Journal of Zhejiang University SCIENCE A
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

A novel algorithm for frequent itemset mining in data warehouses

XU Li-jun (徐利军)†, XIE Kang-lin (谢康林)

(*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200030, China*)

†E-mail: lijunxu@sjtu.edu.cn

Received June 11, 2005; revision accepted Oct. 22, 2005

Abstract: Current technology for frequent itemset mining mostly applies to the data stored in a single transaction database. This paper presents a novel algorithm MultiClose for frequent itemset mining in data warehouses. MultiClose respectively computes the results in single dimension tables and merges the results with a very efficient approach. Close itemsets technique is used to improve the performance of the algorithm. The authors propose an efficient implementation for star schemas in which their algorithm outperforms state-of-the-art single-table algorithms.

Key words: Frequent itemset, Close itemset, Star schema, Dimension table, Fact table
doi:10.1631/jzus.2006.A0216 **Document code:** A **CLC number:** TP31

INTRODUCTION

A data warehouse (Inmon, 1996) is an integrated and time-varying database primarily used for the support of decision-making, and integrates voluminous data from multiple and independent data sources consisting of operational databases in a common repository for querying and analysis. In terms of data modeling, a data warehouse consists of one or several dimensional models that are composed of a central fact table and a set of surrounding dimension tables each corresponding to one of the dimensions of the fact table. There are two kinds of dimensional models: star schemas and snowflake schemas. A star schema consists of a fact table and multiple small dimensions tables that are associated by foreign key to a central fact table. A star schema can be converted into a snowflake schema on condition that its dimension tables have subdimension tables.

While some information and facts can be collected from a data warehouse directly, much more remains hidden as implicit patterns and trends. Data mining in data warehouses often yields important insights and may lead to finding hidden knowledge. Data mining goes beyond the standard on-line analytical processing (OLAP), which mostly serves reporting purposes.

Discovering frequent itemsets is an important subject in many data mining applications, such as the discovery of association rules, correlations, sequential rules and episodes. In brief, discovering frequent itemsets is described as follows: Given a database including a large quantity of transactions, find all frequent itemsets, where a frequent itemset is one that occurs in at least a user-defined percentage of the database. But most associated algorithms (Agrawal *et al.*, 1993; Agrawal and Srikant, 1994; Han *et al.*, 2000; 2002; Pasquier *et al.*, 1999; Pei *et al.*, 2000; Shenoy *et al.*, 2000; Zaki and Hsiao, 2002) are typically designed for the data stored in a single transaction database and can not be directly used in data warehouses as the data is dispersed into multiple tables in data warehouses. Till now, there are few researches on this problem for frequent itemset mining in data warehouses (Cristofor and Simovici, 2001; Jensen and Soparkar, 2000; Ng *et al.*, 2002). It seems to be a feasible approach to use single-table algorithms in the joined table. However, besides the cost of the join

operation, the size of the joined table may be order of magnitude larger than the sum of sizes of all individual tables. What is more, the number of columns of the joined table is approximately equal to the sum of the ones of all individual tables. As frequent itemset mining is sensitive to the size of the data and the number of the columns, the computation time for the joined table is much longer than the sum of the computation time for the individual tables. So it is necessary that we separately consider the problem of frequent itemset mining in data warehouses.

In this paper, we present a novel algorithm MultiClose based on a star schema, which utilizes closed itemsets technique. A set of frequent closed itemsets is a concise representation of frequent itemsets, and is lossless in the sense that it can regenerate all of the frequent itemsets and determine their exact frequencies. In most cases the number of frequent itemsets can be dramatically reduced if it is represented by frequent closed itemsets. Experiments in the papers (Pasquier *et al.*, 1999; Pei *et al.*, 2000; Zaki and Hsiao, 2002) showed the algorithms generating frequent closed itemsets outperform the algorithms for discovering all frequent itemsets. Our algorithm is the first one to apply closed itemsets technique to frequent itemset mining in data warehouses.

PROBLEM DEFINITION

Let us first present some standard terminology for frequent itemset mining.

Definition 1 Let $I=\{i_1,i_2,\ldots,i_m\}$ be a set of distinct literals, called items. A subset of items is denoted as an itemset. An itemset with k items is called a k-itemset. A transaction $T=(tid, X)$ is a tuple where tid is a transaction-id and X is an itemset. A transaction database D is a set of transactions.

Definition 2 The support value of an itemset X, $sup(X)$, is the number of the transactions in D that contain itemset X, i.e., $sup(X)=|\{T|T\supseteq X\wedge T\in D\}|$. It shows statistical significance of an itemset.

Definition 3 An itemset is frequent if it satisfies the minimum support threshold.

Definition 4 Let $X\subseteq I$, we define the closure operator $h(X)$ as the intersection of transactions in D containing itemset X, i.e., $h(X)=\cap\{T|T\supseteq X\wedge T\in D\}$. The closure of an itemset is the greatest itemset that occurs in all transactions in D in which X occurs. An itemset X is closed if $h(X)=X$.

For simplicity, we assume that the data warehouse we investigate following a star schema, which consists of multiple small dimension tables and one big fact table. Each dimension table contains one primary key and some descriptive fields. In fact what we want is to discover potentially useful relationship among these descriptive fields other than primary keys. Note that the relationship may exist among the fields across distinct dimension tables. We assume that all descriptive fields have been processed in advance and transformed into categorical attributes. In order to simplify our discussion we assume the fact table only contains foreign keys from dimension tables. If the fact table contains some fields other than primary keys, we can place these fields into an extra dimension table and insert a new foreign key corresponding to the extra dimension table into the fact table.

RELATED WORK

Jenson and Soparkar (2000) presented an algorithm based on Apriori algorithm (Agrawal and Srikant, 1994) when the database is organized in a star schema. In the first stage it generates frequent itemset in each single table using a slightly modified version of Apriori, and in the second stage looks for frequent itemsets whose items belong to distinct tables via a multi-dimensional count array. It does not construct the whole joined table and process each row as the row is formed, thus storage cost for the joined table is avoided.

Ng *et al.*(2002) proposed an efficient algorithm for a star schema without actually performing the join operation. They designed a special tree structure, prefix tree, which is a compressed representation of the fact table, to speed up the calculation of support values. Each node of the prefix tree involves one value of a foreign key (the primary key of one dimension table) and one corresponding counter. The tree is constructed level by level. Nodes in the same level belong to the same dimension table. The algorithm uses vertical data format and borrows several ideas of VIPER (Shenoy *et al.*, 2000). Experiments showed that the approach of "mining before join" outperforms the approach of "join before mining"

even when the latter adopts known to be fastest single-table mining algorithm. Example 1 shows the main ideas of the algorithm.

Example 1 Suppose there is a database following star schema, and consisting of a fact table F (Table 1) and three dimension tables A, B, C (Table 2). The corresponding prefix tree is shown in Fig.1. Let the support threshold be set to 0.5. We will show how a frequent itemset $x_1x_3y_1y_3$ is determined. The algorithm first discovers frequent itemsets in Table 2 (for A, B and C respectively). If x_1x_3 and y_1y_3 are both frequent, the algorithm will merge them and see if $x_1x_3y_1y_3$ is frequent.

Table 1 Fact table *F*

tid_A	tid_B	tid_C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_1	c_3
a_2	b_1	c_2
a_1	b_3	c_3
a_1	b_4	c_2

Table 2 Dimension tables *A*, *B*, *C*

A		*B*		*C*	
tid	itemset	*tid*	itemset	*tid*	itemset
a_1	$x_1x_3x_5$	b_1	$y_1y_3y_5$	c_1	$z_1z_2z_3$
a_2	$x_2x_4x_6$	b_2	$y_2y_3y_5$	c_2	$z_3z_5z_6$
a_3	$x_1x_3x_4$	b_3	$y_1y_3y_4$	c_3	$z_2z_4z_6$
a_4	$x_2x_3x_6$	b_4	$y_2y_4y_5$		

The detailed steps are as follows:

$tid_T(X)$: a tidset for itemset X in a dimension table T, each element of which is a pair <*tid*, *count*>, where "*tid*" is the transaction ID relating to X in the table T and "*count*" is the number of occurrences of the tid in the fact table. Note that it is not necessary that X belongs to table T.

(1) $tid_A(x_1)=\{a_1(3),a_3(1)\}$, $tid_A(x_3)=\{a_1(3),a_3(1)\}$, $tid_A(x_1x_3)=tid_A(x_1)\cap tid_A(x_3)=\{a_1(3), a_3(1)\}$.

(2) $tid_B(y_1)=\{b_1(3),b_3(1)\}$, $tid_B(y_3)=\{b_1(3),b_2(1), b_3(1)\}$, $tid_B(y_1y_3)=tid_B(y_1)\cap tid_B(y_3)=\{b_1(3),b_3(1)\}$.

(3) The following is directly obtained from the prefix tree (Fig.1):

$$tid_B(x_1x_3)=\{b_1(1),b_3(1),b_4(1)\}\cup\{b_1(1)\}=\{b_1(2),b_3(1),b_4(1)\},$$

$$tid_B(x_1x_3y_1y_3)=tid_B(x_1x_3)\cap tid_B(y_1y_3)=\{b_1(2),b_3(1)\}.$$

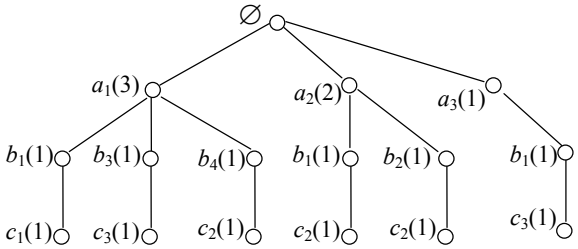

Fig.1 Prefix tree of sample database

MULTICLOSE ALGORITHM

In this section, MultiClose, an efficient algorithm for discovering frequent closed itemsets in a star schema is presented. We first give a general description of the algorithm and discuss how it is implemented in detail in later chapters. The algorithm in general terms is outlined as follows:

Step 1: Preprocessing.

(a) Scan the fact table, store the number of occurrences of the tids and construct the prefix tree;

(b) Scan the dimension tables, convert them to vertical data format with the counts obtained in (a).

Step 2: Local mining.

Discover frequent closed itemsets in each dimension table respectively.

Step 3: Global mining.

Discover frequent closed itemsets across distinct dimension tables via exploiting the results obtained in the local mining phase.

Preprocessing

The vertical data format we used in MultiClose is a variation of VTL (Shenoy *et al.*, 2000). Each dimension table is reorganized a set of columns with each column consisting of an item and an ordered list, each element of which is a pair <*tid*, *count*>, where "*tid*" is the transaction identifier including the item and "*count*" is the number of occurrences of the tid in the fact table.

Local mining

There are several single-table algorithms (Pasquier *et al.*, 1999; Pei *et al.*, 2000; Zaki and Hsiao, 2002) for discovering frequent closed itemsets. We select CHARM (Zaki and Hsiao, 2002) as the prototype of our local mining algorithm due to its distinct

advantages. CHARM adopts vertical data format and simultaneously explores both the itemset space and tidset (a set of tid) space with a novel search method that skips many levels to quickly discover the closed frequent itemsets.

When calculating support values in the dimension tables, we refer the frequency of any tid in the dimension table to the frequency of this tid in the fact table instead of the frequency in the dimension tables.

The main difference between our local mining algorithm and CHARM is that we use a two-level hash table result tree to perform post-pruning. Before an itemset is inserted into the set of frequent closed itemset in CHARM, it must pass subsumption checking to make sure that it is not a subset of one already found frequent closed itemset with identical support value. CHARM uses hash_multimap container in gcc STL for the hash table and the sum of the tids for the hash key. The two-level hash table result tree is derived from FP-tree (Han *et al.*, 2000; 2002). The design of this special data structure is based on the following idea: if an itemset X under consideration and one already found frequent closed itemset Y have the same support value and X is a subset of Y, X is subsumed by Y and is not a closed itemset. Thus the first level hash table uses the support value as the hash key and the second level is based on the last items of itemsets. In order to speed up the operation of subsumption checking, each node also records the support value of the itemset and the length of the path from the root to this node. In case one node belongs to multiple itemsets, the maximum one of the support values of these itemset sharing common prefix is stored. The itemset is inserted into the result tree according to ascending order of frequencies of items. The reason for the adoption of ascending order is explained in the description of the global mining phase.

By comparison, this structure outperforms the original method of CHARM. Moreover, it has a perfect property: it is convenient to regenerate the set of frequent closed itemsets. We just need to traverse the tree according to the algorithm of preorder tree traversal. If the parent node and the child node both have identical support value, it can be concluded that the items of the two nodes belong to a common frequent closed itemset. Otherwise, a new one is generated.

Example 2 Based on the sample database in Example 1, the two-level hash table result tree of tables A and B is shown in Fig.2. Let us show how to generate the set of frequent closed itemsets from the result tree of table B. According to the algorithm of preorder tree traversal, the result is $y_1, y_3, y_5, y_3, y_5, y_5$. First the support values of node y_1 and y_3 are both 4 and the support value of y_5 is 3, so y_1 and y_3 belong to the closed itemset y_1y_3 and y_5 belongs to another closed itemset $y_1y_3y_5$. Then as y_3 and y_5 have different support values, closed itemset y_3 and y_3y_5 are generated. The last frequent closed itemset y_5 is generated from node y_5. In summary, the set of frequent closed itemsets of table B is $\{y_1y_3(4), y_1y_3y_5(3), y_3(5), y_3y_5(4), y_5(5)\}$.

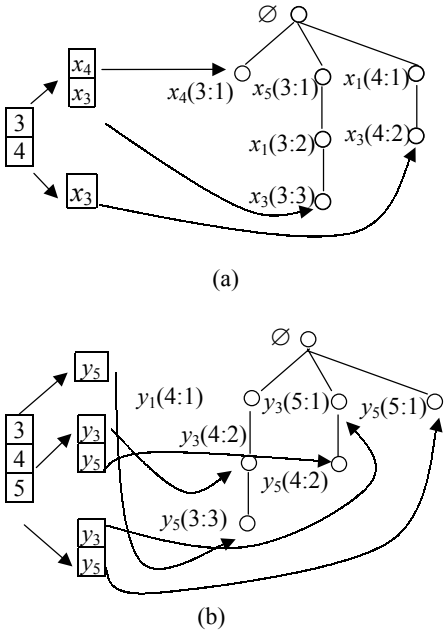

Fig.2 Two-level hash table result trees of tables *A* and *B*

Global mining

We first establish several results to prove the correctness of our algorithm.

Lemma 1 Let X, Y be two itemsets.

$$h(X\cup Y)=h(h(X)\cup h(Y)).$$

Proof

$$X\subseteq h(X),$$

$$Y\subseteq h(Y)\Rightarrow X\cup Y\subseteq h(X)\cup h(Y)\Rightarrow h(X\cup Y)\subseteq h(h(X)\cup h(Y)), \quad (1)$$

$$X\subseteq X\cup Y,\ Y\subseteq X\cup Y\Rightarrow h(X)\subseteq h(X\cup Y),$$

$$h(Y)\subseteq h(X\cup Y)\Rightarrow h(X)\cup h(Y)\subseteq h(X\cup Y)\Rightarrow$$

$$h(h(X)\cup h(Y))\subseteq h(h(X\cup Y))\Rightarrow h(h(X)\cup h(Y))\subseteq h(X\cup Y). \quad (2)$$

Eqs.(1) and (2) result in the following conclusion: $h(X\cup Y)=h(h(X)\cup h(Y))$.

Lemma 2 Let X, Y and Z be three itemsets. If $h(X)=h(Y)$, then $h(X\cup Z)=h(Y\cup Z)$.

Proof According to Lemma 1, $h(X\cup Z)=h(h(X)\cup h(Z))$ $=h(h(Y)\cup h(Z))$ $=h(Y\cup Z)$.

Theorem 1 Let A, B be two dimension tables and $X\cup Y$ be a closed itemset in the joined table of A and B. X and Y belong to two different dimension tables A, B respectively. Then X is a closed itemset in table A and Y is a closed itemset in table B.

Proof Let $h_A(X)$ denotes the closed itemset for X in table A and $h_{AB}(X)$ the closed itemset for X in the joined table of A and B.

$$X\subseteq h_A(X)\Rightarrow X\cup Y\subseteq h_A(X)\cup Y. \quad (3)$$

Then according to Lemma 2,

$$h_{AB}(h_A(X)\cup Y)=h_{AB}(X\cup Y)=X\cup Y$$
$$\Rightarrow h_A(X)\cup Y\subseteq X\cup Y. \quad (4)$$

From Eqs.(1) and (2),

$$X\cup Y=h_A(X)\cup Y\,. \quad (5)$$

Because X and Y do not overlap each other,

$$X\cap Y=\varnothing. \quad (6)$$

From Eqs.(5) and (6), we can conclude: $h_A(X)=X$.

So X is a closed itemset in table A. In the same way, Y is also a closed itemset in table B.

For simplicity, let FC_A denotes the set of frequent closed itemsets in table A, FC_{AB} the set of frequent closed itemsets in the joined table of A and B. Note that a closed itemset in a single dimension table may be subsumed by a closed itemset across tables.

Example 3 The frequent closed itemsets with a minimum support of 50% in A, B (Table 2) and the joined table of A, B is as follows:

FC_A: $\{x_4, x_1x_3, x_1x_3x_5\}$;

FC_B: $\{y_3, y_5, y_1y_3, y_3y_5, y_1y_3y_5\}$;

FC_{AB}: $\{y_3, y_5, x_1x_3, y_1y_3, y_3y_5, x_1x_3x_5, x_1x_3y_5,$ $x_4y_3y_5, y_1y_3y_5, x_1x_3y_1y_3\}$.

There are three frequent closed itemsets across dimension tables A and B, i.e., $\{x_5y_1y_3,\ x_3x_5y_4,$ $x_1x_3y_1y_3\}$. The projected subsets of these itemsets in tables A and B are $\{x_5, x_1x_3, x_3x_5\}$ and $\{y_4, y_1y_3\}$. Obviously, they belong to FC_A and FC_B respectively.

We first consider how to discover frequent closed itemsets across three dimension tables A, B and C like the sample database in Example 1. According to Theorem 1, we can determine all frequent closed itemsets across dimension tables via testing all possible combinations of frequent closed itemsets of distinct dimension tables. The number of possible combinations for finding FC_{AB} is equal to $|FC_A|\times|FC_B|$. Obviously, it is merely feasible when two tables generate a small number of frequent closed itemsets. Otherwise, it may generate a huge set of candidate itemsets. For example, there are merely three itemsets across two dimension tables A, B in the sample database. However, there are 15 possible cases according to Theorem 1. We propose an approach that can efficiently reduce possible cases. After local mining, frequent closed itemsets are stored in the two-level hash table result trees and can be generated from them. So we can discover frequent closed itemset across two tables by traversing the result trees. This approach traverses the result trees to generate frequent closed itemsets according to the algorithm of preorder tree traversal. Let X be the itemset of A under consideration, Y be the one of B under consideration. We first traverse the result tree of A, and test the combinations of X and itemsets from the result tree of B. There are two heuristics to be used in the procedure:

(1) If $X\cup Y$ is infrequent, we do not need to traverse the subtree of the node corresponding to Y.

(2) We assume Z is generated from the subtree of the node corresponding to X. For Z we only need to traverse the subtree of B that involves nodes generating itemsets whose combinations with X are frequent instead of the whole tree.

In nature, the two heuristics use the downward closure property of itemsets to prune the search space −the property that all supersets of an infrequent itemset must themselves be infrequent.

Unlike FP-tree, the result tree is organized in ascending order of items instead of descending order. In this way, a closed itemset with high probability may be generated by two subsets with smaller support values. So we increase opportunities for pruning possible candidates.

The main advantage of vertical mining lies in that support values can be determined through simple tidsets intersection instead of by using complex data structure like horizontal mining algorithms. But the vertical algorithm always suffers from a common disadvantage in that the intersection time increases rapidly when the cardinality of tidsets gets very large. Furthermore, the size of the intermediate result may

become too large to fit into memory. We adopt a novel technique, diffsets (Zaki and Gouda, 2003), which only track the difference between the tidsets of two candidate itemsets. Experiments (Zaki and Gouda, 2003; Zaki and Hsiao, 2002) showed that diffsets dramatically cut down the size of tidsets and considerably increase the speed of intersection operations.

We will briefly explain the idea of diffsets. Let $tid(X)$ denote the tidset of itemset X, and $diff(X)$ the difference of X with respect to a common prefix. For example, there are two itemsets PX and PY that have a common prefix P. Obviously, $tid(PXY)=tid(PX)\cap tid(PY)$. We assume that $diff(PX)$ and $diff(PY)$ are available, and that $diff(PX)=tid(P)-tid(X)$ and $diff(PY)=tid(P)-tid(Y)$. We can conclude that $sup(PXY)=sup(PX)-|diff(PXY)|$ according to the definition of diffsets. As $sup(PX)$ is already figured out, we only need to determine the size of $diff(PXY)$. Since $diff(PXY)=tid(PX)-tid(PY)=(tid(P)-tid(PY))-(tid(P)-tid(PX))=diff(PY)-diff(PX)$, it means that we can calculate $sup(PXY)$ if $diff(PX)$ and $diff(PY)$ are available.

In the implementation we use the approach in (Ng *et al.*, 2002), which is also discussed in Section 3, to compute support values without forming any row of the joined table.

Example 4 In this example we will show how MultiClose makes good use of the result tree and diffsets. The steps combining x_4 with the result tree of table B in Example 2 is as follows:

(1) $diff_B(y_1y_3x_4)=tid_B(y_1y_3)-tid_B(x_4)=\{b_1(3),b_3(1)\}-\{b_1(2),b_2(1)\}=\{b_1(1),b_3(1)\}$.

As $sup(y_1y_3)$ can be obtained from the result tree, $sup(x_4y_1y_3)=sup(y_1y_3)-|diff_B(y_1y_3x_4)|=4-2=2<3$.

(2) As we conclude $x_4y_1y_3y_5$ is infrequent from (1), the calculation of its support value is unnecessary.

(3) $diff_B(y_3x_4)=tid_B(y_3)-tid_B(x_4)=\{b_1(3),b_2(1),b_3(1)\}-\{b_1(2),b_2(1)\}=\{b_1(1),b_3(1)\}$, $sup(x_4y_3)=sup(y_3)-|diff_B(y_3x_4)|=5-2=3$.

(4) $diff_B(y_3y_5)=tid_B(y_3)-tid_B(y_5)=\{b_1(3),b_2(1),b_3(1)\}-\{b_1(3),b_2(1),b_4(1)\}=\{b_3(1)\}$, $diff_B(y_3x_4y_5)=diff_B(y_3y_5)-diff_B(y_3x_4)=\{b_3(1)\}-\{b_1(1),b_3(1)\}=\varnothing$, $sup(x_4y_3y_5)=sup(y_3x_4)-|diff_B(y_3x_4x_5)|=3-0=3$.

As $sup(x_4y_3)$ is equal to $sup(x_4y_3y_5)$, x_4y_3 is not a closed itemset.

(5) $diff_B(y_5x_4)=tid_B(y_5)-tid_B(x_4)=\{b_1(3),b_2(1),b_4(1)\}-\{b_1(2),b_2(1)\}=\{b_1(1),b_4(1)\}$, $sup(x_4y_5)=sup(y_5)-|diff_B(y_5x_4)|=5-2=3$.

As $sup(x_4y_5)$ is equal to $sup(x_4y_3y_5)$, x_4y_5 is not a closed itemset.

Note that a closed itemset in a single dimension table may be subsumed by a closed itemset across tables, so we still need to check closed itemsets in single tables. The two-level hash table result tree is still used to eliminate non-closed itemsets. The final result tree consists of all closed itemsets from the joined table of A and B.

Then we will discuss how to discover FC_{ABC}. As FC_{AB} is stored in the result tree, we may assume the database has a "virtual" dimension table AB. In order to facilitate the computation of support values, we first need to modify the prefix tree. We assign a new tid for one combination of a tid from table A and a tid from table B, which exists in the fact table, and reorganize the tree according to the new tid. Then we can perform similar to those steps above to discover FC_{ABC}. It is obvious that we can easily extend the above case to a star schema consisting of N dimension tables ($N>3$).

Example 5 We suppose tables A and B in Example 1 is replaced by a "virtual" dimension table AB (Table 3). The height of the prefix tree is decreased due to the introduction of new tids (Fig.3). For more details about the transformation, please refer to (Ng *et al.*, 2002).

Table 3 Dimension table *AB*

tid_A	tid_B	tid_{AB}
a_1	b_1	t_1
a_2	b_2	t_2
a_3	b_1	t_3
a_2	b_1	t_4
a_1	b_3	t_5
a_1	b_4	t_6

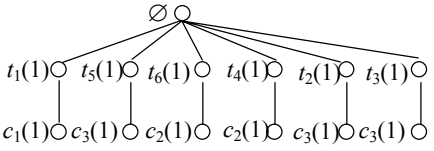

Fig.3 New prefix tree

Last we consider how to extend our algorithm to deal with a data warehouse following a snowflake schema. For simplicity, we assume a dimension table f_1 is connected to a set of subdimension tables in the

snowflake schema. We can divide the mining process into two steps. The first step is to compute results among subdimension tables of f_1. Then we can regard the snowflake schema as a star schema as the frequent closed itemsets of table f_1 are known. The second step is to discover frequent closed itemsets across all dimension tables like a star schema.

EXPERIMENTS

We restrict our experiments to a database consisting of three dimension tables and one fact table. We use the data generator used by Agrawal and Srikant (1994) to produce the dimension tables and assign a tid to each transaction in the dimension tables. The data generator has these parameters: D: the number of transactions; N: the number of items; T: the average length of transactions; I: the average length of the maximal potentially frequent itemsets; L: the number of maximal potentially large itemsets.

We use our data generator to generate the fact table. Each transaction in the fact table is generated through picking tids from three dimension tables. We assign a weight to each tid in the dimension tables, which corresponds to the probability that the tid will be selected. The weight is picked from an exponential distribution with unit means, which is so normalized that the sum of the weights for all tids in each dimension table is equal to one. This generator has only one parameter F which is the number of transactions in the fact table.

We evaluate MultiClose in comparison with CHARM. The experiments are performed on a Celeron 466 MHz with 512 MB, running Windows 2000. MultiClose is coded with Visual C++. The source code of CHARM is provided by its author. As CHARM is a Linux application, we have to run it on Cygwin environment on the same machine. The time of the join operation is added into the total time of CHARM.

We generate two types of datasets, SPARSE and DENSE. Each transaction typically only consists of a small fraction of all items in SPARSE. In contrast, DENSE contains many frequently occurring items. The default values of parameters of the datasets are shown in Table 4. Note that we assume that all dimension tables hold identical parameters.

Table 4 Parameters of datasets SPARSE and DENSE

Dataset	Dimension table					Tact table
	D	N	T	I	L	F
SPARSE	5000	500	10	5	100	100000
DENSE	5000	50	20	15	100	100000

It is known that association rule mining is sensitive to the number of items and the size of the dataset. So we design two kinds of experiments. In the first case, we perform experiments by varying the number of items, the average length of transactions and the number of maximal potential itemsets in each dimension table with a fixed support threshold value (Fig.4a and Fig.5a). We vary the average length of transactions and the number of maximal potential itemsets in order to make sure that the dataset contains enough frequent itemsets. In this case, the running time of CHARM is much longer than the running time of MultiClose. In nature CHARM performs a search over an itemset-tidset tree whose size is decided by the number of items. CHARM may traverse 2^N nodes at most provided that the database includes N items. Thus the performance of CHARM rapidly degrades with increasing number of items. As each dimension table merely includes one third of the items in our implementation, the time of the local mining phase is sure to be much less than the time of CHARM on the complete dataset.

In the second case, we perform experiments in which we vary the number of transactions in the fact table (Fig.4b and Fig.5b). The running time of two algorithms increases linearly with increasing number of the transactions. However, MultiClose still significantly outperforms CHARM in that one transaction in a dimension table may map many transactions in the fact table and the cost of comparison operations is saved.

Our experiments showed the value of Multi Close. We believe the advantage of our algorithm will be still significant even though there are more dimension tables in the star schema.

CONCLUSION

In this paper we propose a novel algorithm, MultiClose, which discover frequent closed itemsets in data warehouses without materializing join tables.

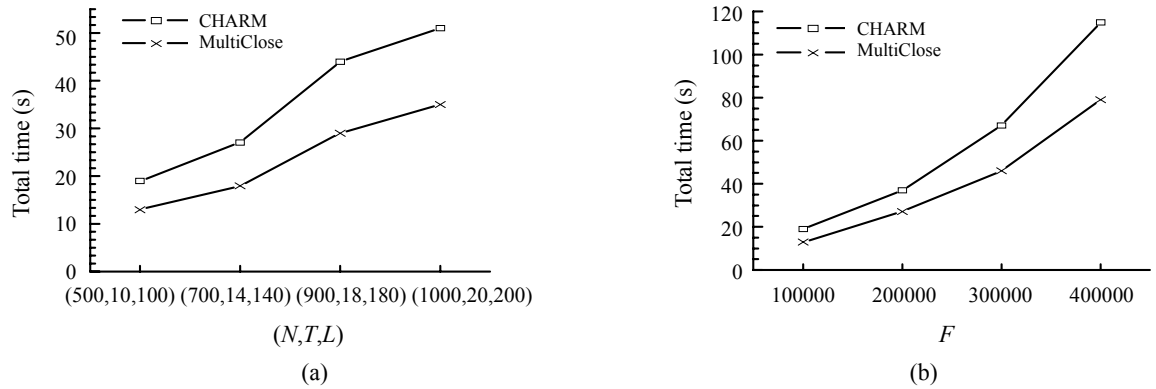

Fig.4 Performance of CHARM and MultiClose on SPARSE (*MinSup*=0.03)

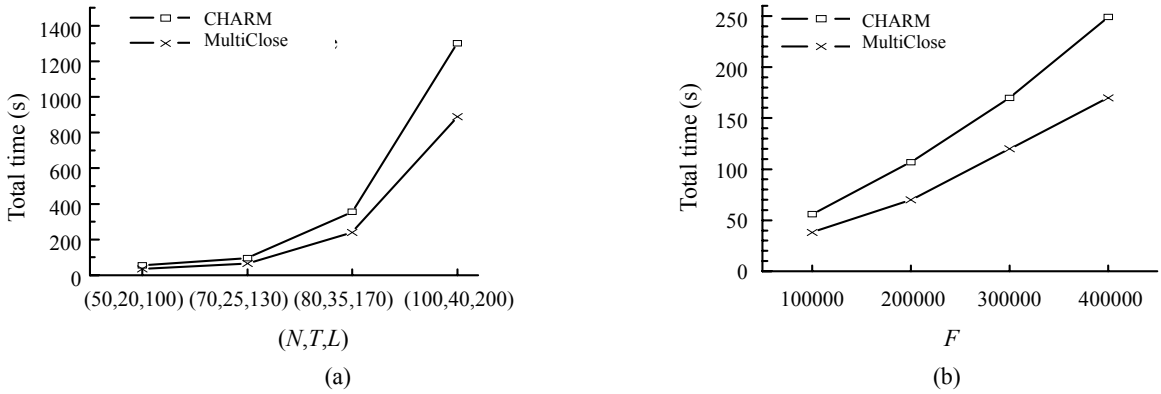

Fig.5 Performance of CHARM and MultiClose on DENSE (*MinSup*=0.3)

The algorithm is the first one in the literature to apply closed itemsets technique to multiple tables mining. We present an efficient implementation based on a star schema in which several smaller dimension tables are associated by foreign key to a central fact table. Furthermore, we discuss how to extend the case to snowflake schemas. We show MultiClose significantly outperforms the state-of-the-art single-table algorithms. Our research also implies the necessity of mining of decentralized datasets.

ACKNOWLEDGMENT

We would like to thank Mohammed J. Zaki for providing us the source code for CHARM.

References

Agrawal, R., Srikant, R., 1994. Fast Algorithms for Mining Association Rules. Proceedings of the International Conference on Very Large Databases, p.487-499.

Agrawal, R., Imilienski, T., Swami, A., 1993. Mining Association Rules Between Sets of Items in Large Databases. Proceedings of the ACM SIGMOD International Conference on the Management of Data, p.207-216.

Cristofor, L., Simovici, D., 2001. Mining Association Rules in Entity-relationship Modeled Databases. Technical Report TR-01-01, Computer Science Department, University of Massachusetts.

Han, J., Pei, J., Yin, Y., 2000. Mining Frequent Patterns without Candidate Generation. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, p.1-12.

Han, J., Wang, J., Lu, Y., Tzvetkov, P., 2002. Mining Top-*K* Frequent Closed Patterns without Minimum Support. Proceedings of the IEEE International Conference on Data Mining, p.211-218.

Inmon, W.H., 1996. Building the Data Warehouse, 2nd Edition. Wiley, Chichester.

Jensen, V.C., Soparkar, N., 2000. Frequent Itemset Counting across Multiple Tables. Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, p.49-61.

Ng, K., Fu, W., Wang, K., 2002. Mining Association Rules from Stars. Proceedings of the IEEE International Con-

ference on Data Mining, p.322-329.

Pasquier, N., Bastide, Y., Taouil, R., Lakha, L., 1999. Discovering Frequent Closed Itemsets for Association Rules. Proceedings of the 7th International Conference on Database Theory, p.398-416.

Pei, J., Han, J., Mao, R., 2000. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, p.21-30

Shenoy, P., Haritsa, J.R., Sundarshan, S., Bhalotia, G., Bawa, M., Shah, D., 2000. Turbo-charging Vertical Mining of Large Databases. Proceedings of ACM SIGMOD International Conference on Management of Data, p.22-33.

Zaki, M.J., Hsiao, C.J., 2002. CHARM: An Efficient Algorithm for Closed Itemset Mining. Proceedings of SIAMOD International Conference on Data Mining, p.457-473.

Zaki, M.J., Gouda, K., 2003. Fast Vertical Mining Using Diffsets. Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p.326-335.