE'97, pp. 220–231.
[14] H. Mannila, H. Towvonen, A. Verkamo, Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery 1 (3) (1997) 259–289.
[15] Q. Mei, C. Zhai, Discovering evolutionary theme patterns from text: an exploration of temporal text mining, in: Proc. KDD'05, pp. 198–207.
[16] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules, in: Proc. ICDT'99, pp. 398–416.

[17] J. Pei, A. Tung, J. Han, Fault-tolerant frequent pattern mining: problems and challenges, in: Proc. DMKD'01.
[18] C. Silverstein, S. Brin, R. Motwani, J. Ullman, Scalable techniques for mining causal structures, in: Proc. VLDB'98, pp. 594–605.
[19] A. Singhal, Modern information retrieval: a brief overview, Bull. IEEE CS Tech. Comm. Data Eng. 24 (4) (2001) 35–43.
[20] M. Zaki, C. Hsiao, Charm: an efficient algorithm for closed itemset mining, in: Proc. SDM'02.

Dong Xin received his BS and MS degrees from the Department of Computer Science and Engineering at Zhejiang University in 1999 and 2002. He is currently a Ph.D. candidate in the Department of Computer Science at University of Illinois at Urbana-Champaign. His research interests include data mining, data warehousing, database system and bioinformatics.

Jiawei Han is a Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He has been researching into data mining, data warehousing, stream data mining, spatiotemporal and multimedia data mining, biological data mining, social network analysis, text and Web mining, and software bug mining, with over 300 conference and journal publications. He has chaired or served in many program committees of international conferences and workshops. He also served or is serving on the editorial boards for Data Mining and Knowledge Discovery, IEEE Transactions on Knowledge and Data Engineering, Journal of Computer Science and Technology, and Journal of Intelligent Information Systems. He is currently serving as founding Editor-in-Chief of ACM Transactions on Knowledge Discovery from Data (TKDD), and on the Board of Directors for the Executive Committee of ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD). Jiawei has received many awards and recognitions, including ACM SIGKDD Innovation Award (2004) and IEEE Computer Society Technical Achievement Award (2005). He is an ACM Fellow (2004).



Xifeng Yan is currently a fifth-year Ph.D. candidate in Department of Computer Science at University of Illinois at Urbana-Champaign. He is working on data mining, structural/graph pattern mining, and their applications in database systems, autonomic computing, and bioinformatics. Mr. Yan has published more than 10 papers in highly refereed journals and conferences such as TODS, SIGMOD, SIGKKDD, VLDB, ISMB, ICDE, and FSE. He has applied for 2 US patents.

Hong Cheng is currently a third year Ph.D. candidate in Department of Computer Science at University of Illinois at Urbana-Champaign. She is working on data mining, including frequent pattern mining and maintenance, pattern summarization and usage, etc. She has published eight research papers in highly refereed conferences such as SIGKDD, VLDB, SDM, ICDM, and ICAPS.