http://www.zju.edu.cn/jzus; E-mail: jzus@zju.edu.cn
ISSN 1009-3095 *Journal of Zhejiang University SCIENCE* V.5, No.1, P.8-15, Jan., 2004

An efficient algorithm for mining closed itemsets*

LIU Jun-qiang (刘君强)†[1,2], PAN Yun-he (潘云鹤)[1]

([1] *Institute of Artificial Intelligence, Zhejiang University, Hangzhou 310027, China*)

([2] *Hangzhou University of Commerce, Hangzhou 310035, China*)

†E-mail: liujunq@mail.hz.zj.cn

Received Oct. 25, 2002; revision accepted Apr.18, 2003

Abstract: This paper presents a new efficient algorithm for mining frequent closed itemsets. It enumerates the closed set of frequent itemsets by using a novel compound frequent itemset tree that facilitates fast growth and efficient pruning of search space. It also employs a hybrid approach that adapts search strategies, representations of projected transaction subsets, and projecting methods to the characteristics of the dataset. Efficient local pruning, global subsumption checking, and fast hashing methods are detailed in this paper. The principle that balances the overheads of search space growth and pruning is also discussed. Extensive experimental evaluations on real world and artificial datasets showed that our algorithm outperforms CHARM by a factor of five and is one to three orders of magnitude more efficient than CLOSET and MAFIA.

Key words: Knowledge discovery, Data mining, Frequent closed patterns, Association rules
Document code: A **CLC number:** TP311; TP391

INTRODUCTION

Mining frequent itemsets is a fundamental and essential problem in many data mining applications including the discovery of association rules, strong rules, correlations, sequential rules, episodes, multi-dimensional patterns, and many other important discovery tasks (Agarwal and Srikant, 1994; Wang *et al.*, 2002). Most algorithms proposed so far work well on datasets where the sizes of itemsets are relatively small. However, they usually crash with dense datasets where the itemset sizes are large. Such datasets include those composed of questionnaire results, regular customer sales transactions, telecommunication data, and biological data from the fields of DNA and protein analysis(Agarwal *et al.*, 2000).

To address this problem, this paper proposes a new efficient algorithm to mine only the frequent closed itemsets. A novel compound frequent itemset tree is developed to organize the solution space, which facilitates the efficient local pruning and fast global subsumption checking of solution space, and which also requires much less memory. Our algorithm searches the solution space by integrating depth first and breadth first search strategies, opportunistically choosing between two different structures, array-based or tree-based, to represent projected transaction subsets, and heuristically deciding to build unfiltered pseudo projections or to make filtered ones according to the characteristics of the subsets. Comparative experiments showed that our algorithm is much more efficient than CHARM, CLOSET and MAFIA and is also the most scalable.

Related works

Pasquier *et al.* (1998) proposed to mine only closed set of frequent itemsets instead of complete set. The former is lossless in the sense that it uniquely determines the set of all frequent itemsets and their exact frequency. Meanwhile the closed set can be orders of magnitude smaller than the complete set of frequent itemsets, especially on dense datasets. Pasquier *et al.* (1999) developed an apriori-based algorithm A-Close that employs breadth first search to find frequent closed itemset. A-Close constructs a so-called generator set to integrate the pruning step to limit the search space. But A-Close still suffers from redundant scans of datasets and high costs of

* Project supported by the Ministry of Education (No. 111101-G10110) and the Zhejiang Province Natural Science Foundation (No. 602140), China

pattern matching inherent to breadth first search. Thereafter, many researcher presented quite a few algorithms that employ depth first search.

Pei *et al*. (2000) proposed the algorithm CLOSET based on FP-Growth, which represents projected transaction subsets by FP-trees and employs depth first search. CLOSET suffers from inefficiency caused by recursive building of "conditional FP-trees" that consume lots of CPU-time and memory. MAFIA is another depth first algorithm proposed by Burdick *et al*.(2001) and employs vertical bitmap representing projected transaction subsets, which consumes much more memory than array based and tree based structures when the support threshold is below 1/32. The operations of item counting, transaction subset projecting, and search space pruning are inefficient. CHARM presented by Zaki and Hsiao (2002) is the most efficient algorithm among the published. It enumerates closed itemsets by using a dual itemset-tidset search tree. The difficulty with CHARM is that memory expenditure of projected transaction subset of either the tidset form or the diffset form is huge, and the projecting operation is inefficient, especially for long itemsets or with low minimum support thresholds in large datasets.

PROBLEM DESCRIPTION

Given $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items, let database $\boldsymbol{T} = \{(tid_1, t_1), (tid_2, t_2), \ldots, (tid_n, t_n)\}$ be a set of transactions, where tid_k is the transaction identifier and t_k is a set of items, i.e., $t_k \subseteq \boldsymbol{I}$. We define the necessary concepts for describing our algorithm as follows.

Definition 1 Full itemset mapping, f, from powerset of $\boldsymbol{T}$ to powerset of $\boldsymbol{I}$, $f(T) = \{i \in \boldsymbol{I} \mid \forall (tid, t) \in T \subseteq \boldsymbol{T}, i \in t\}$, where $f(T)$ is the full itemset contained in $T \subseteq \mathbf{T}$.

Example 1 Let $\boldsymbol{I} = \{a, b, c, d, e, f, g, h, i, k, l, m, n, o, p, s\}$, $\boldsymbol{T} = \{(01, \{a, c, d, f, g, i, m, p\}), (02, \{a, b, c, f, l, m, o\}), (03, \{b, c, h, m, o\}), (04, \{b, f, k, p, s\}), (05, \{a, c, e, f, l, m, n, p\})\}$, then the full itemset contained in transaction 01, 02 and 03 is $\{c, m\}$.

Definition 2 The projection mapping, g, from powerset of $\boldsymbol{I}$ to powerset of $\boldsymbol{T}$, $g(I) = \{(tid, t) \in \boldsymbol{T} \mid \forall i \in I \subset \boldsymbol{I}, i \in t\}$, where $g(I)$ is transaction subsets that support $I \subseteq \boldsymbol{I}$.

If $I \subseteq t_k$, then I is said to be supported by transaction (tid_k, t_k). In Example 1, itemset $\{c, m\}$ is supported by transaction 01, 02, 03 and 05.

Property 1 (1) $T_1 \subseteq T_2 \Rightarrow f(T_1) \supseteq f(T_2)$;
(2) $T \subseteq g(I) \Rightarrow f(T) \supseteq I$;
(3) $g(I_1 \cup I_2) = g(I_1) \cap g(I_2)$;
(4) $I_1 \subseteq I_2 \Rightarrow g(I_1) \supseteq g(I_2)$.

Definition 3 Closure operator, h, on powerset of $\boldsymbol{I}$, $h(I) = f(g(I))$. An itemset $C \subseteq \boldsymbol{I}$ is closed, *iff* $h(C) = C$.

Property 2 (1) $I \subseteq h(I)$; (2) $h(h(I)) = h(I)$; (3) $I_1 \subseteq I_2 \Rightarrow h(I_1) \subseteq h(I_2)$.

Definition 4 The absolute support of $I \subseteq \boldsymbol{I}$, $support(I)$, is $\| g(I) \|$, and relative support is $\| g(I) \| \div \| \boldsymbol{T} \|$.

The absolute support of I is the number of transactions that support I, and the relative support is the percentage of transactions in $\boldsymbol{T}$ that support I.

Corollary 1 $support(I) = support(h(I))$.

Definition 5 Itemset C is frequent, if $support(C) \geqslant minsup$, a user given threshold. The set of frequent closed itemsets $FC = \{C \subseteq \boldsymbol{I} \mid C = h(C) \wedge support(C) \geqslant minsup\}$.

In Example 1, $h(\{c, m\}) = \{c, m\}$, hence $\{c, m\}$ is closed. $\{c, m\}$ has a support of 4 (80%). Given *minsup* of 3, there are 18 frequent itemsets and 5 frequent closed itemsets.

Property 3 Any subset of a frequent itemset is frequent (*Apriori*). Any superset of an infrequent itemset is infrequent (*Negative Apriori*).

Proof Given $J \subseteq I \subseteq \boldsymbol{I}$, according to Property 1 (4), we have $g(J) \supseteq g(I)$, i.e., $support(J) \geqslant support(I)$. Therefore, if I is frequent, i.e., $support(I) \geqslant minsup$, then J is frequent; if J is infrequent, i.e., $support(J) \leqslant minsup$, then I is infrequent.

COMPOUND FREQUENT ITEMSET TREE

We propose a novel structure, compound frequent itemset tree, abbreviated as CFIST, which can be used to enumerate either the complete set or the closed set of frequent itemsets.

Compound frequent itemset tree

Definition 6 Given the ordering of items $\prec$, minimum support *minsup*, the compound fre-

quent itemset tree, abbreviated as CFIST, is defined as $CFIST = (V, E)$, where V is the set of nodes and E is the set of edges. $\forall v \in V, v = (i, w, A, T, I)$, labeling item $i \in \boldsymbol{I}$ is denoted as $v.item$, weight w as $v.weight$, auxiliary set of items $A \subset \boldsymbol{I}$ as $v.AIS$, projected transaction subset $T \subseteq \boldsymbol{T}$ as $v.PTS$, and set of local frequent items $I \subseteq \boldsymbol{I}$ as $v.IL$. An edge from node $p = (i_p, w_p, A_p, T_p, I_p)$ to node $c = (i_c, w_c, A_c, T_c, I_c)$ is defined as $(p, c) \in E$, where p is the parent of c, and $i_c \in I_p$, $w_c = \| T_c \| \geqslant minsup$, $A_c \subset I_p$, $T_c = g(\{i_c\}) \cap T_p = g(\{i_c\}) \cap (\bigcup A_c) \cap T_p$, $I_c = \{i \in I_p \mid i_c \prec i \wedge i \notin A_p\}$.

CFIST is an ordered tree, where the labeling items of any node's children from left to right, and labeling items of nodes along any path from top down to bottom, follow the given ordering $\prec$. It can be used to represent the complete set of frequent itemsets as well as the closed set. In Example 1, given *minsup* of 3, alphabetic ordering of items $\prec$, the complete set of frequent itemsets is enumerated by the CFIST as shown in Fig. 1a.

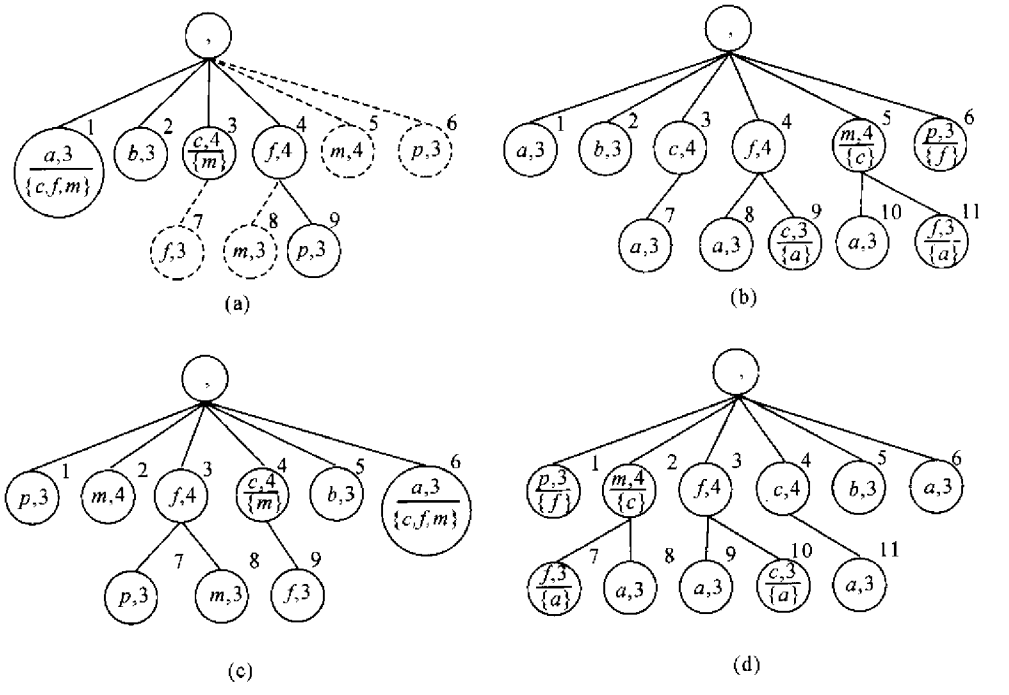

Fig. 1 Compound frequent itemset tree

(a) CFIST of TT type; (b) CFIST of TB type; (c) CFIST of BT type; (d) CFIST of BB type

The set of labeling items disjoint with any subsets of auxiliary sets of items of nodes along any path starting from the root represents a frequent itemset, with the weight of the ending node as the support of the itemset. Apparently, a path with compound nodes compresses several frequent itemsets. In Fig. 1a, node 3 compresses two frequent itemset $\{c\}$ and $\{c, m\}$, and the path from node 3 to node 7 compresses other two itemsets $\{c, f\}$ and $\{c, f, m\}$.

Definition 7 The set of all labeling items of nodes along any path starting from the root and ending at node p, is called a key itemset of p, denoted as $kis(p)$. The disjoint of all auxiliary sets of items along the path with $kis(p)$ is called a full itemset of p, denoted as $fis(p)$.

The full itemset of a CFIST node is a candidate of frequent closed itemset. The growth of CFIST is also a process for finding projected transaction subset, abbreviated as PTS, for each node p, which consists of transactions that support the $fis(p)$. The PTS of the root is $\boldsymbol{T}$, and any other PTS is derived by projecting the PTS of its parent.

Lemma 1 Given CFIST node p and $q \neq p$, $kis(p) \subset fis(q)$, then q must be created before p.

Proof Suppose that the sequences of nodes from the root to p and q are $p_1 \ldots p_m$ and $q_1 \ldots q_n$ respectively, there must be some k such that $p_k.item \neq q_k.item$, and $\forall j < k$,

$p_j.item = q_j.item$. According to Definition 6, $\forall i \in q_k.IL$, $q_k.item \prec i$. If $p_k.item \prec q_k.item$, then $p_k.item \notin fis(q)$, we get $kis(p) \not\subset fis(q)$ which is the contrary of the known condition. Therefore, $q_k.item \prec p_k.item$, in other words, q must be created before p.

The variants of CFIST

The CFIST given by Definition 6 is of standard type, or called TT type. Actually, either the sequence of the labeling items of any node's children from left to right, or the sequence of labeling items of nodes along any path from top down to bottom, can be arranged in the reverse ordering of $\prec$. As shown in Figs.1b, 1c and 1d are the variants, called CFIST of TB Type, of BT Type, and of BB Type respectively.

Lemma 2 Suppose CFIST is generated in left to right, top down direction, the following hold: (1) For CFIST of TT or BB type, if X_1 is generated before X_2, then $X_1 \not\subset X_2$; (2) For CFIST of TB or BT type, if X_1 is generated before X_2, then $X_2 \not\subset X_1$.

Merging CFIST nodes

Theorem 1 Given CFIST node p be the parent of node c, if $p.weight = c.weight$, then c can be merged into p.

Proof Let $p = (i_p, w_p, A_p, T_p, I_p)$, $c = (i_c, w_c, A_c, T_c, I_c)$, and the compound node obtained by merging c into p be $p' = (i_{p'}, w_{p'}, A_{p'}, T_{p'}, I_{p'})$. Because $T_c = g(\{i_c\}) \cap T_p \subseteq T_p$ and $w_c = w_p$, i.e., $\| T_c \| = \| T_p \|$, we have $T_p = T_c = T_{p'}$; that is, p' has the same closure as p and c. In addition, $I_c = \{i \in I_p \mid i_c \prec i \wedge i \notin A_p\} \subseteq I_p$, $i_c \notin I_c$, we have $I_c \subseteq I_p - \{i_c\} = I_{p'}$, therefore the closed set of frequent itemsets derived from p' must contain the set from p and c.

Corollary 2 If CFIST node c is a child of node p, then $c.weight < p.weight$.

By merging nodes, CFIST not only skips many levels of search space, but also saves lots of memory. For example, in Fig.1a, the node 1 will be a subtree with 4 levels and 9 nodes instead without merging nodes.

PRUNING TECHNIQUES

In order to further limit the search of solution space for the closed set of frequent itemsets, prompt tree pruning is critical.

Local pruning

Theorem 2 Given CFIST node p and n are brothers, if $p.weight = n.weight$ and $n.item \in p.AIS$, then n can be pruned.

Proof Let $p = (i_p, w_p, A_p, T_p, I_p)$, $n = (i_n, w_n, A_n, T_n, I_n)$, and parent $a = (i_a, w_a, A_a, T_a, I_a)$. Because $\{i_n\} \subseteq A_p \subseteq \{i_p\} \cup A_p$, so $T_n = g(\{i_n\}) \cap T_a \supseteq g(\{i_p\} \cup A_p) \cap T_a = T_p$. And $w_p = w_n$, i.e., $\| T_p \| = \| T_n \|$, hence $T_p = T_n$, p and n have the same closure. Because $i_n \in A_p$, $i_p \prec i_n$, $I_n \subseteq I_p$, therefore the set of closed itemsets derived from p must contain the set from n.

For example, in Fig.1a, node 3 and node 5 are brothers with the same support of 4, and the labeling item m of node 5 is contained in the auxiliary set of items of node 4, therefore node 5 can be pruned.

Global subsumption checking

Theorem 3 Given CFIST node p and $q \neq p$, if $p.weight = q.weight$ and $kis(p) \subset fis(q)$, then p can be pruned.

Proof Given $kis(p) \subset fis(q)$, so $g(kis(p)) \supset g(fis(q))$. Apparently, $g(kis(p)) = g(fis(p))$, so $g(fis(p)) \supset g(fis(q))$. Because $p.weight = q.weight$, i.e., $\| g(fis(p)) \| = \| g(fis(q)) \|$, therefore $g(fis(p)) = g(fis(q))$. Let r be a node such that $fis(r) = fis(p) \cup (fis(q)$, so $g(fis(r)) = g(fis(p) \cup fis(q)) = g(fis(p)) \cap g(fis(q))$. Therefore $g(fis(r)) = g(fis(p))$, and $fis(r) \supset fis(p)$, that is $fis(p)$ is not closed, and r is created before p, so p can be pruned.

In Fig.1a, the supports of node 7 and node 1 are of the same value 3, and the key itemset of node 7, $kis(node\ 7) = \{c, f\}$ is contained in the full itemset of node 1, $fis(node\ 1) = \{a, c, f, m\}$, therefore node 7 can be pruned. Because $kis(p)$ is a subset of $fis(p)$, pruning by applying Theorem 3 will be more efficient than directly comparing both full itemsets.

Fast hashing

If the full itemset of node p is subsumed by the full itemset of node q, then we have $fis(p) \subset fis(q)$ and $support(fis(p)) = support(fis(q))$. Therefore, an obvious way for subsump-

tion checking is to thread nodes into a hashing table according to their supports. But many unrelated itemsets may have the same support. So, this is not an efficient way. On the other hand, $fis(p) \subset fis(q)$ and $support(fis(p)) = support(fis(q))$ also means $g(fis(p)) = g(fis(q))$. However, direct comparison of the projected transaction subsets, i.e., the comparison of $g(fis(p))$ and $g(fis(q))$ is very expensive.

We adopt a compromise solution by using a linear function of $g(fis(p))$, the sum of tids of all transactions in $g(fis(p))$ as the hash key. Therefore, we check if $fis(p)$ be subsumed by $fis(q)$ as follows: if node p and q are threaded in the same hash bucket, if p and q have the same hash key, if p and q have the same support, and if $kis(p) \subset fis(q)$.

In Fig. 1a, the hash key of node 8, node 7 and node 1 are 8 where the former two nodes are subsumed by the latter. The hash key of node 2 is 9. The hash key of node 6 and node 9 are 10 where the former is subsumed by the latter. In Fig. 1a, nodes with dotted lines are pruned eventually, and the other five nodes form the closed set of frequent itemsets.

ALGORITHM

Our new algorithm is called CAP that is the abbreviation of mining Closed itemsets by Adaptive Pruning. CAP is built on the top of the algorithm OpportuneProject, Liu *et al*. (2002), which we proposed for mining complete set of frequent itemsets. CAP improves the performance of tree growth by adapting tree search strategy, the representation of PTS, and the methods of item counting in and projection creating of PTSs to the characteristics of PTSs. CAP balances the overheads of tree growth and tree pruning to maximize the efficiency and scalability.

A hybrid search strategy

Primarily, CAP grows the CFIST by depth first search, where PTSs are maintained in memory for all nodes on the path starting from the root to the node that is currently being explored. Depth first search avoids repetitive creations of PTSs that incurs CPU-bound pattern matching inherent to breadth first search. Depth first search is especially efficient for dense databases and for low support thresholds.

For very large datasets, CAP grows the CFIST breadth first at the beginning. Whenever the reduced transaction set that supports all nodes at level k can be represented by a memory based structure, CAP grows the lower portion under level k depth first.

Representing and projecting PTS

Representing sparse PTS by an array-based structure, TVLA: As shown in Fig. 2, in TVLA (threaded varied length arrays), each local frequent item has an entry in the item list (IL), each transaction is stored in an array, transactions with the same heading item (which need not be stored in the array) are threaded together by a linked queue (LQ) that is attached to the entry with the same item in IL. For example, the LQ(a) in Fig. 2 threads transactions 01, 02, and 05 that support the node 1, and the LQ(b) only threads part of the transactions that support the node 2 at that time.

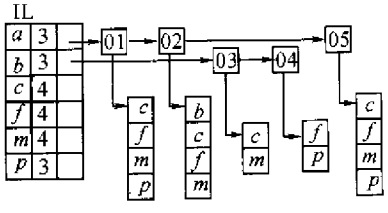

Fig. 2 Representing PTS by TVLA

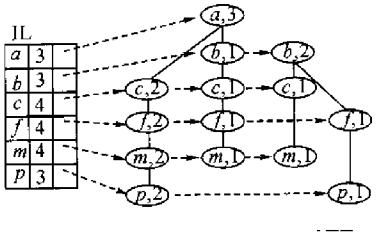

Fig. 3 Representing PTS by TTF

It is simple to get a child node's PTS from its parent node's PTS in the TVLA form. First, the transaction arrays that support a node's first child are already threaded by the LQ attached to the first entry of IL. By shifting transactions threaded in the LQ that are currently explored to subsequent LQs, we can get PTSs that support the second child, and so on. A transaction is shifted by threading it into a proper LQ according to the item next to the heading item. A child TVLA can share transaction arrays with its parent

TVLA that is unfiltered, or has its own local copy of transactions that trims out items irrelevant to further projection, i.e., filtered.

Representing dense PTS by a tree-based structure, TTF: As shown in Fig.3, TTF consists of an item list (IL), and a transaction forest. Each node in the forest is labeled by (i, w) where i is an item and w is a count that is the number of transactions represented by the path starting from a root ending at the node. All nodes labeled by the same item are threaded by the entry in IL with the same item. TTF is filtered if only local frequent items appear in TTF, otherwise unfiltered. For example, the filtered TTF representation for the PTS of the null root in Fig.2 is shown in Fig.3, where the path $(a,3)-(c,2)-(f,2)-(m,2)-(p,2)$ represents transaction 01 and 05. And the node $(c,2)$, $(c,1)$ and $(c,1)$ are threaded by the third entry of IL with support of 4.

The TTF representation of a child node's PTS can be derived by applying either bottom up or top down pseudo projection of its parent TTF. The child node's pseudo TTF shares the same memory area with its parent TTF.

Selecting the representation and projection method according to characteristics of PTS: CAP employs TVLA to represent PTS of nodes at high levels on CFIST, where PTSs are usually diversified and randomly distributed. They have less chance to share common prefix with each other. TTF does not compress transactions well. Since TTF needs much more additional storage overhead than TVLA, TTF is usually space expensive relative to TVLA in this case. On the other hand, at lower levels or denser branches on CFIST, where there are fewer local frequent items in PTSs and the relative support is larger, TTF compresses better. At that time CAP uses TTF to represent PTSs.

When a PTS shrinks sharply, CAP materializes its filtered child PTS, which is much more efficient than unfiltered or pseudo projection for further counting and projecting operations. On the opposite situation, CAP creates pseudo or unfiltered projections, which save memory and avoid expensive pattern matching operations incurred by recursive creation of PTSs, yet has little negative impact on the efficiency of projecting and counting operations.

Balancing CFIST growth vs. pruning overheads

In the growth process of CFIST of TB or BT type, the efficiency of tree growth is very high because pseudo or unfiltered projections can be applied. But, in this case, the non-closed branches are discovered later; therefore the efficiency of tree pruning is low. On the other hand, on CFIST of TT or BB type, the unclosed branches are discovered in time, the pruning operation is highly efficient, but the tree growth process is less efficient. CAP applies a hybrid type of CFIST, in that the upper levels of CFIST are of TT or BB type, and the lower levels are of TB or BT type.

The algorithm CAP

Now the algorithm CAP, mining Closed pattern by Adaptive Pruning, is given as shown in Fig.4.

```
CAP(T, I, ≺, minsup)
1)create CFIST root R: R.PTS = T; R.IL = I;
2)ClosedBF(R,0, ≺, minsup);
3)ClosedGDF(R, ≺, minsup);
ClosedBF(R, L, ≺, minsup)
4) for each t =(t.tid, t.items)∈R.PTS
5)   for each v at level L reached by projecting t
6)     for each i∈v.IL∩t.items
7)       support(i)++; tidsum(i)+=t.tid;
8)   if(t can't project to L)then remove t of R.PTS;
9) for each v at level L
10)  for each i∈v.IL with (w=support(i))≥minsup
11)    MLPSC(v, i, w, tidsum(i),, {j∈v.IL | i≺j});
12)if(NoMem(R.PTS))then ClosedBF(R, L+1, ≺, minsup)
ClosedGDF(v, ≺, minsup)
13)for each i∈v.IL
14)   T=g({i})∩v.PTS;
15)   ts=SumOfTids(T);
16)   I={j∈v.IL | i≺j};
17)   if((w=||T||))≥minsup &&(c=MLPSC(v,i,w,ts,T,I))
18)   then ClosedGDF(c, ≺, minsup);
```

Fig.4 The algorithm CAP

First, CAP calls ClosedBF to grow the upper portion of CFIST by breadth first search until the reduced set of transactions can be held in a memory based structure. Then, ClosedDF is called to build the lower portion of CFIST by depth first search. MLPSC is responsible for the generation of nodes, merging of nodes, local pruning, and global subsumption checking.

EXPERIMENTAL EVALUATIONS

The experiments were performed on an 800

MHz Pentium IV PC with 512 MB of main memory and 20 GB of hard drive, running Microsoft Windows 2000 Server. The executables of comparative algorithms, including CHARM, optimized version of CLOSET, and MAFIA with the option of closed sets, are provided by the original authors.

Four datasets are used in this evaluation. T25i20d100k is an artificial dataset generated by a data generator obtained from IBM Almaden①, and is regarded as something between the sparse and the dense. Connect4 is a very dense dataset from UCI Machine Learning Repository②. Bms-pos and bms-webview-1 are real world datasets provided by Blue Martini Software Inc., and are categorized as sparse datasets.

The performance measures are execution times and memory usages of the algorithms on the datasets with different minimum support thresholds. For the convenience of analysis, the maximum itemset length and number of frequent closed itemsets are also given with the minimum support threshold. For example, 0.3%(3; 3 K) means at minimum support threshold of 0.3% the maximum itemset length is 3 and the size of the closed set of frequent itemsets is 3 K. Fig.5 shows the execution time curves for the four algorithms on the four datasets respectively. The vertical axis displays execution time on a logarithmic scale, and the horizontal axis displays relative support thresholds (%).

On T25i20d100k as shown in Fig.5a, at the support threshold of 0.3%(3; 3 K), MAFIA requires 1403 seconds; the other three algorithms requires 3 to 7 seconds. Lower than 0.3%, CAP is more efficient than CHARM by a factor of five, and is one and three orders of magnitude more efficient than CLOSET and MAFIA respectively. For example, at 0.2%(13; 56 K), CAP requires 4 seconds, CHARM requires 20 seconds, CLOSET requires 51 seconds, and MAFIA requires 4779 seconds. At 0.175% (22; 2.3 M), CAP requires 33 seconds, and CHARM requires 168 seconds, while CLOSET and MAFIA fail because of being out of memory. Lower than 0.175%, CHARM also fails. Moreover, CAP only requires 165 seconds even at 0.13%(26; 7.8 M).

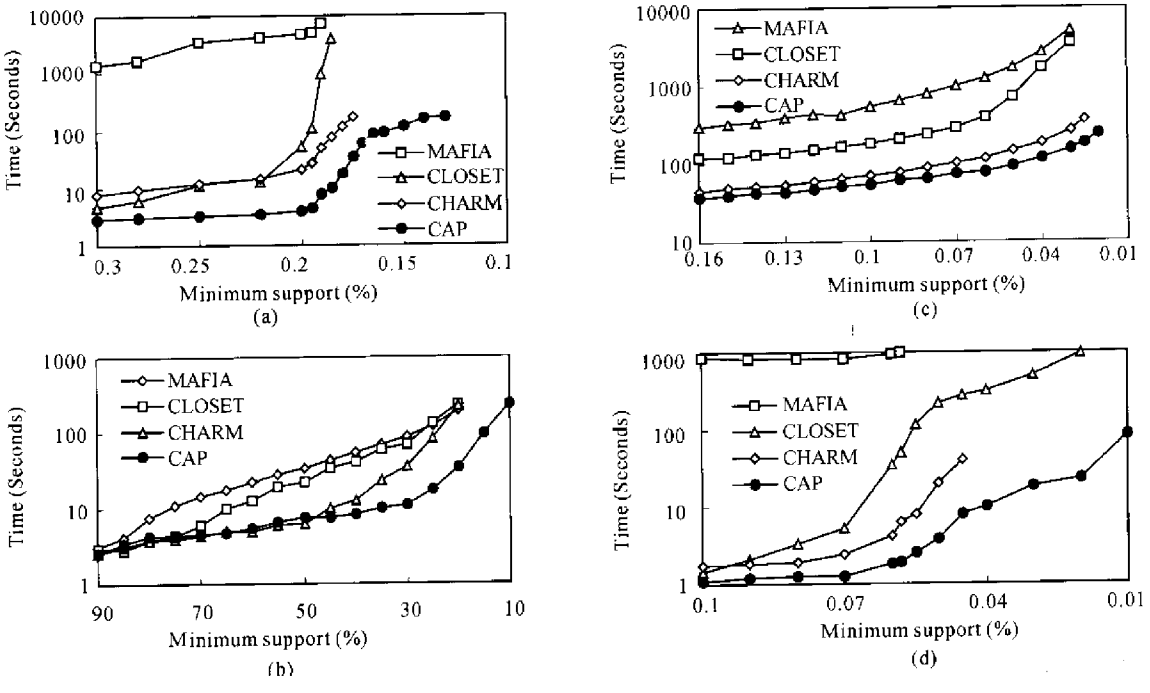

Fig.5 Performance comparison of CAP with CHARM, MAFIA, and CLOSET
(a) on t25i20d100k; (b) on connect4; (c) on bms-pos; (d) on bms-webview-1

① http://www.almaden.ibm.com/cs/quest/syndata.html
② http://www.ics.uci.edu/mlearn/MLRepository.html

On connect4 as shown in Fig.5b, CAP and CLOSET are more efficient than MAFIA and CHARM. At the support threshold lower than 50% (21; 153 K), CAP is more efficient than CLOSET too. For example, at 30% (25; 563 K), CAP finishes in 11 seconds, CLOSET finishes in 36 seconds, MAFIA finishes in 70 seconds, and CHARM finishes in 86 seconds. At 20%(27; 1.8 M), CAP finishes in 34 seconds, and the other three finishes in around 200 seconds. Lower than 20%, CLOSET, MAFIA and CHARM fail because of being out of memory. Moreover, CAP finishes in only 236 seconds even at 10%(29; 9.95 M).

On bms-pos as shown in Fig.5c, CAP is one order of magnitude more efficient than CLOSET and MAFIA. CAP outperforms CHARM by a factor of two at low support levels. For example, at 0.03%(12; 1.7 M), the execution time of CAP is 154 seconds, CHARM is 268, MAFIA is 3540, and CLOSET is 5138. Lower than 0.03%, the latter two algorithms fail because of being out of memory. At 0.025%(12; 2.6 M), CAP finishes in 180 seconds, and CHARM finishes in 356 seconds. At 0.02%(12; 4.3 M), CAP finishes in 237 seconds, while CHARM fails because of being out of memory.

On bms-webview-1 as shown in Fig. 5d, MAFIA can hardly run because of huge memory overhead. CAP is one order of magnitude more efficient than CLOSET. At support threshold lower than 0.05%, CAP outperforms CHARM by a factor of five. For example, at 0.05%(45; 127 K), CAP requires 4 seconds, CHARM requires 20 seconds, and CLOSET requires 216 seconds. At 0.045% (99; 139 K), CAP requires 8 seconds, CHARM requires 40 seconds, and CLOSET requires 270 seconds. At 0.04%(102; 155 K), CHARM fails because of being out of memory, CLOSET requires 323 seconds, while CAP requires only 10 seconds. Even at 0.01%(154; 1.2 M), CAP requires only 84 seconds.

In short, CAP outperforms CHARM by a factor of five, and is one to three orders of magnitude more efficient than CLOSET and MAFIA. Moreover, CAP is the most scalable algorithm.

ACKNOWLEDGEMENT

We would like to thank Mohammed J. Zaki for providing us the executable of CHARM, Doug Burdick for MAFIA, Jiawei Han for CLOSET, and Zijian Zheng for giving us access to the datasets bms-pos and bms-webview-1.

References

Agarwal, R., Aggarwal, C. and Prasad, V., 2000. Depth First Generation of Long Patterns. *In*: The 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, USA.

Agrawal, R. and Srikant, R., 1994. Fast Algorithms for Mining Association Rules. *In*: VLDB'94, Santiago, Chile, p.487 – 499.

Burdick, D., Calimlim, M. and Gehrke, J., 2001. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. *In*: The 17th International Conference on Data Engineering, Heidelberg, Germany.

Liu, J., Pan, Y., Wang, K. and Han, J., 2002. Mining Frequent Itemsets by Opportunistic Projection. *In*: The 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Alberta, Canada, p. 229 – 238.

Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L., 1998. Pruning Closed Itemset Lattices for Association Rules. *In*: The BDA French Conference on Advanced Databases, France.

Pasquier, N., Bastide, Y., Taouil, R. and Lakhal., L., 1999. Discovering Frequent Closed Itemsets for Association Rules. *In* : ICDT'99, Jerusalem, Israel, p.398 – 416.

Pei, J., Han, J. and Mao, R., 2000. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. *In*: The ACM-SIGMOD International Workshop on Data Mining and Knowledge Discovery, Dallas, TX.

Wang, K., Liu, T., Han, J. and Liu, J., 2002. Top Down FP-Growth for Association Rule Mining. *In*: The 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Taipei, p.334 – 340.

Zaki, M.J. and Hsiao, C.J., 2002. CHARM: An Efficient Algorithm for Closed Itemset Mining. *In*: The 2nd SIAM International Conference on Data Mining, Arlington, VA, USA.