CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets *

Jian Pei, Jiawei Han, and Runying Mao
Intelligent Database Systems Research Lab.
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
E-mail: {peijian, han, rmao}@cs.sfu.ca

Abstract

Association mining may often derive an undesirably large set of frequent itemsets and association rules. Recent studies have proposed an interesting alternative: mining frequent closed itemsets and their corresponding rules, which has the same power as association mining but substantially reduces the number of rules to be presented.

In this paper, we propose an efficient algorithm, CLOSET, for mining closed itemsets, with the development of three techniques: (1) applying a compressed, frequent pattern tree FP-tree structure for mining closed itemsets without candidate generation, (2) developing a single prefix path compression technique to identify frequent closed itemsets quickly, and (3) exploring a partition-based projection mechanism for scalable mining in large databases. Our performance study shows that CLOSET is efficient and scalable over large databases, and is faster than the previously proposed methods.

1 Introduction

It has been well recognized that frequent pattern mining plays an essential role in many important data mining tasks, e.g. associations [2, 7], sequential patterns [3], episodes [8], partial periodicity [5], etc. However, it is also well known that frequent pattern mining often generates a very large number of frequent itemsets and rules, which reduces not only efficiency but also effectiveness of mining since users have to sift through a large number of mined rules to find useful ones.

There is an interesting alternative, proposed recently by Pasquier et al. [9]: *instead of mining the complete set of frequent itemsets and their associations, association mining only needs to find frequent closed itemsets and their corresponding rules.* An important implication is that mining frequent closed itemsets has the same power as mining the complete set of frequent itemsets, but it will substantially reduce redundant rules to be generated and increase both efficiency and effectiveness of mining.

Let's examine a simple example. Suppose a database contains only two transactions, "$\{(a_1, a_2, \ldots, a_{100}), (a_1, a_2, \ldots, a_{50})\}$", the minimum support threshold is 1 (i.e., every occurrence is frequent), and the minimum confidence threshold is 50%. The traditional association mining method will generate $2^{100} - 1 \approx 10^{30}$ frequent itemsets, which are (a_1), …, (a_{100}), (a_1, a_2), …, (a_{99}, a_{100}), …, $(a_1, a_2, \ldots, a_{100})$, and a tremendous number of association rules, whereas a frequent closed itemset mining will generate only two frequent closed itemsets: $\{(a_1, a_2, \ldots, a_{50}), (a_1, a_2, \ldots, a_{100})\}$, and one association rule, "$(a_1, a_2, \ldots, a_{50}) \Rightarrow (a_{51}, a_{52}, \ldots, a_{100})$", since all the others can be derived from this one easily.

In this paper, we study efficient mining of frequent closed itemsets in large databases. Pasquier et al. [9] propose an Apriori-based mining algorithm, called A-close. Zaki and Hsiao [10] propose another mining algorithm, CHARM, which improves mining efficiency by exploring an item-based data structure. According to our analysis, A-close and CHARM are still costly when mining long patterns or with low minimum support thresholds in large databases. As a continued study on frequent pattern mining without candidate generation [6], we propose an efficient method for mining closed itemsets. Three techniques are developed for this purpose: (1) the framework of a recently developed efficient frequent pattern mining method, FP-growth [6], is extended, (2) strategies are devised to reduce the search space dramatically

* The work was supported in part by the Natural Sciences and Engineering Research Council of Canada (grant NSERC-A3723), the Networks of Centres of Excellence of Canada (grant NCE/IRIS-3), and the Hewlett-Packard Lab, U.S.A.

and identify the frequent closed itemsets quickly, and (3) a partition-based projection mechanism is established to make the mining efficient and scalable for large databases. Our performance study shows that CLOSET is efficient and scalable over large databases, and is faster than the previously proposed methods.

The remaining of the paper is organized as follows. In Section 2, the problem of mining frequent closed itemsets is defined and related concepts are introduced. In Section 3, we introduce our method, CLOSET, step-by-step. Section 4 reports the performance comparison of our method with A-close and CHARM as well as the scalability study. We summarize our work and discuss some future research directions in Section 5.

2 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of **items**. An **itemset** X is a non-empty subset of I. For brevity, itemset $X = \{i_{j_1}, i_{j_2}, \ldots, i_{j_m}\}$ can also be denoted as $X = i_{j_1} i_{j_2} \cdots i_{j_m}$. An itemset with m items is called an **m-itemset**. Duple $\langle tid, X\rangle$ is called a **transaction** if tid is a transaction identifier and X is an itemset. A **transaction database** TDB is a set of transactions.

An itemset X is **contained** in transaction $\langle tid, Y\rangle$ if $X \subseteq Y$. Given a transaction database TDB, the **support**[1] of an itemset X, denoted as $sup(X)$, is the number of transactions in TDB which contains X. An **association rule** $R : X \Rightarrow Y$ is an implication between two itemsets X and Y where $X, Y \subset I$ and $X \cap Y = \emptyset$. The **support of the rule**, denoted as $sup(X \Rightarrow Y)$, is defined as $sup(X \cup Y)$. The **confidence** of the rule, denoted as $conf(X \Rightarrow Y)$, is defined as $\frac{sup(X \cup Y)}{sup(X)}$.

As discussed by many studies, given a transaction database TDB, a minimal support threshold min_sup, and a minimal confidence threshold min_conf, the problem of **association rule mining** is to find the complete set of association rules in the database with support and confidence passing the thresholds, respectively. Also, as it has been shown in [2], the problem of mining association rules can be divided into two sub-problems:

1. Find all *frequent itemsets* in the transaction database with respect to the given support threshold. An itemset is called a **frequent itemset** if its support is no less than min_sup.
2. For each frequent itemset Y found, generate all association rules $X \Rightarrow Y - X$ where $X \subset Y$, if its confidence is no less than min_conf.

The requirement of mining the *complete set* of association rules leads to two problems: (1) there may exist a large number of frequent itemsets in a transaction database, especially when the support threshold is low, and (2) there may exist a huge number of association rules. It is hard for users to comprehend and manipulate a huge number of rules.

An interesting alternative to this problem is the mining of *frequent closed itemsets* and their corresponding association rules, proposed in [9].

Definition 1 (Frequent closed itemset) An itemset X is a **closed itemset** if there exists no itemset X' such that (1) X' is a proper superset of X, and (2) every transaction containing X also contains X'. A closed itemset X is **frequent** if its support passes the given support threshold. □

Thus, instead of mining association rules on all the itemsets, one can mine association rules on frequent closed itemsets only.

Definition 2 (Association rule on frequent closed itemsets) Rule $X \Rightarrow Y$ is an **association rule on frequent closed itemsets** if (1) both X and $X \cup Y$ are frequent closed itemsets, (2) there does not exist frequent closed itemset Z such that $X \subset Z \subset (X \cup Y)$, and (3) the confidence of the rule passes the given confident threshold[2]. □

Similar to mining association rules, the complete set of association rules on frequent closed itemsets can be mined in a two-step process: (1) mining the set of frequent closed itemsets with min_sup, and (2) generating the complete set of association rules on the frequent closed itemsets with min_conf.

Example 1 (Association rule mining) A transaction database TDB is given in Table 1. $\langle 40, \{a, c, d, f\}\rangle$ is a *transaction*, in which 40 is the *transaction identifier*, and $\{a, c, d, f\}$ is an *itemset*. Itemset $\{a, c, d, f\}$ can also be denoted as $acdf$.

Given $min\_sup = 2$ and $min\_conf = 50\%$, *association rules* can be mined in a two-step process:

- *Find frequent itemsets in the transaction database.* This can be done by Apriori as shown in most

[1] For convenience of discussion, *support* is defined here as *absolute* occurrence frequency. Notice it is defined in some literature as the relative one, i.e., the occurrence frequency vs. the total number of transactions in the transaction database.

[2] The second requirement is based on the rationale that if there are rules $X \Rightarrow Y$ and $(X \cup Y) \Rightarrow Z$, the rule $X \Rightarrow (Y \cup Z)$ is redundant, since $sup(X \Rightarrow (Y \cup Z))$ yields to $sup(X \cup Y) \Rightarrow Z$ and $conf(X \Rightarrow (Y \cup Z)) = conf(X \Rightarrow Y) \times conf((X \cup Y) \Rightarrow Z)$.

Transaction ID	Items in transaction
10	a, c, d, e, f
20	a, b, e
30	c, e, f
40	a, c, d, f
50	c, e, f

Table 1: The transaction database TDB.

association mining studies. There are in total 20 frequent itemsets in TDB, out of which only six are closed: $acdf$, cef, ae, cf, a and e. Neither ac nor d is closed since every transaction containing them also contains f.

- *Generate all association rules on each frequent itemset.* For example, for frequent itemset $acdf$, cf is a subset with support 4, so the *confidence* of rule $R : cf \Rightarrow ad$ is $\frac{sup(acdf)}{sup(cf)} = \frac{2}{4} \geq 50\%$. Thus, R is an *association rule*. It can be verified that for frequent itemset $acdf$, in total 14 association rules can be generated: every implication from $X \subset \{a, c, d, f\}$ $(X \neq \emptyset)$ to $\{a, c, d, f\} - X$ is an association rule.

Notice that all the association rules generated from $acdf$, except for $cf \Rightarrow ad$ and $a \Rightarrow cdf$, are with confidence 100%. For example, $c \Rightarrow adf$ with confidence 100% since items c and f always happen together. That is, only the association rules on frequent closed itemsets look interesting. Moreover, to compute confidence of such interesting association rules, only the supports of frequent closed itemsets are needed. For example, to derive $cf \Rightarrow ad$ and $a \Rightarrow cdf$, we only need to know the support of $acdf$, cf and a.

Let's derive all the association rules on frequent closed itemsets. Each rule is presented in the form of $X \Rightarrow Y$ $(support, confidence)$, where X and Y are itemsets. Frequent closed itemset $acdf$ has only two subsets, cf and a, as frequent closed itemsets. Thus, it generates two association rules: $cf \Rightarrow ad$ (2, 50%) and $a \Rightarrow cdf$ (2, 67%). The other four are $e \Rightarrow cf$ (3, 75%), $cf \Rightarrow e$ (3, 75%), $e \Rightarrow a$ (2, 50%) and $a \Rightarrow e$ (2, 67%). □

We refer readers to [9] for the theoretical foundation of association rules on frequent closed itemsets. In this paper, we focus on how to find the complete set of frequent closed itemsets efficiently from large database, which is called the **frequent closed itemset mining problem**.

Before introducing our method on mining frequent closed itemsets, we present one property of closed itemsets, which follows the definition of closed itemsets.

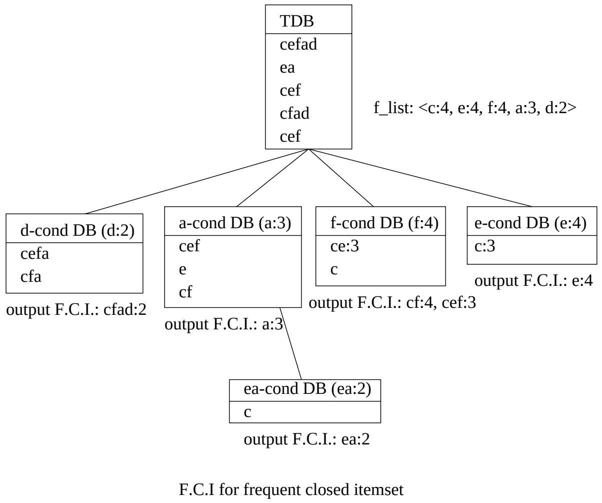

Figure 1: Mining *frequent closed itemsets* (abbreviated as F.C.I.) using CLOSET.

Lemma 2.1 *Let X and Y be two itemsets, and $sup(X) = sup(Y)$. Y is not a closed itemset if $Y \subset X$.* □

3 Efficiently Mining Frequent Closed Itemsets

In this section, we study efficient mining of frequent closed itemsets. In Section 3.1, we first illustrate the mining process of CLOSET with an example. Then, we presents the CLOSET algorithm in Section 3.2. Enhancement of the scalability of the method is discussed in Section 3.3.

3.1 Mining frequent closed itemsets with projected database: An example

Let's examine how to mine frequent closed itemsets using the following example.

Example 2 (CLOSET) For the same transaction database TDB in Table 1 with $min\_sup = 2$, we introduce a divide-and-conquer method for mining frequent closed itemset. The method explores the concepts of projected database [1, 6], as shown in Figure 1.

1. Find frequent items. Scan TDB to find the set of frequent items and derive a (global) *frequent item list*, called **f_list**, and f_list $= \langle c : 4, e : 4, f : 4, a : 3, d : 2\rangle$, where the items are sorted in support descending order, and the number after ":" indicates the support of the item. For easier understanding, the frequent items in each transaction are listed in Figure 1 according to the

order of f_list and any infrequent item, such as b, is omitted. For example, abe is listed as ea.

2. Divide search space. All the frequent closed itemsets can be divided into 5 non-overlap subsets based on the f_list: (1) the ones containing item d, (2) the ones containing item a but no d, (3) the ones containing item f but no a nor d, (4) the ones containing e but no f, a nor d, and (5) the one containing only c. Once all subsets are found, the complete set of frequent closed itemsets is done.

3. Find subsets of frequent closed itemsets. The subsets of frequent closed itemsets can be mined by constructing corresponding *conditional databases* and mine each recursively.

   (a) Find frequent closed itemsets containing d. Only transactions containing d are needed. The *d-conditional database*, denoted as $TDB|_d$, contains all the transactions having d, which is $\{cefa,\ cfa\}$. Notice that item d is omitted in each transaction since it appears in every transaction in the d-conditional database.

   The support of d is 2. Items c, f, and a appear twice respectively in $TDB|_d$. That is, every transaction containing d also contains c, f, and a. Moreover, e is infrequent since it appears only once in $TDB|_d$. Therefore, $cfad : 2$ is a frequent closed itemset. Since this itemset covers every frequent item in $TDB|_d$, the mining of $TDB|_d$ finishes.

   (b) Find frequent closed itemsets containing a but no d. Similarly, the *a-conditional database*, $TDB|_a = \{cef, e, cf\}$. Item d in such transactions are omitted, since all frequent closed itemsets containing d have been found in $TDB|_d$.

   Since $sup(a) = 3$ and there is no any item appearing in every transactions in the a-conditional database, $a : 3$ is a frequent closed itemset.

   To find the remaining frequent closed itemsets containing a but no d, we need to further project the a-conditional database. First, the set of frequent items in the a-conditional database forms a *local frequent item list*, f_list$_a$ $= \langle c : 2, e : 2, f : 2\rangle$[3]. Local infrequent item is ignored even if it is in global f_list.

   According to f_list$_a$, the frequent closed itemsets containing a but no d can be further partitioned into three subsets: (1) the ones containing af but no d, (2) the ones containing ae but not d or f, and (3) the ones containing ac but no d, e or f. They can be mined by constructing conditional databases recursively.

   The support of fa equals to that of $cfad$, which is a super set of fa and also a frequent closed itemset already found. That means every transaction containing fa must also contain $cfad$. Therefore, there is no frequent closed itemset containing fa but no d. Similarly, there is no frequent closed itemset containing ca but not d, e or f, since ca is a subset of $cfad$ and $sup(ca) = sup(cfad)$.

   The ea-conditional database, $TDB|_ea = \{c\}$, cannot generate any frequent items. Thus, $ea : 2$ should be a frequent closed itemset.

   (c) Find frequent closed itemsets containing f but no a nor d. The *f-conditional database*, $TDB|_f = \{ce : 3, c\}$, where $ce : 3$ indicates that ce happens three times. Since c happens in every transaction in the f-conditional database, and cf is not a subset of any frequent closed itemset with the same support, $cf : 4$ is a frequent closed itemset. Since the support of fc also equals to those of f and c, f and c always happen together, so there is no frequent closed itemsets containing c but no f. Also, that $cef : 3$ is not a subset of any itemset found, so it is a frequent closed itemset.

   (d) Find frequent closed itemsets containing e but no f, a nor d. Similarly, the *e-conditional database*, $TDB|_e = \{c : 3\}$. But ce is not a closed itemset since it is a proper subset of cef and $sup(ce) = sup(cef)$. However, $e : 4$ is a frequent closed itemsets.

   (e) Find frequent closed itemsets containing only c. In Step 3c, we know that there is no frequent closed itemsets containing c but no f, so there is no frequent closed itemsets containing only c.

4. In summary, the set of frequent closed itemsets found is $\{acdf : 2,\ a : 3,\ ae : 2,\ cf : 4,\ cef : 3,\ e : 4\}$. □

3.2 CLOSET: Algorithm and Soundness

Now, let us justify the correctness and completeness of the mining process in Example 2.

Definition 3 (Frequent item list, f_list) Given a transaction database TDB and a support threshold min_sup, the list of all frequent items in support descending order is called the **frequent item list**, or f_list in short. □

Lemma 3.1 *Given a transaction database TDB, a support threshold min_sup, and* f_list $= \langle i_1, i_2, \ldots, i_n\rangle$,

[3] In this example, it happens f_list$_a$ is a prefix of (global) f_list, with different counts. In general, the local frequent items can be re-arranged according to the local support counts.

the problem of mining the complete set of frequent closed itemsets can be divided into n sub-problems: The j^{th} problem $(1 \leq j \leq n)$ is to find the complete set of frequent closed itemsets containing i_{n+1-j} but no i_k (for $n+1-j < k \leq n$). □

The problem partitioning can be performed recursively. That is, each subset of frequent closed itemsets can be further divided when necessary. This forms a divide-and-conquer framework. To mine the subsets of frequent closed itemsets, we construct corresponding conditional databases.

Definition 4 (Conditional database) Given a transaction database TDB. Let i be a frequent item in TDB. The **i-conditional database**, denoted as $TDB|_i$, is the subset of transactions in TDB containing i, and all the occurrences of infrequent items, item i, and items following i in the f_list are omitted.

Let j be a frequent item in X-conditional database $TDB|_X$, where X is a frequent itemset. The **jX-conditional database**, denoted as $TDB|_{jX}$, is the subset of transactions in $TDB|_X$ containing j and all the occurrences of local infrequent items, item j, and items following j in local f_list$_X$ are omitted. □

To find the frequent closed itemsets containing i but no other items following i in f_list, we construct the i-conditional database. Then the subproblem can be divided further if necessary. For instance, in Example 2, we further construct the fa- and ea-conditional databases based on the a-conditional database.

How can we identify the frequent closed itemsets from conditional databases? The following lemma provides the theoretical foundation that CLOSET can find frequent closed itemsets correctly.

Lemma 3.2 *If X is a frequent closed itemset, then there is no item appearing in every transaction in the X-conditional database.*

Proof. If there exists an item i appearing in every transaction in the X-conditional database, we have $sup(iX) = sup(X)$. Following Lemma 2.1, X cannot be a closed itemset. Thus, we have the lemma. □

Lemma 3.3 *If an itemset Y is the maximal set of items appearing in every transaction in the X-conditional database, and $X \cup Y$ is not subsumed by some already found frequent closed itemset with identical support, then $X \cup Y$ is a frequent closed itemset.*

Proof. If an itemset Y is the maximal set of items appearing in every transaction in the X-conditional database, $X \cup Y$ is potentially a frequent closed itemset. The crucial point becomes whether later generated frequent closed itemset may subsume it. Suppose there exists a frequent closed itemset $X \cup Y \cup Z$ which subsumes $X \cup Y$, i.e., being frequent and having identical support k. Z will occur together with X at least k times and should be either in X's conditional database or earlier, based on the rules of construction conditional databases. Thus it cannot appear later. Thus, we have the lemma. □

The search for closed itemsets can be improved further by a few optimization techniques as shown below.

Optimization 1 : Compress transactional and conditional databases using an FP-tree structure.

An FP-tree [6] is a *prefix tree* structure, representing compressed but complete frequent itemset information for a database. Its construction is simple. The transactions with the same prefix share the portion of a path from the root. Similarly, conditional FP-trees can be constructed for conditional databases. We refer readers to [6] for details about the FP-tree and the related techniques. There are the following benefits for using FP-tree in the closed itemsets computation.

- FP-tree compresses databases for frequent itmset mining. Transactions sharing common prefix paths of a branch of the tree will not create any new nodes in an FP-tree. Moreover, the deeper the recursion in the construction of conditional databases, the better chance of sharing, and the more compact the conditional FP-tree.

- Conditional databases can be derived from FP-tree efficiently. This is shown in [6]. Since FP-tree may compress multiple transactions into one path, the projection of this path is equivalent to the scan of multiple transactions.

Optimization 2 : Extract items appearing in every transaction of conditional database.

If there exists a set of items Y appearing in every transaction of the X-conditional database, $X \cup Y$ forms a frequent closed itemset if it is not a proper subset of some frequent closed itemset with the same support. For instance, in Example 2, since c, f, and a appear in every transaction in the d-conditional database, $cfad$ should be a frequent closed itemset. Note since such items can be easily identified at the item counting phase, such an optimization takes effect even before constructing the FP-tree for the

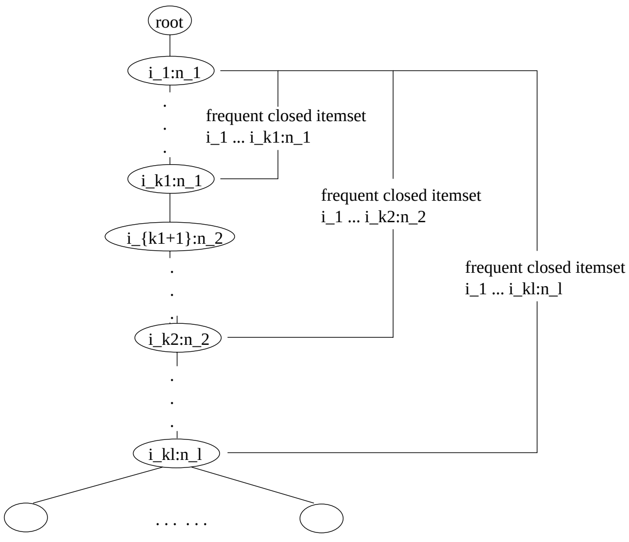

Figure 2: Directly extract frequent closed itemsets from FP-tree.

conditional database. The items extracted should be excluded from the local frequent item list and the conditional database. The soundness of the optimization follows Lemma 3.3.

Optimization 2 takes effect when forming the conditional database. It has the following benefits: (1) it reduces the size of FP-tree because the conditional database contains less number of items after such extraction, and (2) it may reduce the level of recursions since it combines a few items into one.

Optimization 3 : Directly extract frequent closed itemsets from FP-tree.

If there exists a single prefix path in an FP-tree, some frequent closed itemsets can be extracted directly from the conditional database. For example, the f-conditional database in Example 2 has transactions $ce : 3$ and $c : 1$. Its corresponding FP-tree has only one branch: $\langle c : 4, e : 3\rangle$. In this case, one can directly enumerate itemsets $cf : 4$ and $cef : 3$. Let us examine this in more detail.

Definition 5 Let i be a frequent item in the X-conditional database. If there is only one node N labeled i in the corresponding FP-tree, every ancestor of N has only one child, and N has (1) no child, (2) more than one child, or (3) one child with count value smaller than that of N, then the **i-single segment itemset** is the union of itemset X and the set of items including N and N's ancestors (excluding the root). □

Lemma 3.4 *The i-single segment itemset Y is a frequent closed itemset if the support of i within the conditional database passes the given threshold and Y is not a proper subset of any frequent closed itemset already found.*

Proof. In FP-tree, the count of N's every ancestor is no less than that of N. Since the support of i within the conditional database passes the support threshold, Y is a frequent itemset. Now we show Y is closed. Suppose there is an item j appearing in every transaction containing Y but $j \notin Y$. The support of j in the conditional database must be equal to that of i. Since $j \notin Y$, j must follow i in the local frequent item list and all item between i and j (including i and j) have the same support, i.e., they also appear in every transaction in the conditional database. According to the construction of FP-tree, i should have only one son node, which is labeled by the item following i in the local frequent item list, and the count of that node is exactly the same as that of i. That leads to a conflict with Y is the i-single segment itemset. Thus, we have the lemma. □

Optimization 3 shares similar benefits as Optimization 2. It allows the program to identify frequent closed itemsets quickly, reduces the size of the remaining FP-tree to be examined, and reduces the level of recursions since it combines multiple items into one.

Optimization 4 : Prune search branches.

Let X and Y be two frequent itemsets with the same support. If $X \subset Y$, and Y is a closed itemset, then there is no need to search the X-conditional database because there is no hope to generate frequent closed itemset from there. For example, in Example 2, we do not need to search the c-conditional database, since c is a subset of fc, which is a frequent closed itemset with the same support. The soundness of the optimization is verified in the following lemma.

Lemma 3.5 *Let X and Y be two frequent itemsets with the same support. If $X \subset Y$, and Y is closed, then there exists no frequent closed itemset containing X but not $Y - X$.*

Proof. Let Z be a frequent closed itemset containing X. Suppose Z does not contain some item $i \in Y - X$. Since $X \subset Y$, according to the *A-priori* heuristic, $sup(X) \geq (Y)$. $sup(X) = sup(Y)$ holds only if for every transaction containing X, it also contains $Y - X$. So item i must appear in every transaction containing Z, since $X \subset Z$. That means Z is not closed. So we have the lemma. □

Based on the above reasoning and analysis, we have the algorithm of CLOSET as follows.

Algorithm 1 (CLOSET): Mining frequent closed itemsets by the FP-tree method.

Input: Transaction database TDB and support threshold min_sup;

Output: The complete set of frequent closed itemsets;

Method:

1. Initialization. Let FCI be the set of frequent closed itemset. Initialize $FCI \longleftarrow \emptyset$;
2. Find frequent items. Scan transaction database TDB, compute frequent item list f_list;
3. Mine frequent closed itemsets recursively. Call CLOSET($\emptyset, TDB$, f_list, FCI).

Subroutine CLOSET(X, DB, f_list, FCI)

Parameters:

- X: the frequent itemset if DB is an X-conditional database, or $\emptyset$ if DB is TDB;
- DB: transaction database of conditional database;
- f_list: frequent item list of DB;
- FCI: The set of frequent closed itemsets already found.

Method:

1. Let Y be the set of items in f_list such that they appear in every transaction of DB, insert $X \cup Y$ to FCI if it is not a proper subset of some itemset in FCI with the same support; // Applying Optimization 2
2. Build FP-tree for DB, items already be extracted should be excluded; // Applying Optimization 1
3. Apply Optimization 3 to extract frequent closed itemsets if it is possible;
4. Form conditional database for every remaining item in f_list, at the same time, compute local frequent item lists for these conditional databases;
5. For each remaining item i in f_list, starting from the last one, call CLOSET($iX, DB|_i$, f_list$_i$, FCI) if iX is not a subset of any frequent closed itemset already found with the same support count, where $DB|_i$ is the i-conditional database with respect to DB and f_list$_i$ is the corresponding frequent item list. // Applying Optimization 4 □

Lemma 3.6 *An itemset is a frequent closed itemset iff* CLOSET *says so.*

Proof. An itemset X is identified as a frequent closed itemset by CLOSET when (1) X is frequent, (2) there is no item appearing in every transaction in X-conditional database, and (3) X is not a proper subset of any frequent closed itemset already found. To have the lemma, we show that there is no frequent closed itemset Y which can be found later such that $X \subset Y$. Suppose we can find such an itemset Y. Then $(Y - X) \neq \emptyset$ must happen in every transaction of the X-conditional database. That leads to a conflict with the fact that there is no item appearing in every transaction in the X-conditional database. Thus, we have the lemma. □

The correctness of the algorithm has been reasoned step-by-step in this section. It generates the complete set of frequent closed itemsets, as shown in Lemma 3.6. The four optimization techniques work with the divide-and-conquer method to ensure that the frequent closed itemsets can be extracted efficiently, and the search space can be reduced substantially. However, if the transaction database is very large, we cannot assume that the FP-tree can always be held in main memory. In next section, we develop some techniques to ensure the scalability of CLOSET in large databases.

3.3 Scaling up CLOSET in large databases

As specified in the last section, FP-tree contributes substantially to the efficiency of CLOSET. When the transaction database is large, it is unrealistic to construct a main memory-based FP-tree. In such cases, we can first construct conditional databases without FP-tree, or construct disk-based FP-trees. Disk-based FP-tree has been discussed in [6]. In this section, we focus on building conditional databases without FP-tree.

A naive method is to expand all conditional databases from one parent at a time. However, such a method basically duplicates TDB $\frac{l}{2}$ times, where l is the average number of frequent items in transactions. If the transaction database is very large, the transactions are long, and there are many frequent items, construction of many conditional databases could be a costly operation.

Here, we propose a *partition-based* approach, which can reduce the space cost dramatically. We illustrate the principle using the following example.

Example 3 Let us consider construction of conditional databases in Example 2 using a partition-based approach, as demonstrated in Figure 3.

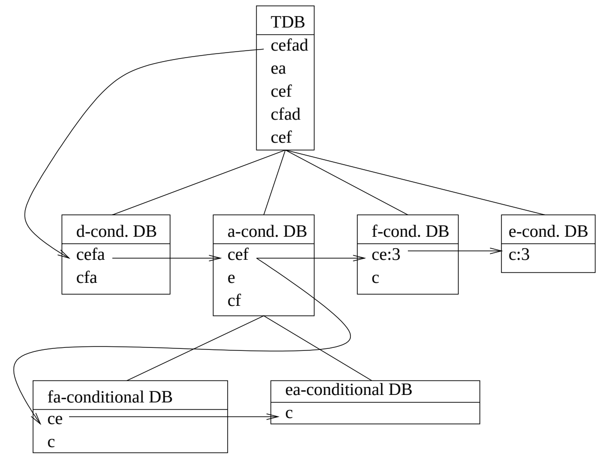

Figure 3: Constructing conditional databases in partition-based approach.

In the construction of conditional databases, instead of copying a transaction to every conditional database it takes part in, we only copy it to that of the last f_list item it contains. For example, d is the last item of the f_list the first transaction $cefad$ contains. So, instead of being copied into d- a- f- and e-conditional databases simultaneously, the tuple is only copied to the d-conditional database. After the d-conditional database is processed, the transaction is transferred to the conditional database of the second to the last item a, and so on. In such a way, we guarantee that at each level of recursion, the database is partitioned at most once. But once the partition is done, the original database can be gone. Such a partition-based conditional database construction needs to scan the database only once.

Please note that in the processing of a-conditional database, it takes one scan of the a-conditional database to partition it to fa- and ea-conditional databases. At the same time, transactions in the a-conditional database should be copied to the f- and e-conditional databases. Figure 3 shows that how $cefad$ is copied to various conditional databases in turn. □

With the partition-based conditional database construction, CLOSET can proceed without FP-tree at the first several rounds when the transaction database is large, and FP-trees are constructed only when the size of conditional databases can fit in memory.

One may wonder if we still can use Optimization 3 without FP-tree. Fortunately, we still can use it by maintaining one branch of FP-tree. The spirit is that we only maintain the upper portion of FP-tree from the root to the first node with more than one son branch.

4 Performance Study

In this section, we report our performance study of the three algorithms for mining frequent closed itemsets: CLOSET, CHARM, and A-close. A-close finds frequent closed itemsets by (1) using the Apriori framework, (2) pruning redundancies in candidates, and (3) post-processing to generate complete but non-duplicate result. CHARM explores a *vertical* data format, and find frequent closet itemsets by computing intersections of sets of transaction *id*s (*tid*s) for itemsets.

All the experiments are performed on a 233MHz Pentium PC with 128MB main memory, running on Microsoft Windows/NT. All the programs are written in Microsoft/Visual C++6.0. The A-close and CHARM are implemented as described in [9] and [10]. We use *runtime*, i.e., the period between input and output, to report our result, instead of using *CPU time* measured in some literature.

We test the three methods on various datasets, including synthetic ones generated by the standard procedure described in [2], and real datasets used in [4, 10]. Limited by space, we reported here only the results on three datasets as follows.

- Synthetic dataset T25I20D100K with 10K items. In this dataset, the average transaction size and average maximal potentially frequent itemset size are set to 25 and 20, respectively, while there are totally 100K transactions. This dataset is sparse. Most of frequent itemsets are closed.

- Real dataset I: *Connect-4*. This data set is from the UC-Irvine Machine Learning Database Repository[4]. It is compiled from the *Connect-4* game state information. The total number of transactions is 67, 557, while each transaction is with 43 items. It is a dense dataset with a lot of long frequent itemsets.

- Real dataset II: *pumsb*. This data set is from the IBM Almaden Research Center[5]. There are 49, 046 transactions in it, while each transaction has 74 items. It is a dense dataset with many long frequent itemsets.

4.1 Reduction of the size of itemsets using frequent closed itemsets

Our experiments show that the number of frequent itemsets which need to be represented in mining can be reduced by an order of magnitude in a dense database if they are represented by frequent closed itemsets. For example, Table 2 lists the numbers

[4] http://www.ics.uci.edu/~mlearn/MLRepository.html

[5] http://www.almaden.ibm.com/cs/quest/demos.html

of frequent closed itemsets (#F.C.I) and frequent itemsets (#F.I), as well as their ratio, in dataset *Connect-4*.

Support	#F.C.I	#F.I	$\frac{\#F.I}{\#F.C.I}$
64179 (95%)	812	2,205	2.72
60801 (90%)	3,486	27,127	7.78
54046 (80%)	15,107	533,975	35.35
47290 (70%)	35,875	4,129,839	115.12

Table 2: The number of frequent closed itemsets and frequent itemsets in dataset *Connect-4*.(F.C.I for *frequent closed itemsets* and F.I for *frequent itemsets.*)

If we want to mine association rules in a dense database, such as *Connect-4*, mining the set of frequent closed itemsets and then generating rules only on them will reduce search space substantially and generate much smaller set of rules. As the support threshold decreases, the saving becomes increasingly substantial.

4.2 Comparison of A-close, CHARM and CLOSET

The scalabilities of A-close, CHARM and CLOSET are tested using various datasets. CLOSET outperforms both CHARM and A-close.

As shown in Figure 4, in sparse dataset I20T25100K, a majority of frequent itemsets are closed itemsets. The performance of A-close is close to that of Apriori. The advantage of CLOSET over A-close is basically the same as that of FP-growth over Apriori. In this dataset, CHARM also wins Apriori. Since the support threshold is low, and the transaction identification (*tid*) sets for frequent itemsets are relatively small, CHARM is efficient. But it is slower than CLOSET.

The advantage of CLOSET becomes significant on dense datasets. The results on dataset *Connect-4* is shown in Figure 5. Please note that the runtime in this figure is in logarithmic scale. For example, CLOSET uses only 1690 seconds to find out the complete set of 130,101 frequent closed itemsets, when the support threshold is set to 33779 (50%). A-close even cannot find the result for support threshold 54046 (80%) within that time.

Pumsb is a challenging dataset. The results over this dataset are shown in Figure 6. A-close uses more than 250 seconds to find out the frequent closed itemsets for support threshold 90%, but CLOSET needs only less than 100 seconds to find out that for support threshold 80%.

From the experiments, we can observe that a non-trivial cost of CHARM is from many intersection operations over large sets of *tid*s. For example, in dataset *Connect-4*, if the support threshold is set to 95%, each set of *tid*s of frequent itemset contains at least $67557 \times 95\% = 64179$ *tid*s.

In order to test the scalability of CLOSET, we generate the synthetic datasets with size in 2 to 10 times, and replicate the transactions of real datasets 2 to 10 times. We keep the support threshold constant in percentage. The results are shown in Figure 7. The figure shows that CLOSET is scalable with the increase of the number of transactions. It is interesting to see that the runtime of CLOSET over real datasets increases much slower than the sizes of real datasets do. That is because CLOSET scans the transaction databses only twice. After that, the mining is confined to the FP-tree. No matter how many times the datasets are replicated, the FP-tree remains in the same shape with respect to the constant support threshold in percentage.

In summary, CLOSET is efficient and scalable in mining frequent closed itemsets in large databases. It is much faster than A-close, and also faster than CHARM.

5 Conclusions

Mining complete set of itemsets often suffers from generating a very large number of itemsets and association rules. Mining frequent closed itemsets provides an interesting alternative since it inherits the same analytical power as mining the whole set of frequent itemsets but generates a much smaller set of frequent itemsets and leads to less and more interesting association rules than the former.

In this paper, we proposed an FP-tree-based database projection method, CLOSET, for efficient mining of frequent closed itemsets in large databases. Our proposed algorithm, CLOSET, for mining closed itemsets adopts three techniques: (1) applying a compressed, frequent pattern tree FP-tree structure for mining closed itemsets without candidate generation, (2) developing a single prefix path compression technique to identify frequent closed itemsets quickly, and (3) exploring a partition-based projection mechanism for scalable mining in large databases.

Our performance study shows that CLOSET is efficient and scalable over large databases, and is faster than the previously proposed methods.

Acknowledgements

We would like to express our thanks to Nicolas Pasquier and Lotfi Lakhal for promptly sending us their recent papers on frequent closed itemsets. We also thank the anonymous reviewers for their comments.

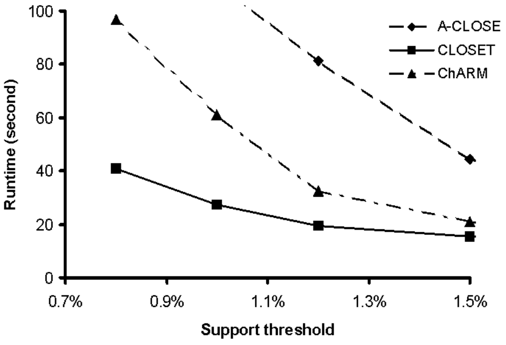

Figure 4: Scalability with support threshold on sparse dataset T25I20D100K.

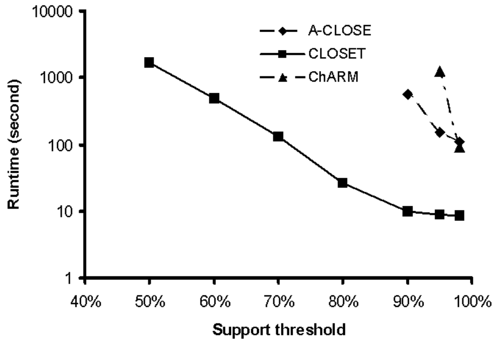

Figure 5: Scalability with support threshold on dense dataset *Connect-4*.

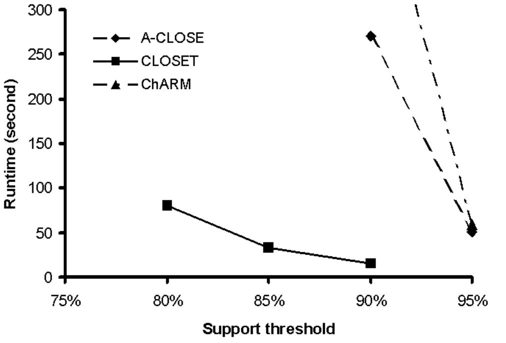

Figure 6: Scalability with support threshold on dense dataset *pumsb*.

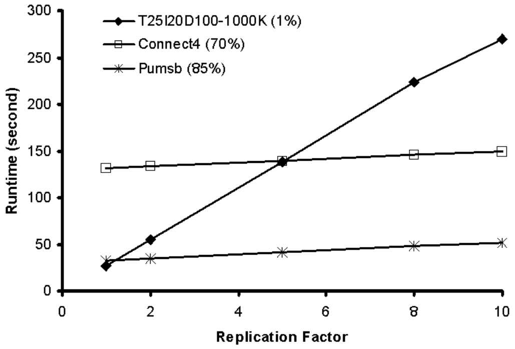

Figure 7: Size scaleup on datasets.

References

[1] R. Agarwal, C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. In *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, (to appear), 2000.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995.

[4] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 85–93, Seattle, Washington, June 1998.

[5] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, pages 106–115, Sydney, Australia, April 1999.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, Dallas, TX, May 2000.

[7] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 181–192, Seattle, WA, July 1994.

[8] H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[9] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, January 1999.

[10] M. J. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. In *Technical Report 99-10*, Computer Science, Rensselaer Polytechnic Institute, 1999.