Discovery Science 2004, LNAI 3245, 2004.
IEEE ICDM Workshop FIMI'04 Best implementation award winner (Frequent Itemset Mining Implementation 2004, Zaki & Goethals, Nov. 2004)

An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases

Takeaki Uno[1], Tatsuya Asai[2,⋆], Yuzo Uchida[2], and Hiroki Arimura[2]

[1] National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan
uno@nii.jp
[2] Department of Informatics, Kyushu University, 6-10-1, Hakozaki, Fukuoka, Japan
{t-asai,y-uchida,arim}@i.kyushu-u.ac.jp

Abstract. The class of closed patterns is a well known condensed representations of frequent patterns, and have recently attracted considerable interest. In this paper, we propose an efficient algorithm LCM (Linear time Closed pattern Miner) for mining frequent closed patterns from large transaction databases. The main theoretical contribution is our proposed *prefix-preserving closure extension* of closed patterns, which enables us to search all frequent closed patterns in a depth-first manner, in linear time for the number of frequent closed patterns. Our algorithm do not need any storage space for the previously obtained patterns, while the existing algorithms needs it. Performance comparisons of LCM with straightforward algorithms demonstrate the advantages of our prefix-preserving closure extension.

1 Introduction

Frequent pattern mining is one of the fundamental problems in data mining and has many applications such as association rule mining [1, 5, 7] and condensed representation of inductive queries [12]. To handle frequent patterns efficiently, equivalence classes induced by the occurrences had been considered. Closed patterns are maximal patterns of an equivalence class.

This paper addresses the problems of enumerating all frequent closed patterns. For solving this problem, there have been proposed many algorithms [14, 13, 15, 20, 21]. These algorithms are basically based on the enumeration of frequent patterns, that is, enumerate frequent patterns, and output only those being closed patterns. The enumeration of frequent patterns has been studied well, and can be done in efficiently short time [1, 5]. Many computational experiments supports that the algorithms in practical take very short time per pattern on average.

However, as we will show in the later section, the number of frequent patterns can be exponentially larger than the number of closed patterns, hence the computation time can be exponential in the size of datasets for each closed pattern on average. Hence, the existing algorithms use heuristic pruning to cut off non-closed frequent patterns. However, the pruning are not complete, hence they still have possibilities to take exponential time for each closed pattern.

⋆ Presently working for Fujitsu Laboratory Ltd., e-mail: asai.tatsuya@jp.fujitsu.com

E. Suzuki and S. Arikawa (Eds.): DS 2004, LNAI 3245, pp. 16–31, 2004.

Moreover, these algorithms have to store previously obtained frequent patterns in memory for avoiding duplications. Some of them further use the stored patterns for checking the "closedness" of patterns. This consumes much memory, sometimes exponential in both the size of both the database and the number of closed patterns. In summary, the existing algorithms possibly take exponential time and memory for both the database size and the number of frequent closed patterns. This is not only a theoretical observation but is supported by results of computational experiments in FIMI'03[7]. In the case that the number of frequent patterns is much larger than the number of frequent closed patterns, such as BMS-WebView1 with small supports, the computation time of the existing algorithms are very large for the number of frequent closed patterns.

In this paper, we propose a new algorithm LCM (Linear time Closed pattern Miner) for enumerating frequent closed patterns. Our algorithm uses a new technique called *prefix preserving extension* (ppc-extension), which is an extension of a closed pattern to another closed pattern. Since this extension generates a new frequent closed pattern from previously obtained closed pattern, it enables us to completely prune the unnecessary non-closed frequent patterns. Our algorithm always finds a new frequent closed pattern in linear time of the size of database, but never take exponential time. In the other words, our algorithm always terminates in linear time in the number of closed patterns. Since any closed pattern is generated by the extension from exactly one of the other closed patterns, we can enumerate frequent closed patterns in a depth-first search manner, hence we need no memory for previously obtained patterns. Thereby, the memory usage of our algorithm depends only on the size of input database. This is not a theoretical result but our computational experiments support the practical efficiency of our algorithm. The techniques used in our algorithm are orthogonal to the existing techniques, such as FP-trees, look-ahead, and heuristic preprocessings. Moreover, we can add the existing techniques for saving the memory space so that our algorithm can handle huge databases much larger than the memory size.

For practical computation, we propose *occurrence deliver*, *anytime database reduction*, and *fast ppc-test*, which accelerate the speed of the enumeration significantly. We examined the performance of our algorithm for real world and synthesis datasets taken from FIMI'03 repository, including the datasets used in KDD-cup [11], and compare with straightforward implementations. The results showed that the performance of the combination of these techniques and the prefix preserving extension is good for many kinds of datasets.

In summary, our algorithm has the following advantages.

- Linear time enumeration of closed patterns
- No storage space for previously obtained closed patterns
- Generating any closed pattern from another unique closed pattern
- Depth-first generation of closed patterns
- Small practical computational cost of generation of a new pattern

The organization of the paper is as follows. Section 2 prepares notions and definitions. In the Section 3, we give an example to show the number of frequent patterns can be up to exponential to the number of closed patterns. Section 4 explains the existing schemes for closed pattern mining, then present the prefix-preserving closure extension and our algorithm. Section 5 describes several improvements for practical use. Section 6

presents experimental results of our algorithm and improvements on synthesis and realworld datasets. We conclude the paper in Section 7.

2 Preliminaries

We give basic definitions and results on closed pattern mining according to [1, 13, 6].

Let $\mathcal{I} = \{1, \ldots, n\}$ be the set of *items*. A *transaction database* on $\mathcal{I}$ is a set $\mathcal{T} = \{t_1, \ldots, t_m\}$ such that each t_i is included in $\mathcal{I}$. Each t_i is called a *transaction*. We denote the total size of $\mathcal{T}$ by $||\mathcal{T}||$, i.e., $||\mathcal{T}|| = \sum_{t \in \mathcal{T}} |t|$. A subset P of $\mathcal{I}$ is called a *pattern* (or *itemset*). For pattern P, a transaction including P is called an *occurrence* of P. The *denotation* of P, denoted by $\mathcal{T}(P)$ is the set of the occurrences of P. $|\mathcal{T}(P)|$ is called the *frequency* of P, and denoted by $frq(P)$. For given constant $\theta \in \mathsf{N}$, called a *minimum support*, pattern P is *frequent* if $frq(P) \geq \theta$. For any patterns P and Q, $\mathcal{T}(P \cup Q) = \mathcal{T}(P) \cap \mathcal{T}(Q)$ holds, and if $P \subseteq Q$ then $\mathcal{T}(Q) \supseteq \mathcal{T}(P)$.

Let $\mathcal{T}$ be a database and P be a pattern on $\mathcal{I}$. For a pair of patterns P and Q, we say P and Q are *equivalent* to each other if $\mathcal{T}(P) = \mathcal{T}(Q)$. The relationship induces equivalence classes on patterns. A maximal pattern and a minimal pattern of an equivalence class, w.r.t. set inclusion, are called a *closed pattern* and *key pattern*, respectively. We denote by $\mathcal{F}$ and $\mathcal{C}$ the sets of all frequent patterns and the set of frequent closed patterns in $\mathcal{T}$, respectively.

Given set $\mathcal{S} \subseteq \mathcal{T}$ of transactions, let $\mathcal{I}(\mathcal{S}) = \bigcap_{T \in \mathcal{S}} T$ be the set of items common to all transactions in $\mathcal{S}$. Then, we define the *closure* of pattern P in $\mathcal{T}$, denoted by $Clo(P)$, by $\bigcap_{T \in \mathcal{T}(P)} T$. For every pair of patterns P and Q, the following properties hold **(Pasquier et al.[13])**.

(1) If $P \subseteq Q$, then $Clo(P) \subseteq Clo(Q)$.
(2) If $\mathcal{T}(P) = \mathcal{T}(Q)$, then $Clo(P) = Clo(Q)$.
(3) $Clo(Clo(P)) = Clo(P)$.
(4) $Clo(P)$ is the unique smallest closed pattern including P.
(5) A pattern P is a closed pattern if and only if $Clo(P) = P$.

Note that a key pattern is not the unique minimal element of an equivalence class, while the closed pattern is unique. Here we denote the set of frequent closed patterns by $\mathcal{C}$, the set of frequent patterns by $\mathcal{F}$, the set of items by $\mathcal{I}$, and the size of database by $||\mathcal{T}||$.

For pattern P and item $i \in P$, let $P(i) = P \cap \{1, \ldots, i\}$ be the subset of P consisting only of elements no greater than i, called the *i-prefix* of P. Pattern Q is a *closure extension* of pattern P if $Q = Clo(P \cup \{i\})$ for some $i \notin P$. If Q is a closure extension of P, then $Q \supset P$, and $frq(Q) \leq frq(P)$.

3 Difference Between Numbers of Frequent Patterns and Frequent Closed Patterns

This section shows that the the number of frequent patterns can be quite larger than the number of frequent closed patterns. To see it, we prove the following theorem. Here an irredundant transaction database is a transaction database such that

- no two transactions are the same,
- no item itself is an infrequent pattern, and
- no item is included in all transactions.

Intuitively, if these conditions are satisfied, then we can neither contract nor reduce the database.

Theorem 1. *There are infinite series of irredundant transaction databases $\mathcal{T}$ such that the number $|\mathcal{F}|$ of frequent patterns is exponential in m and n while the number $|\mathcal{C}|$ of frequent closed patterns is $O(m^2)$, where m is the number of transactions in $\mathcal{T}$ and n is the size of the itemset on which the transactions are defined. In particular, the size of $\mathcal{T}$ is $\Theta(nm)$.*

Proof. Let n be any number larger than 4 and m be any even number satisfying that $n - (\lceil \lg m \rceil + 2)$ is larger than both n^ϵ and m^ϵ for a constant ϵ. Let

$X = \{1, ..., n - 2(\lceil \lg m \rceil + 2)\}$
$Y_1 = \{n - 2(\lceil \lg m \rceil + 2) + 1, ..., n - (\lceil \lg m \rceil + 2)\}$
$Y_2 = \{n - (\lceil \lg m \rceil + 2) + 1, ..., n\}$
$\mathcal{J}_1 :=$ a subset of $2^{Y_1} \setminus \{\emptyset, Y_1\}$ of size $m/2 - 1$ without duplications
$\mathcal{J}_2 :=$ a subset of $2^{Y_2} \setminus \{\emptyset, Y_2\}$ of size $m/2 - 1$ without duplications.

Since $|Y_1| = |Y_2| = \lceil \lg m \rceil + 2$, such $\mathcal{J}_1$ and $\mathcal{J}_2$ always exist. Then, we construct a database as follows.

$$\mathcal{T} := \begin{array}{l} \{X \cup Y_1 \cup S | S \in \mathcal{J}_2\} \\ \bigcup \{X \cup Y_2 \cup S | S \in \mathcal{J}_1\} \\ \bigcup \{Y_1 \cup Y_2\} \\ \bigcup \{X\} \end{array}$$

The database has m transactions defined on an itemset of size n. The size of database is $\Theta(nm)$.

We set the minimum support to $m/2$. We can see that no transactions are the same, no item is included in all transactions, and any item is included in at least $m/2$ transactions. Let $\mathcal{F}$ and $\mathcal{C}$ be the set of frequent patterns and the set of frequent closed patterns, respectively.

Since any transaction in $\mathcal{T}$ except for $Y_1 \cup Y_2$ includes X, any subset of X is a frequent pattern. Since $|X| = n - (\lceil \lg m \rceil + 2)$ is larger than both n^ϵ and m^ϵ, we see that $|\mathcal{F}|$ is exponential in both n and m.

On the other hand, any frequent closed pattern includes X. Hence, any frequent closed pattern is equal to $X \cup S$ for some $S \subseteq Y_1 \cup Y_2$. Since $|Y_1 \cup Y_2| = 2(\lceil \lg m \rceil + 2)$, we have

$$\begin{aligned} |\mathcal{C}| &\leq 2^{2(\lceil \lg m \rceil + 2)} \\ &= 2^{2 \lg m + 6} \\ &= 64m^2. \end{aligned}$$

Therefore, $|\mathcal{C}| = \mathrm{O}(m^2)$. □

From the theorem, we can see that frequent pattern mining based algorithms can take exponentially longer time for the number of closed patterns. Note that such patterns may appear in real world data in part, because some transactions may share a common large pattern.

4 Algorithm for Enumerating Closed Patterns

We will start with the existing schemes for closed pattern enumeration.

4.1 Previous Approaches for Closed Pattern Mining

A simple method of enumerating frequent closed patterns is to enumerate all frequent patterns, classify them into equivalence classes, and find the maximal pattern for each equivalent class. This method needs $O(|\mathcal{F}|^2)$ time and $O(|\mathcal{F}|)$ memory, since pairwise comparisons of their denotations are needed to classify frequent patterns. Although the number of comparisons can be decreased by using some alphabetical sort of denotations such as radix sort, it still needs $O(|\mathcal{F}|)$ computation. As we saw, $|\mathcal{F}|$ is possibly quite larger than $|\mathcal{C}|$, hence this method consumes a great deal of time and memory.

Some state-of-the-art algorithms for closed pattern mining, such as CHARM [21] and CLOSET [15], use heuristic pruning methods to avoid generating unnecessary non-closed patterns. Although these pruning methods efficiently cut off non-closed pattens, the number of generated patterns is not bounded by a polynomial of $|\mathcal{C}|$.

Pasquier et al.[13] proposed the use of closure operation to enumerate closed patterns. Their idea is to generate frequent patterns, and check whether the patterns are closed patterns or not by closure operation. Although this reduces the storage space for non-closed patterns, the algorithm still requires $|\mathcal{C}|$ space. They actually generate frequent key patterns instead of frequent patterns, to reduce the computational costs. Thus, the computation time is linear in the number of frequent key patterns, which is less than $|\mathcal{F}|$ but can be up to exponential in $|\mathcal{C}|$.

4.2 Closed Patterns Enumeration in Linear Time with Small Space

Our algorithm runs in linear time in $|\mathcal{C}|$ with storage space significantly smaller than that of Pasquier et al.'s algorithm [13], since it operates no non-closed patterns using depth-first search The following lemmas provide a way of efficiently generating any closed pattern from another closed pattern. In this way, we construct our basic algorithm. Then, the basic algorithm is improved to save the memory use by using ppc extension.

Lemma 1. *Let P and Q, $P \subseteq Q$ be patterns having the same denotation, i.e., $\mathcal{T}(P) = \mathcal{T}(Q)$. Then, for any item $i \notin P$, $\mathcal{T}(P \cup \{i\}) = \mathcal{T}(Q \cup \{i\})$.*

Proof. $\mathcal{T}(P \cup \{i\}) = \mathcal{T}(P) \cap \mathcal{T}(\{i\}) = \mathcal{T}(Q) \cap \mathcal{T}(\{i\}) = \mathcal{T}(Q \cup \{i\})$. □

Lemma 2. *Any closed pattern $P \neq \perp$ is a closure extension of other closed patterns.*

Algorithm Closure_version
1. $\mathcal{D} := \{\perp\}$
2. $\mathcal{D}' := \{\ Clo(P \cup \{i\}) \mid P \in \mathcal{D}, i \in \mathcal{I} \backslash P\ \}$
3. **if** $\mathcal{D}' = \emptyset$ **then output** $\mathcal{D}$; **halt**
4. $\mathcal{D} := \mathcal{D} \cup \mathcal{D}'$; **go to** 2

Fig. 1. Basic algorithm for enumerating frequent closed patterns.

Proof. Let Q be a pattern obtained by repeatedly removing items from P until its denotation changes, and i be the item removed last. Then, $Clo(Q \cup \{i\}) = P$. Such a pattern must exist since $P \neq \perp$. Since $\mathcal{T}(Q) \neq \mathcal{T}(Q \cup \{i\})$, $i \notin Clo(Q)$. From Property 1, $Clo(Q \cup \{i\}) = Clo(Clo(Q) \cup \{i\})$. Thus, P is a closure extension of $Q \cup \{i\}$. □

Through these lemmas, we can see that all closed patterns can be generated by closure extensions to closed patterns. It follows the basic version of our algorithm, which uses levelwise (breadth-first) search similar to Apriori type algorithms [1] using closed expansion instead of tail expansion. We describe the basic algorithm in Figure 1. Since the algorithm deals with no non-closed pattern, the computational cost depends on $|\mathcal{C}|$ but not on $|\mathcal{F}|$. However, we still need much storage space to keep $\mathcal{D}$ in memory.

A possible improvement is to use depth-first search instead of Apriori-style levelwise search. For enumerating frequent patterns, Bayardo [5] proposed an algorithm based on *tail extension*, which is an extension of a pattern P by an item larger than the maximum item of P. Since any frequent pattern is a tail extension of another unique frequent pattern, the algorithm enumerates all frequent patterns without duplications in a depth-first manner, with no storage space for previously obtained frequent patterns. This technique is efficient, but cannot directly be applied to closed pattern enumeration, since a closed pattern is not always a tail-extension of another closed pattern.

We here propose *prefix-preserving closure extension* satisfying that any closed pattern is an extension of another unique closed pattern unifying ordinary closure-expansion and tail-expansion. This enables depth-first generation with no storage space.

4.3 Prefix-Preserving Closure Extension

We start with definitions. Let P be a closed pattern. The *core index* of P, denoted by $core\_i(P)$, is the minimum index i such that $\mathcal{T}(P(i)) = \mathcal{T}(P)$. We let $core\_i(\perp) = 0$.

Here we give the definition of ppc-extension. Pattern Q is called a *prefix-preserving closure extension* (*ppc-extension*) of P if

(i) $Q = Clo(P \cup \{i\})$ for some $i \in P$, that is, Q is obtained by first adding i to P and then taking its closure,
(ii) item i satisfies $i \notin P$ and $i > core\_i(P)$, and
(iii) $P(i-1) = Q(i-1)$, that is, the $(i-1)$-prefix of P is preserved.

Actually, ppc-extension satisfies the following theorem. We give an example in Fig. 2.

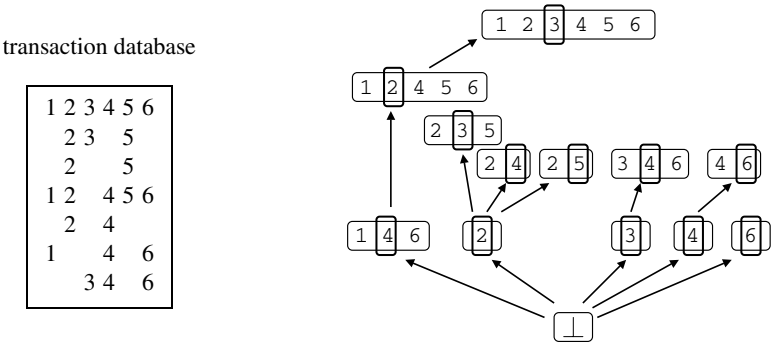

Fig. 2. Example of all closed patterns and their ppc extensions. Core indices are circled.

Theorem 2. *Let $Q \neq \perp$ be a closed pattern. Then, there is just one closed pattern P such that Q is a ppc-extension of P.*

To prove the theorem, we state several lemmas.

Lemma 3. *Let P be a closed pattern and $Q = Clo(P \cup \{i\})$ be a ppc-extension of P. Then, i is the core index of Q.*

Proof. Since $i > core\_i(P)$, we have $\mathcal{T}(P) = \mathcal{T}(P(i))$. From Lemma 2, $Clo(P(i) \cup \{i\}) = Clo(P \cup \{i\}) = Clo(Q)$, thus $core\_i(Q) \leq i$. Since the extension preserves the i-prefix of P, we have $P(i-1) = Q(i-1)$. Thus, $Clo(Q(i-1)) = Clo(P(i-1)) = P \neq Q$. It follows that $core\_i(Q) > i-1$, and we conclude $core\_i(Q) = i$. □

Let Q be a closed pattern and $\mathcal{P}(Q)$ be the set of closed patterns such that Q is their closure extension. We show that Q is a ppc-extension of a unique closed pattern of $\mathcal{P}(Q)$.

Lemma 4. *Let $Q \neq \perp$ be a closed pattern, and $P = Clo(Q(core\_i(Q) - 1))$. Then, Q is a ppc-extension of P.*

Proof. Since $\mathcal{T}(P) = \mathcal{T}(Q(core\_i(Q)-1))$, we have $\mathcal{T}(P \cup \{i\}) = \mathcal{T}(Q(core\_i(Q)-1) \cup \{i\}) = \mathcal{T}(Q(core\_i(Q)))$. This implies $Q = Clo(P \cup \{i\})$, thus Q satisfies condition (i) of ppc-extension. Since $P = Clo(Q(core\_i(Q) - 1))$, $core\_i(P) \leq i-1$. Thus, Q satisfies condition (ii) of ppc-extension. Since $P \subset Q$ and $Q(i-1) \subseteq P$, we have $P(i-1) = Q(i-1)$. Thus, Q satisfies condition (iii) of ppc-extension. □

Proof of Theorem 2: From Lemma 4, there is at least one closed pattern P in $\mathcal{P}(Q)$ such that Q is a ppc-extension of P. Let $P = Clo(Q(core\_i(Q) - 1))$. Suppose that there is a closed pattern $P' \neq P$ such that Q is a ppc-extension of P'. From lemma 3, $Q = Clo(P' \cup \{i\})$. Thus, from condition (iii) of ppc-extension, $P'(i-1) = Q(i-1) = P(i-1)$. This together with $\mathcal{T}(P) = \mathcal{T}(P(i-1))$ implies that $\mathcal{T}(P) \supset \mathcal{T}(P')$. Thus, we can see $\mathcal{T}(P'(i-1)) \neq \mathcal{T}(P')$, and $core\_i(P') \geq i$. This violates condition (ii) of ppc-extension, and is a contradiction. □

```
Algorithm LCM(T:transaction database, θ:support )
  1. call ENUM_CLOSEDPATTERNS(⊥) ;
Procedure ENUM_CLOSEDPATTERNS( P: frequent closed pattern )
  2. if P is not frequent then Return;
  2. output P;
  3. for i = core_i(P) + 1 to |I|
  4.   Q = Clo(P ∪ {i});
  5.   If P(i − 1) = Q(i − 1) then      // Q is a ppc-extension of P
  6.     Call ENUM_CLOSEDPATTERNS(Q);
  7. End for
```

Fig. 3. Description of Algorithm LCM.

From this theorem, we obtain our algorithm LCM, described in Figure 3, which generate ppc-extensions for each frequent closed pattern.

Since the algorithm takes $O(||\mathcal{T}(P)||)$ time to derive the closure of each $P \cup \{i\}$, we obtain the following theorem.

Theorem 3. *Given a database $\mathcal{T}$, the algorithm LCM enumerates all frequent closed patterns in $O(||\mathcal{T}(P)|| \times |\mathcal{I}|)$ time for each pattern P with $O(||\mathcal{T}||)$ memory space.*

The time and space complexities of the existing algorithms [21, 15, 13] are $O(||\mathcal{T}|| \times |\mathcal{F}|)$ and $O(||\mathcal{T}|| + |\mathcal{C}| \times |\mathcal{I}|)$, respectively. As we saw in the example in Section 3, the difference between $|\mathcal{C}|$ and $|\mathcal{F}|$, and the difference between $|\mathcal{C}| \times |\mathcal{I}|$ and $||\mathcal{T}||$ can be up to exponential. As compared with our basic algorithm, the ppc extension based algorithm exponentially reduces the memory complexity when $O(|\mathcal{C}|)$ is exponentially larger than $||\mathcal{T}||$. In practice, such exponential differences often occur (see results in Section 6). Thus, the performance of our algorithm possibly exponentially better than the existing algorithms in some instances.

5 Reducing Practical Computation Time

The computation time of LCM described in the previous section is linear in $|\mathcal{C}|$, with a factor depending on $||\mathcal{T}|| \times |\mathcal{I}|$ for each closed pattern $P \in \mathcal{C}$. However, this still takes a long time if it is implemented in a straightforward way. In this section, we propose some techniques for speeding up frequency counting and closure operation. These techniques will increase practical performance of the algorithms and incorporated into the implementations used in the experiments in Section 6 of the paper, although independent of our main contribution. In Figure 7, we describe the details of LCM with these practical techniques.

5.1 Occurrence Deliver

Occurrence deliver reduces the construction time for $\mathcal{T}(P \cup \{i\})$, which is used for frequency counting and closure operation. This technique is particularly efficient for

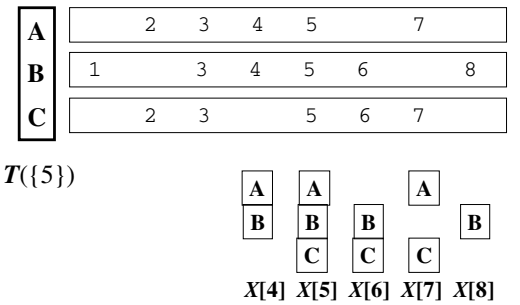

Fig. 4. Occurrence deliver: build up denotations by inserting each transaction to each its member.

sparse datasets, such that $|\mathcal{T}(P \cup \{i\})|$ is much smaller than $|\mathcal{T}(P)|$ on average. In a usual way, $\mathcal{T}(P \cup \{i\})$ is obtained by $\mathcal{T}(P) \cap \mathcal{T}(P \cup \{i\})$ in $O(|\mathcal{T}(P)| + |\mathcal{T}(P \cup \{i\})|)$ time (this is known as *down-project* []). Thus, generating all ppc-extensions needs $|\mathcal{I}|$ scans and takes O($||\mathcal{T}(P)||$) time.

Instead of this, we build for all $i = core\_i(P), \ldots, |\mathcal{I}|$ denotations $\mathcal{T}(P \cup \{i\})$, simultaneously, by scanning the transactions in $\mathcal{T}(P)$. We initialize $X[i] := \emptyset$ for all i. For each $t \in \mathcal{T}(P)$ and for each $i \in t; i > core\_i(P)$, we insert t to $X[i]$. Then, each $X[i]$ is equal to $\mathcal{T}(P \cup \{i\})$. See Fig. 4 for the explanation. This correctly computes $\mathcal{T}(P \cup \{i\})$ for all i in O($||\mathcal{T}(P)||$). Table 1 shows results of computational experiments where the number of item-accesses were counted. The numbers are deeply related to the performance of frequency counting heavily depending on the number of item-accesses.

Table 1. The accumulated number of item-accesses over all iterations.

Dataset and support	connect,65%	pumsb,80%,	BMS-webview2,0.2%	T40I10D100K,5%
Straightforward	914074131	624940309	870850439	830074845
Occurrence deliver	617847782	201860874	17217493	73406900
Reduction factor	1.47	3.09	50.5	11.3

5.2 Anytime Database Reduction

In conventional practical computations of frequent pattern mining, the size of the input database is reduced by removing infrequent items i such that $\mathcal{T}(\{i\}) < \theta$, and then merging the same transactions into one. Suppose that we are given the database on the left hand side of Fig. 5, and minimum support 3.

In the database, items 4, 6, 7, and 8 are included in at most two transactions, hence patterns including these items can not be frequent. Items 1 and 2 are included in all transactions, hence any frequent patterns can contain any subset of $\{1, 2\}$. It follows that items 1 and 2 are redundant. Hence, we remove items 1, 2, 4, 6, 7, and 8 from the database. After the removal, we merge the same transactions into one. We then record the multiplicity of transactions. Consequently, we have a smaller database, right hand side of Fig. 5, with fewer items and fewer transactions, which includes the same set

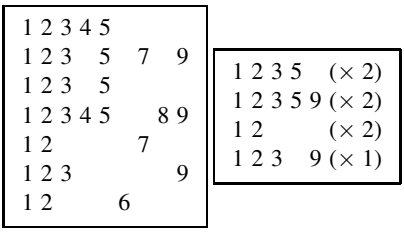

Fig. 5. Example of anytime database reduction.

of frequent patterns. We call this operation *database reduction* (or *database filtering*). Many practical algorithms utilize this operation at initialization. Database reduction is known to be efficient in practice, especially for large support [7].

Here we propose *anytime database reduction*, which is to apply database reduction in any iteration. For an iteration (invokation) of Enum_ClosedPatterns inputting pattern P, the following items and transactions are unnecessary for any of its descendant iterations:

(1) transactions not including P,
(2) transactions including no item greater than $core\_i(P)$,
(3) items included in P,
(4) items less than or equal to $core\_i(P)$, and
(5) items i satisfies that $frq(P \cup \{i\}) < \theta$ (no frequent pattern includes i).

In each iteration, we restrict the database to $\mathcal{T}(P)$, apply database reduction to remove such items and transactions, merge the same transactions, and pass it to child iterations.

Anytime database reduction is efficient especially when support is large. In the experiments, in almost all iterations, the size of the reduced database is bounded by a constant even if the patterns have many occurrences. Table 2 lists simple computational experiments comparing conventional database reduction (applying reduction only at initialization) and anytime database reduction. Each cell shows the accumulated number of transactions in the reduced databases in all iterations.

Table 2. Accumulated number of transactions in database in all iterations.

Dataset and support	connect, 50%	pumsb, 60%,	BMS-WebView2, 0.1%	T40I10D100K, 0.03%
Database reduction	188319235	2125460007	2280260	1704927639
Anytime database reduction	538931	7777187	521576	77371534
Reduction factor	349.4	273.2	4.3	22.0

We also use anytime database reduction for closure operation. Suppose that we have closed pattern P with core index i. Let transactions $t_1, ..., t_k \in \mathcal{T}(P)$ have the same i-suffix, i.e., $t_1 \cap \{i, ..., |\mathcal{I}|\} = t_2 \cap \{i, ..., |\mathcal{I}|\} =, ..., = t_k \cap \{i, ..., |\mathcal{I}|\}$.

Lemma 5. *Let $j; j < i$ be an item, and j' be items such that $j' > i$ and included in all $t_1, ..., t_k$. Then, if j is not included in at least one transaction of $t_1, ..., t_k$, $j \notin Clo(Q)$ holds for any pattern Q including $P \cup \{i\}$.*

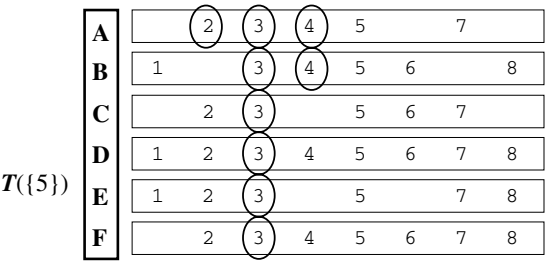

Fig. 6. Transaction A has the minimum size. Fast ppc test accesses only circled items while closure operation accesses all items.

Proof. Since $\mathcal{T}(Q)$ includes all $t_1, ..., t_k$, and j is not included in one of $t_1, ..., t_k$. Hence, j is not included in $Clo(Q)$. □

According to this lemma, we can remove j from $t_1, ..., t_k$ from the database if j is not included in at least one of $t_1, ..., t_k$. By removing all such items, $t_1, ..., t_k$ all become $\bigcap_{h=1,...,k} t_h$. Thus, we can merge them into one transaction similar to the above reduction. This reduces the number of transactions as much as the reduced database for frequency counting. Thus, the computation time of closure operation is shorten drastically. We describe the details on anytime database reduction for closure operation.

1. Remove transactions not including P
2. Remove items i such that $frq(P \cup \{i\}) < \theta$
3. Remove items of P
4. Replace the transactions $T_1, ..., T_k$ having the same i-suffix by the intersection, i.e., replace them by $\bigcap\{T_1, ..., T_k\}$.

5.3 Fast Prefix-Preserving Test

Fast prefix-preserving test (fast ppc-test) efficiently checks condition (iii) of ppc-extension for $P \cup \{i\}$. A straightforward way for this task is to compute the closure of $P \cup \{i\}$. This usually takes much time since it accesses all items in the occurrences of P. Instead of this, we check for each j whether $j; j < i, j \notin P(i-1)$ is included in $Clo(P \cup \{i\})$ or not. Item j is included in $Clo(P \cup \{i\})$ if and only if j is included in every transaction of $\mathcal{T}(P \cup \{i\})$. If we find a transaction not including j, then we can immediately conclude that j is not included in every transaction of $\mathcal{T}(P)$. Thus, we do not need to access all items in occurrences. In particular, we have to check items j in occurrence t^* of the minimum size, since other items can not be included in every transaction of $\mathcal{T}(P)$. Fig. 6 shows an example of accessed items by fast ppc test.

This results $O(\sum_{j \in t^* \setminus P} |\mathcal{T}(P \cup \{i\} \cup \{j\})|)$ time algorithm, which is much faster than the straightforward algorithm with $O(\sum_{j<i} |\mathcal{T}(P \cup \{i\} \cup \{j\})|)$ time. Table 3 shows the results of computational experiments comparing the number of accesses between closure operation and fast ppc test.

Table 3. Accumulated number of accessed items in all iterations.

Dataset and support	connect,60%	pumsb,75%,	BMS-WebVeiw2,0.1%	T40I10D100K,0.1%
Closure operation	1807886455	741205890	50313395	2093327534
Fast ppc test	2333551	2703318	127722	1701748
Reduction factor	774.7	274.1	393.9	1230

Algorithm LCM ($\mathcal{T}$:transaction database, θ:support)
1. **call** ENUM_CLOSEDPATTERNS$(\mathcal{T}, \perp, \mathcal{T})$;

Procedure ENUM_CLOSEDPATTERNS ($\mathcal{T}$: transaction database,
P:frequent closed pattern, Occ: transactions including P)
2. **Output** P;
3. Reduce $\mathcal{T}$ by *Anytime database reduction*;
4. Compute the frequency of each pattern $P \cup \{i\}, i > core\_i(P)$
by *Occurrence deliver* with P and Occ ;
5. **for** $i := core\_i(P) + 1$ **to** $|\mathcal{I}|$
6. $Q := Clo(P \cup \{i\})$;
7. **If** $P(i-1) = Q(i-1)$ and Q is frequent **then** // Q is a ppc-extension of P
Call ENUM_CLOSEDPATTERNS(Q);
8. **End for**

The version of fast-ppc test is obtained by replacing the lines 6 and 7 by
6. **if** *fast-ppc test* is true for $P \cup \{i\}$ and $P \cup \{i\}$ is frequent **then**
Call ENUM_CLOSEDPATTERNS(Q);

Fig. 7. Algorithm LCM with practical speeding up.

In fact, the test requires an adjacency matrix (sometimes called a *bitmap*) representing the inclusion relation between items and transactions. The adjacency matrix requires O($|\mathcal{T}| \times |\mathcal{I}|$) memory, which is quite hard to store for large instances. Hence, we keep columns of the adjacency matrix for only transactions larger than $|\mathcal{I}|/\delta$, where δ is a constant number. In this way, we can check whether $j \in t$ or not in constant time if j is large, and also in short time by checking items of t if t is small. The algorithm uses O($\delta \times ||\mathcal{T}||$) memory, which is linear in the input size.

6 Computational Experiments

This section shows the results of computational experiments for evaluating the practical performance of our algorithms on real world and synthetic datasets. Fig. 8 lists the datasets, which are from the FIMI'03 site (`http://fimi.cs.helsinki.fi/`): retail, accidents; IBM Almaden Quest research group website (T10I4D100K); UCI ML repository (connect, pumsb); (at `http://www.ics.uci.edu/~mlearn/MLRepository.html`) Click-stream Data by Ferenc Bodon (kosarak); KDD-CUP 2000 [11] (BMS-WebView-1, BMS-POS) (at `http://www.ecn.purdue.edu/KDDCUP/`).

Dataset	#items	#Trans	AvTrSz	$\|\mathcal{F}\|$	$\|\mathcal{C}\|$	support (%)
BMS-POS	1,657	517,255	6.5	122K–33,400K	122K–21,885K	0.64–0.01
BMS-Web-View1	497	59,602	2.51	3.9K–NA	3.9K–1,241K	0.1–0.01
T10I4D100K	1,000	100,000	10.0	15K–335K	14K–229K	0.15–0.025
kosarak	41,270	990,000	8.10	0.38K–56,006K	0.38K–17,576K	1–0.08
retail	16,470	88,162	10.3	10K–4,106K	10K–732K	0.32–0.005
accidents	469	340,183	33.8	530–10,692K	530–9,959K	70–10
pumsb	7,117	49,046	74.0	2.6K–NA	1.4K–44,453K	95–60
connect	130	67,577	43.0	27K–NA	3.4K–8,037K	95–40

Fig. 8. Datasets: AvTrSz means average transaction size.

To evaluate the efficiency of ppc extension and practical improvements, we implemented several algorithms as follows.

- freqset: algorithm using frequent pattern enumeration
- straight: straightforward implementation of LCM (frequency counting by tuples)
- occ: LCM with occurrence deliver
- occ+dbr: LCM with occurrence deliver and anytime database reduction for both frequency counting and closure
- occ+fchk: LCM with occurrence deliver, anytime database reduction for frequency counting, and fast ppc test

The figure also displays the number of frequent patterns and frequent closed patterns, which are written as #freqset and #freq closed. The algorithms were implemented in C and compiled with gcc 3.3.1. The experiments were run on a notebook PC with mobile Pentium III 750MHz, 256MB memory. Fig. 9 plots the running time with varying minimum supports for the algorithms on the eight datasets.

From Fig. 9, we can observe that LCM with practical optimization (occ, occ+dbr, occ+fchk) outperforms the frequent pattern enumeration-based algorithm (freqset). The speed up ratio of the ppc extension algorithm (straight) against algorithm freqset totally depends on the ratio of #freqset and #freq. closed. The ratio is quite large for several real world and synthetic datasets with small supports, such as BMS-WebView, retails, pumsb, and connect. On such problems, sometimes the frequent pattern based algorithm takes quite long time while LCM terminates in short time.

Occurrence deliver performs very well on any dataset, especially on sparse datasets, such as BMS-WebView, retail, and IBM datasets. In such sparse datasets, since $\mathcal{T}(\{i\})$ is usually larger than $\mathcal{T}(P \cup \{i\})$, occurrence deliver is efficient.

Anytime database reduction decreases the computation time well, especially in dense datasets or those with large supports. Only when support is very small, closed to zero, the computation time were not shorten, since a few items are eliminated and few transactions become the same. In such cases, fast ppc test performs well. However, fast ppc test does not accelerate speed so much in dense datasets or large supports.

For a detailed study about the performance of our algorithm compared with other algorithms, consult the companion paper [18] and the competition report of FIMI'03 [7]. Note that the algorithms we submitted to [18, 7] were old versions, which does not include anytime database reduction, thus they are slower than the algorithm in this paper.

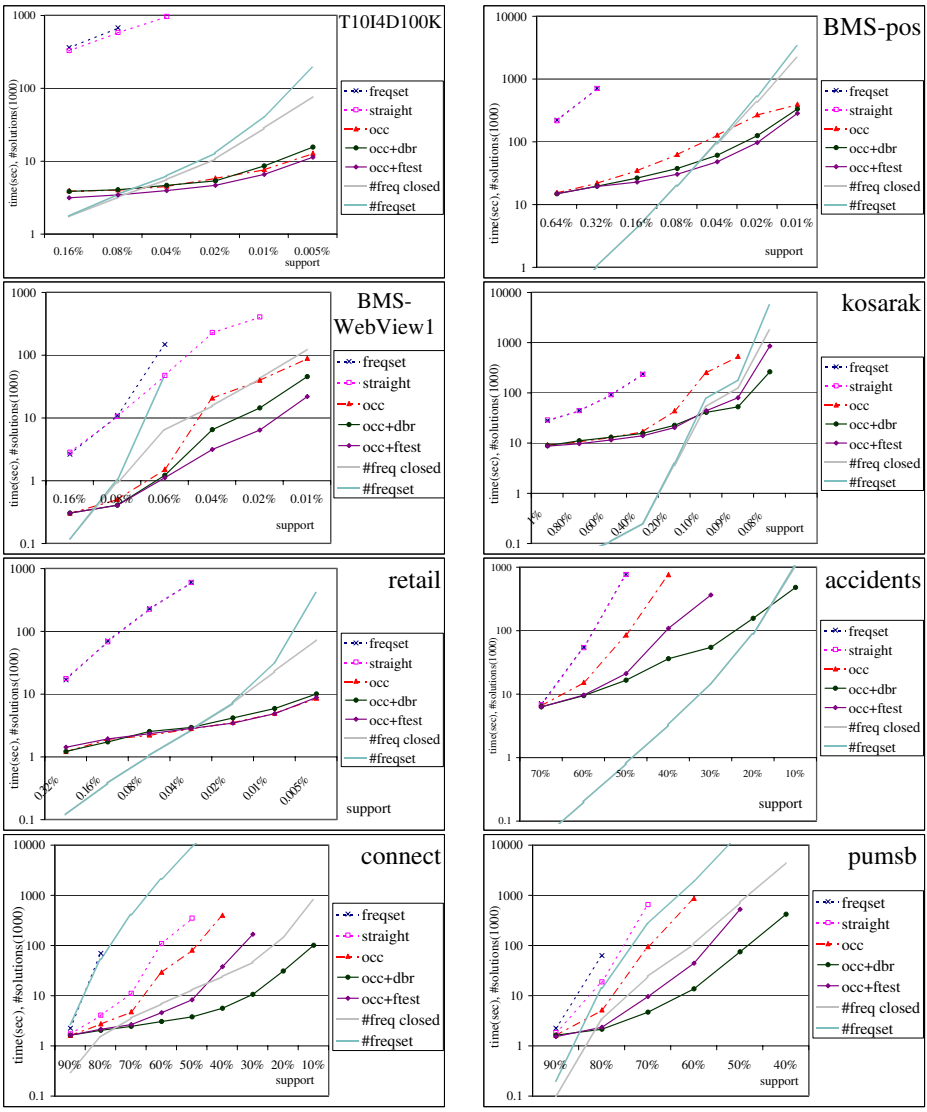

Fig. 9. Computation time for datasets.

7 Conclusion

We addressed the problem of enumerating all frequent closed patterns in a given transaction database, and proposed an efficient algorithm LCM to solve this, which uses memory linear in the input size, i.e., the algorithm does not store the previously obtained patterns in memory. The main contribution of this paper is that we proposed prefix-preserving closure extension, which combines tail-extension of [5] and closure operation of [13] to realize direct enumeration of closed patterns.

We recently studied frequent substructure mining from ordered and unordered trees based on a deterministic tree expansion technique called the *rightmostexpansion* [2–4]. There have been also pioneering works on closed pattern mining in sequences and graphs [17, 19]. It would be an interesting future problem to extend the framework of prefix-preserving closure extension to such tree and graph mining.

Acknowledgment

We would like to thank Professor Ken Satoh of National Institute of Informatics and Professor Kazuhisa Makino of Osaka University for fruitful discussions and comments on this issue. This research was supported by group research fund of National Institute of Informatics, Japan. We are also grateful to Professor Bart Goethals, and people supporting FIMI'03 Workshop/Repository, and the authors of the datasets for the datasets available by the courtesy of them.

References

1. R. Agrawal,H. Mannila,R. Srikant,H. Toivonen,A. I. Verkamo, *Fast Discovery of Association Rules*, In *Advances in Knowledge Discovery and Data Mining,* MIT Press, 307–328, 1996.
2. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, *Efficient Substructure Discovery from Large Semi-structured Data*, In *Proc. SDM'02*, SIAM, 2002.
3. T. Asai, H. Arimura, K. Abe, S. Kawasoe, S. Arikawa, *Online Algorithms for Mining Semi-structured Data Stream*, In *Proc. IEEE ICDM'02*, 27–34, 2002.
4. T. Asai, H. Arimura, T. Uno, S. Nakano, *Discovering Frequent Substructures in Large Unordered Trees*, In *Proc. DS'03*, 47–61, LNAI 2843, 2003.
5. R. J. Bayardo Jr., *Efficiently Mining Long Patterns from Databases*, In Proc. SIGMOD'98, 85–93, 1998.
6. Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal, *Mining Frequent Patterns with Counting Inference*, *SIGKDD Explr.*, 2(2), 66–75, Dec. 2000.
7. B. Goethals, *the FIMI'03 Homepage*, `http://fimi.cs.helsinki.fi/`, 2003.
8. E. Boros, V. Gurvich, L. Khachiyan, K. Makino, *On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets*, In *Proc. STACS 2002*, 133-141, 2002.
9. D. Burdick, M. Calimlim, J. Gehrke, *MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases*, In *Proc. ICDE 2001*, 443-452, 2001.
10. J. Han, J. Pei, Y. Yin, *Mining Frequent Patterns without Candidate Generation*, In *Proc. SIGMOD'00*, 1-12, 2000
11. R. Kohavi, C. E. Brodley, B. Frasca, L. Mason, Z. Zheng, *KDD-Cup 2000 Organizers' Report: Peeling the Onion*, *SIGKDD Explr.*, 2(2), 86-98, 2000.
12. H. Mannila, H. Toivonen, *Multiple Uses of Frequent Sets and Condensed Representations*, In *Proc. KDD'96*, 189–194, 1996.
13. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, *Efficient Mining of Association Rules Using Closed Itemset Lattices*, *Inform. Syst.*, 24(1), 25–46, 1999.
14. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, *Discovering Frequent Closed Itemsets for Association Rules*, In *Proc. ICDT'99*, 398-416, 1999.
15. J. Pei, J. Han, R. Mao, *CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets*, In *Proc. DMKD'00*, 21-30, 2000.
16. R. Rymon, *Search Through Systematic Set Enumeration*, In *Proc. KR-92*, 268–275, 1992.

17. P. Tzvetkov, X. Yan, and J. Han, *TSP: Mining Top-K Closed Sequential Patterns*, In *Proc. ICDM'03*, 2003.
18. T. Uno, T. Asai, Y. Uchida, H. Arimura, *LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets*, In *Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003. (Available as CEUR Workshop Proc. series, Vol. 90, http://ceur-ws.org/vol-90)
19. X. Yan and J. Han, *CloseGraph: Mining Closed Frequent Graph Patterns*, In *Proc. KDD'03*, ACM, 2003.
20. M. J. Zaki, *Scalable Algorithms for Association Mining*, *Knowledge and Data Engineering*, 12(2), 372–390, 2000.
21. M. J. Zaki, C. Hsiao, CHARM: An Efficient Algorithm for Closed Itemset Mining, In *Proc. SDM'02*, SIAM, 457-473, 2002.