Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure

Mohammed J. Zaki, *Member*, *IEEE*, and Ching-Jui Hsiao

Abstract—The set of frequent *closed* itemsets uniquely determines the exact frequency of all itemsets, yet it can be orders of magnitude smaller than the set of all frequent itemsets. In this paper, we present CHARM, an efficient algorithm for mining all frequent closed itemsets. It enumerates closed sets using a dual itemset-tidset search tree, using an efficient hybrid search that skips many levels. It also uses a technique called *diffsets* to reduce the memory footprint of intermediate computations. Finally, it uses a fast hash-based approach to remove any "nonclosed" sets found during computation. We also present CHARM-L, an algorithm that outputs the closed itemset lattice, which is very useful for rule generation and visualization. An extensive experimental evaluation on a number of real and synthetic databases shows that CHARM is a state-of-the-art algorithm that outperforms previous methods. Further, CHARM-L explicitly generates the frequent closed itemset lattice.

Index Terms—Closed itemsets, frequent itemsets, closed itemset lattice, association rules, data mining.

1 Introduction

Mining frequent patterns or itemsets is a fundamental and essential problem in many data mining applications. These applications include the discovery of association rules, strong rules, correlations, sequential rules, episodes, multidimensional patterns, and many other important discovery tasks [11]. The problem is formulated as follows: Given a large database of item transactions, find all frequent itemsets, where a frequent itemset is one that occurs in at least a user-specified percentage of the database.

Most of the proposed pattern-mining algorithms are a variant of Apriori [1]. Apriori employs a breadth-first search (BFS) that enumerates every single frequent itemset. Apriori uses the *downward closure* property of itemset support to prune the search space—the property that all subsets of a frequent itemset must be frequent. Thus, only the known frequent itemsets at one level are extended by one more item to yield potentially frequent "candidate" itemsets at the next level in the BFS. A pass over the database is made at each level to find the true frequent itemsets among the candidates. Apriori-inspired algorithms [5], [15], [18] show good performance with sparse data sets such as market-basket data, where the frequent patterns are very short. However, with dense data sets such as telecommunications and census data, where there are many, long frequent patterns, the performance of these algorithms degrades incredibly. A frequent pattern of length l implies the presence of $2^l - 2$ additional frequent patterns as well, each of which is explicitly examined by such algorithms. It is practically unfeasible to mine the set of all frequent patterns for other than small l. On the other hand, in many real-world problems (e.g., patterns in biosequences, census data, etc.), finding long itemsets of length 30 or 40 is not uncommon [4].

There are two current solutions to the long pattern mining problem. The first one is to mine only the maximal frequent itemsets [2], [4], [6], [10], [13], which are typically orders of magnitude fewer than all frequent patterns. While mining maximal sets help understand the long patterns in dense domains, they lead to a loss of information; since subset frequency is not available, maximal sets are not suitable for generating rules. The second is to mine only the frequent closed sets [3], [16], [17], [20], [21]. Closed sets are *lossless* in the sense that they can be used to uniquely determine the set of all frequent itemsets and their *exact* frequency. At the same time, closed sets can themselves be orders of magnitude smaller than all frequent sets, especially on dense databases.

1.1 Contributions

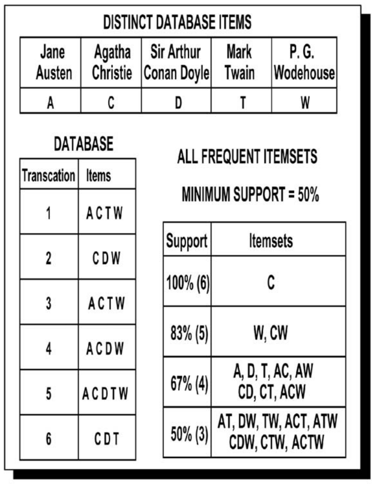
We introduce CHARM,[1] an efficient algorithm for enumerating the set of all frequent closed itemsets, and CHARM-L, an efficient algorithm for generating the closed itemset lattice. There are a number of innovative ideas employed in the development of CHARM(-L); these include:

1. They simultaneously explore both the itemset space and transaction space over a novel *IT-tree* (itemset-tidset tree) search space. In contrast, most previous methods exploit only the itemset search space.
2. They use a highly efficient hybrid search method that skips many levels of the IT-tree to quickly identify the frequent closed itemsets, instead of having to enumerate many possible subsets.
3. CHARM uses a fast hash-based approach and CHARM-L uses an intersection-based approach to eliminate nonclosed itemsets during subsumption checking. Both algorithms utilize the novel vertical

• *The authors are with the Computer Science Department, Rensselaer Polytechnic Institute, Troy NY 12180. E-mail: {zaki, hsiao}@cs.rpi.edu.*

Manuscript received 16 Dec. 2002; revised 7 Feb. 2004; accepted 30 Aug. 2004; published online 17 Feb. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 117980.

1. CHARM stands for Closed Association Rule Mining; the "H" is gratuitous.



DISTINCT DATABASE ITEMS

Jane Austen	Agatha Christie	Sir Arthur Conan Doyle	Mark Twain	P. G. Wodehouse
A	C	D	T	W

DATABASE

Transcation	Items
1	A C T W
2	C D W
3	A C T W
4	A C D W
5	A C D T W
6	C D T

ALL FREQUENT ITEMSETS

MINIMUM SUPPORT = 50%

Support	Itemsets
100% (6)	C
83% (5)	W, CW
67% (4)	A, D, T, AC, AW CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW CDW, CTW, ACTW

Fig. 1. Example DB.

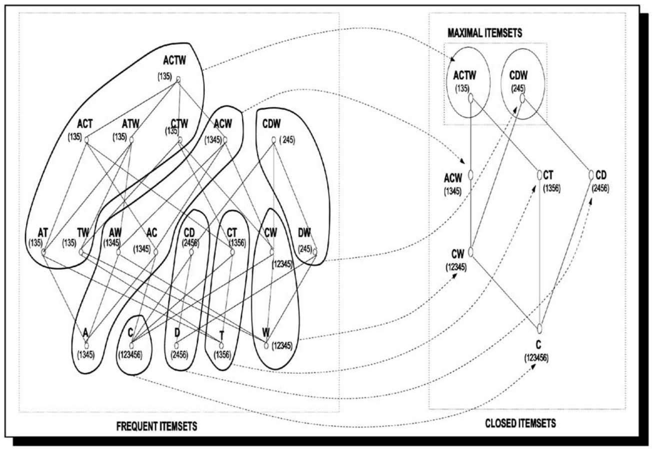


Fig. 2. Frequent, closed, and maximal itemsets.

data representation called *diffsets* [23], recently proposed by us, for fast frequency computations. Diffsets keep track of differences in the tids of a candidate pattern from its prefix pattern. Diffsets drastically cut down (by orders of magnitude) the size of memory required to store intermediate results. Thus, the entire working set of patterns can fit entirely in main-memory, even for large databases.

4. CHARM-L explicitly outputs the frequent itemset lattice, which is useful for rule generation and visualization.

We assume in this paper that the initial database is disk-resident, but that the intermediate results fit entirely in memory. Several factors make this a realistic assumption. First, CHARM(-L) breaks the search space into small independent chunks (based on prefix equivalence classes [22]). Second, diffsets lead to extremely small memory footprint (this is experimentally verified). Finally, CHARM(-L) uses simple set difference (or intersection) operations and requires no complex internal data structures (candidate generation and counting happens in a single step). The current trend toward large (gigabyte-sized) main memories, combined with the above features, makes CHARM and CHARM-L practical and efficient algorithms for reasonably large databases.

We compare CHARM against previous methods for mining closed sets such as Close [16], Closet [17], Closet+ [20], Mafia [6], and Pascal [3]. Extensive experiments confirm that CHARM provides significant improvement over existing methods for mining closed itemsets, for both dense as well as sparse data sets. We also compare and show that CHARM-L outperforms an approach that generates the itemset lattice in a postprocessing step.

2 Frequent Pattern Mining

Let $\mathcal{I}$ be a set of items and $\mathcal{D}$ a database of transactions, where each transaction has a unique identifier (*tid*) and contains a set of items. The set of all tids is denoted as $\mathcal{T}$. A set $X \subseteq \mathcal{I}$ is also called an *itemset* and a set $Y \subseteq \mathcal{T}$ is called a *tidset*. An itemset with k items is called a k-itemset. For convenience, we write an itemset $\{A, C, W\}$ as ACW and a tidset $\{2, 4, 5\}$ as 245. For an itemset X, we denote its corresponding tidset as $t(X)$, i.e., the set of all tids of transactions that contain X as a subset. For a tidset Y, we denote its corresponding itemset as $i(Y)$, i.e., the set of items common to all the transactions with tids in Y. Note that $t(X) = \bigcap_{x \in X} t(x)$, and $i(Y) = \bigcap_{y \in Y} i(y)$. For example, in Fig. 1,

$$\begin{aligned} t(ACW) &= t(A) \cap t(C) \cap t(W) \\ &= 1345 \cap 123456 \cap 12345 \\ &= 1345 \end{aligned}$$

and $i(12) = i(1) \cap i(2) = ACTW \cap CDW = CW$. We use the notation $X \times t(X)$ to refer to an itemset-tidset pair and call it an *IT-pair*.

The *support* [1] of an itemset X, denoted $\sigma(X)$, is the number of transactions in which it occurs as a subset, i.e., $\sigma(X) = |t(X)|$. An itemset is *frequent* if its support is greater than or equal to a user-specified *minimum support* (*min_sup*) value, i.e., if $\sigma(X) \geq min\_sup$. A frequent itemset is called *maximal* if it is not a subset of any other frequent itemset. Let $c : P(\mathcal{I}) \rightarrow P(\mathcal{I})$ be the *closure* operator, defined as $c(X) = i(t(X))$, where $X \subseteq \mathcal{I}$. An frequent itemset X is called *closed* if and only if $c(X) = X$ [9]. Alternatively, a frequent itemset X is closed if there exists no proper superset $Y \supset X$ with $\sigma(X) = \sigma(Y)$. For instance, in Fig. 1,

$$c(AW) = i(t(AW)) = i(1345) = ACW.$$

Thus, AW is not closed. On the other hand,

$$c(ACW) = i(t(ACW)) = i(1345) = ACW,$$

thus ACW is closed.

As a running example, consider the database shown in Fig. 1. There are five different items, $\mathcal{I} = \{A, C, D, T, W\}$, and six transactions, $\mathcal{T} = \{1, 2, 3, 4, 5, 6\}$. The table on the right shows all 19 frequent itemsets contained in at least three transactions, i.e., $min\_sup = 50\%$. Fig. 2 shows the 19 frequent itemsets organized as a subset lattice; their corresponding tidsets have also been shown. In contrast to the 19 frequent itemsets, there are only 7 closed itemsets, obtained by collapsing all the itemsets that have the same tidset, shown in the figure by the enclosed regions. Looking at the closed itemset lattice, we find that there are 2 maximal frequent itemsets (marked with a circle), $ACTW$ and CDW.

Let $\mathcal{F}$ denote the set of frequent itemsets, $\mathcal{C}$ the set of closed itemsets, and $\mathcal{M}$ the set of maximal itemsets. By definition, a frequent closed set is a frequent itemset and a maximal frequent itemset is closed, giving us the fact that $\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$. Fig. 1 depicts this relationship on our example database. Theoretically, in the worst case, there can be $2^{|\mathcal{I}|}$ frequent and closed frequent itemsets (i.e., every frequent set can also be closed) and $\binom{|\mathcal{I}|}{|\mathcal{I}|/2} = 2^{|\mathcal{I}|/2}$ maximal frequent itemsets. In practice, however, $\mathcal{C}$ can be orders of magnitude smaller than $\mathcal{F}$ (especially for dense data sets), while $\mathcal{M}$ can itself be orders of magnitude smaller than $\mathcal{C}$. The closed sets are also lossless in the sense that the exact frequency of all frequent sets can be determined from $\mathcal{C}$, while $\mathcal{M}$ leads to a loss of information (since subset frequency is not kept).

2.1 Related Work

Our past work [21], [25] addressed the problem of nonredundant rule generation, provided that closed sets are available; an algorithm to efficiently mine the closed sets was not described in that paper. There have been several algorithms proposed for this task.

Close [16] is an Apriori-like algorithm that directly mines frequent closed itemsets. There are two main steps in Close. The first is to use bottom-up search to identify *generators*, the smallest frequent itemsets that determines a closed itemset. For example, consider the frequent itemset lattice in Fig. 2. The item A is a generator for the closed set ACW since it is the smallest itemset with the same tidset as ACW. All generators are found using a simple modification of Apriori. After finding the frequent sets at level k, Close compares the support of each set with its subsets at the previous level. If the support of an itemset matches the support of any of its subsets, the itemset cannot be a generator and is thus pruned. The second step in Close is to compute the closure of all the generators found in the first step. To compute the closure of an itemset, we have to perform an intersection of all transactions where it occurs as a subset. The closures for all generators can be computed in just one database scan, provided all generators fit in memory. Nevertheless, computing closures this way is an expensive operation.

The authors of Close recently developed Pascal [3], an improved algorithm for mining closed and frequent sets. They introduce the notion of *key patterns* and show that other frequent patterns can be inferred from the key patterns without access to the database. They show that Pascal, even though it finds both frequent and closed sets, is typically twice as fast as Close, and 10 times as fast as Apriori. Since Pascal enumerates all patterns, it is only practical when pattern length is short (as we shall see in the experimental section). The *Closure* algorithm [7] is also based on a bottom-up search. It performs only marginally better than Apriori, thus CHARM should outperform it easily.

Recently, two new algorithms for finding frequent closed itemsets have been proposed. Closet [17] uses a novel frequent pattern tree (FP-tree [12]) structure, which is a compressed representation of all the transactions in the database. It uses a recursive divide-and-conquer and database projection approach to mine long patterns. Closet+ [20] is an enhancement of Closet with various new and previously known search and closure testing strategies. We will show later that CHARM outperforms Closet and Closet+ by orders of magnitude as support is lowered. Mafia [6] is primarily intended for maximal pattern mining, but has an option to mine the closed sets as well. Mafia relies on efficient compressed and projected vertical bitmap based frequency computation. At higher supports, both Mafia and CHARM exhibit similar performance, but, as one lowers the support, the gap widens exponentially. CHARM can deliver over a factor of 10 improvements over Mafia for low supports.

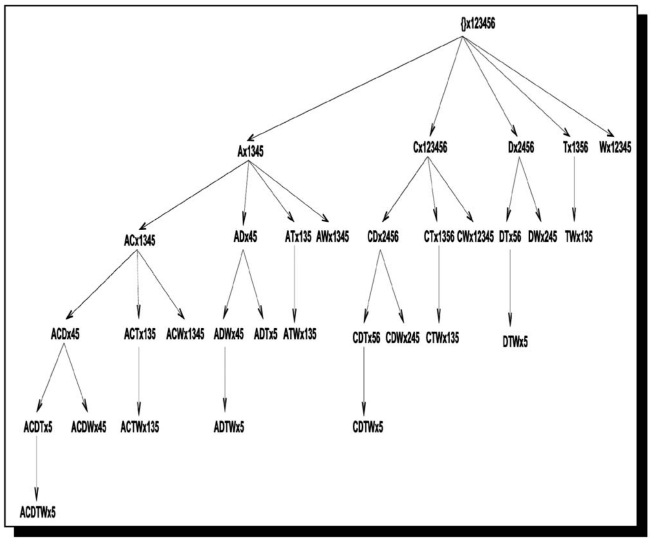


Fig. 3. IT-tree: itemset-tidset search tree.

There have been several efficient algorithms for mining maximal frequent itemsets, such as MaxMiner [4], Depth-Project [2], Mafia [6], and GenMax [10]. It is not practical to first mine maximal patterns and then to check if each subset is closed since we would have to check 2^l subsets, where l is the length of the longest pattern (we can easily have patterns of length 30 to 40 or more; see Section 6). In [24], we tested a modified version of MaxMiner to discover closed sets in a postprocessing step and found it to be too slow for all except short patterns.

3 Itemset-Tidset Search Tree and Equivalence Classes

Let $\mathcal{I}$ be the set of items. Define a function $p(X,k) = X[1:k]$ as the k length prefix of X and a *prefix-based* equivalence relation θ_k [22] on itemsets as follows: $\forall X,\ Y \subseteq \mathcal{I}$, $X \equiv_{\theta_k} Y \Leftrightarrow p(X,k) = p(Y,k)$. That is, two itemsets are in the same class if they share a common k length prefix.

CHARM performs a search for closed frequent sets over a novel IT-tree search space, shown in Fig. 3. Each node in the IT-tree, represented by an itemset-tidset pair, $X \times t(X)$, is in fact a prefix-based class. All the children of a given node X belong to its equivalence class since they all share the same prefix X. We denote an equivalence class as $[P] = \{l_1, l_2, \cdots, l_n\}$, where P is the parent node (the prefix), and each l_i is a single item, representing the node

```
ENUMERATE-FREQUENT($[P]$):
  for all $l_i \in [P]$ do
    $[P_i] = \emptyset$
    for all $l_j \in [P]$, with $j > i$ do
      $I = l_j$
      $T = t(l_i) \cap t(l_j)$
      if $|T| \geq min\_sup$ then
        $[P_i] = [P_i] \cup \{I \times T\}$
    Enumerate-Frequent($[P_i]$)
    delete $[P_i]$
```

Fig. 4. Pattern enumeration: depth first search.

$Pl_i \times t(Pl_i)$. For example, the root of the tree corresponds to the class $[] = \{A, C, D, T, W\}$. The leftmost child of the root consists of the class $[A]$ of all itemsets containing A as the prefix, i.e., the set $\{C, D, T, W\}$. As can be discerned, each class member represents one child of the parent node. A class represents items that the prefix can be extended with to obtain a new frequent node. Clearly, no subtree of an infrequent prefix has to be examined. The power of the equivalence class approach is that it breaks the original search space into *independent* subproblems. For any subtree rooted at node X, one can treat it as a completely new problem; one can enumerate the patterns under it and simply prefix them with the item X and so on.

Frequent pattern enumeration is straightforward in the IT-tree framework. For a given node or prefix class, one can perform intersections of the tidsets of all pairs of elements in a class and check if min_sup is met; support counting is simultaneous with generation. Each resulting frequent itemset is a class unto itself, with its own elements, that will be recursively expanded. That is to say, for a given class of itemsets with prefix P, $[P] = \{l_1, l_2, \cdots, l_n\}$, one can perform the intersection of $t(l_i)$ with all $t(l_j)$ with $j > i$ to obtain a new class of frequent extensions, $[Pl_i] = \{l_j \mid j > i$ and $\sigma(Pl_il_j) \geq min\_sup\}$. For example, from the null root $[] = \{A, C, D, T, W\}$, with $min\_sup = 50\%$, we obtain the classes $[A] = \{C, T, W\}$, $[C] = \{D, T, W\}$, and $[D] = \{W\}$, and $[W] = \{\}$. Note that class $[A]$ does not contain D since AD is not frequent. Fig. 4 gives a pseudocode description of a depth first (DFS) exploration of the IT-tree for all frequent patterns. CHARM improves upon this basic enumeration scheme, using the conceptual framework provided by the IT-tree; it is not assumed that all the tidsets will fit in memory, rather CHARM materializes only a small portion of the tree in memory at any given time.

3.1 Basic Properties of Itemset-Tidset Pairs

For any two nodes in the IT-tree, $X_i \times t(X_i)$ and $X_j \times t(X_j)$, if $X_i \subseteq X_j$ then it is the case that $t(X_j) \subseteq t(X_i)$. For example, for $ACW \subseteq ACTW$, $t(ACW) = 1345 \supseteq 135 = t(ACTW)$. Let us define $f : \mathcal{P}(\mathcal{I}) \mapsto N$ to be a one-to-one mapping from itemsets to integers. For any two itemsets X_i and X_j, we say $X_i \leq_f X_j$ iff $f(X_i) \leq f(X_j)$. The function f defines a total order over the set of all itemsets. For example, if f denotes the lexicographic ordering, then itemset $AC \leq AD$, but if f sorts itemsets in increasing order of their support, then $AD \leq AC$ if $\sigma(AD) \leq \sigma(AC)$. There are four basic properties of IT-pairs (pictorially depicted in Fig. 5) that CHARM leverages for fast exploration of closed sets. Assume that we are currently processing a node $P \times t(P)$, where $[P] = \{l_1, l_2, \cdots, l_n\}$ is the prefix class. Let X_i denote the itemset Pl_i, then each member of $[P]$ is an IT-pair $X_i \times t(X_i)$.

Theorem 1. *Let $X_i \times t(X_i)$ and $X_j \times t(X_j)$ be any two members of a class $[P]$. The following four properties hold:*

1. *If $t(X_i) = t(X_j)$, then $c(X_i) = c(X_j) = c(X_i \cup X_j)$.*
2. *If $t(X_i) \subset t(X_j)$, then $c(X_i) \neq c(X_j)$, but*
$$c(X_i) = c(X_i \cup X_j).$$
3. *If $t(X_i) \supset t(X_j)$, then $c(X_i) \neq c(X_j)$, but*
$$c(X_j) = c(X_i \cup X_j).$$

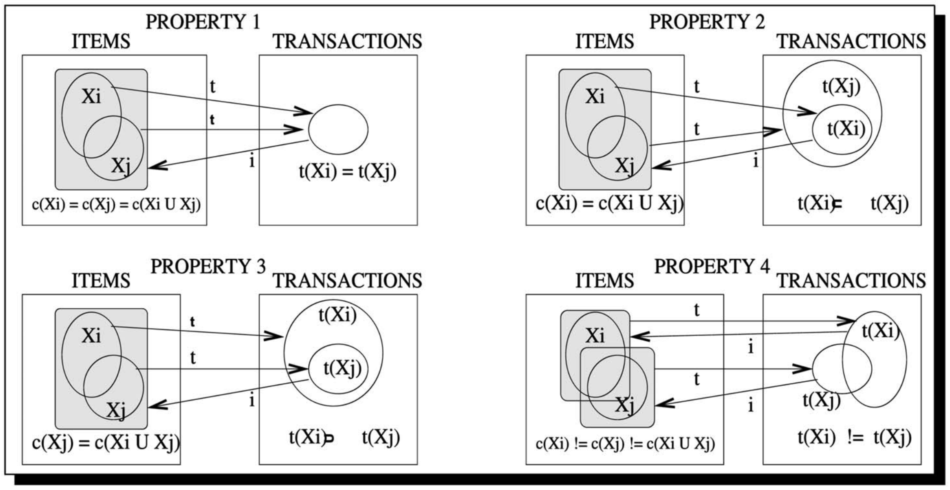


Fig. 5. Basic properties of itemsets and tidsets.

```
CHARM ($\mathcal{D}$, $min\_sup$):
1.  $[\emptyset] = \{l_i \times t(l_i) : l_i \in \mathcal{I} \wedge \sigma(l_i) \geq min\_sup\}$
2.  CHARM-EXTEND ($[\emptyset], \mathcal{C} = \emptyset$)
3.  return $\mathcal{C}$ //all closed sets

CHARM-EXTEND ($[P], \mathcal{C}$):
4.  for each $l_i \times t(l_i)$ in $[P]$
5.      $P_i = P \cup l_i$ and $[P_i] = \emptyset$
6.      for each $l_j \times t(l_j)$ in $[P]$, with $j > i$
7.          $X = l_j$ and $Y = t(l_i) \cap t(l_j)$
8.          CHARM-PROPERTY($X \times Y, l_i, l_j, P_i, [P_i], [P]$)
9.      SUBSUMPTION-CHECK($\mathcal{C}, P_i$)
10.     CHARM-EXTEND ($[P_i], \mathcal{C}$)
11.     delete $[P_i]$

CHARM-PROPERTY ($X \times Y, l_i, l_j, P_i, [P_i], [P]$):
12. if ($\sigma(X) \geq minsup$) then
13.     if $t(l_i) = t(l_j)$ then //Property 1
14.         Remove $l_j$ from $[P]$
15.         $P_i = P_i \cup l_j$
16.     else if $t(X_i) \subset t(X_j)$ then //Property 2
17.         $P_i = P_i \cup l_j$
18.     else if $t(X_i) \supset t(X_j)$ then //Property 3
19.         Remove $l_j$ from $[P]$
20.         Add $X \times Y$ to $[P_i]$ //use ordering $f$
21.     else if $t(X_i) \neq t(X_j)$ then //Property 4
22.         Add $X \times Y$ to $[P_i]$ //use ordering $f$
```

Fig. 6. The CHARM Algorithm.

4. *If $t(X_i) \not\subseteq t(X_j)$ and $t(X_i) \not\supseteq t(X_j)$, then*

$$c(X_i) \neq c(X_j) \neq c(X_i \cup X_j).$$

Proof and Discussion.

1. If $t(X_i) = t(X_j)$, then, obviously, $i(t(X_i)) = i(t(X_j)$, i.e., $c(X_i) = c(X_j)$. Further, $t(X_i) = t(X_j)$ implies that $t(X_i \cup X_j) = t(X_i) \cap t(X_j) = t(X_i)$. Thus,

$$i(t(X_i \cup X_j)) = i(t(X_i)),$$

giving us $c(X_i \cup X_j) = c(X_i)$. This property implies that we can replace every occurrence of X_i with $X_i \cup X_j$ and we can remove the element X_j from further consideration since its closure is identical to the closure of $X_i \cup X_j$.

2. If $t(X_i) \subset t(X_j)$, then

$$t(X_i \cup X_j) = t(X_i) \cap t(X_j) = t(X_i) \neq t(X_j),$$

giving us $c(X_i \cup X_j) = c(X_i) \neq c(X_j)$. Thus, we can replace every occurrence of X_i with $X_i \cup X_j$ since they have identical closures. But, since $c(X_i) \neq c(X_j)$, we cannot remove element X_j from class $[P]$; it generates a different closure.

3. Similar to Case 2 above.

4. If $t(X_i) \not\subseteq t(X_j)$ and $t(X_i) \not\supseteq t(X_j)$, then clearly $t(X_i \cup X_j) = t(X_i) \cap t(X_j) \neq t(X_i) \neq t(X_j)$, giving us $c(X_i \cup X_j) \neq c(X_i) \neq c(X_j)$. No element of the class can be eliminated; both X_i and X_j lead to different closures (neither is a subset of the other). □

4 CHARM: Algorithm Design and Implementation

We now present CHARM, an efficient algorithm for mining all the closed frequent itemsets. We will first describe the algorithm in general terms, independent of the implementation details. We then show how the algorithm can be implemented efficiently. CHARM simultaneously explores both the itemset space and tidset space using the IT-tree, unlike previous methods which typically exploit only the itemset space. CHARM uses a novel search method, based on the IT-pair properties, that skips many levels in the IT-tree to quickly converge on the itemset closures, rather than having to enumerate many possible subsets.

The pseudocode for CHARM appears in Fig. 6. The algorithm starts by initializing the prefix class $[P]$, of nodes to be examined to the frequent single items and their tidsets ($l_i \times t(l_i),\ l_i \in \mathcal{I}$) in Line 1. We assume that the elements in $[P]$ are ordered according to a suitable total order f. The main computation is performed in CHARM-EXTEND which returns the set of closed frequent itemsets $\mathcal{C}$.

CHARM-EXTEND is responsible for considering each combination of IT-pairs appearing in the prefix class $[P]$. For each IT-pair $l_i \times t(l_i)$ (Line 4), it combines it with the other IT-pairs $l_j \times t(l_j)$ that come after it (Line 6) according to the total order f. Each l_i generates a new prefix, $P_i = P \cup l_i$, with class $[P_i]$, which is initially empty (Line 5). At line 7, the two IT-pairs are combined to produce a new pair $X \times Y$, where $X = l_j$ and $Y = t(l_i) \cap t(l_j)$. Line 8 tests which of the four IT-pair properties can be applied by calling CHARM-PROPERTY. Note that this routine may modify the current class $[P]$ by deleting IT-pairs that are already subsumed by other pairs. It also inserts the newly generated IT-pairs in the new class $[P_i]$. It can also modify the prefix P_i in case Properties 1 and 2 hold. We then insert the itemset P_i in the set of closed itemsets $\mathcal{C}$ (Line 9), provided that P_i is not subsumed by a previously found closed set (we later describe how to perform fast subsumption checking). Once all l_j have been processed, we recursively explore the new class $[P_i]$ in a depth-first manner (Line 10). After we return, any closed itemset containing P_i has already been generated. We then return to Line 4 to process the next (unpruned) IT-pair in $[P]$.

Dynamic element reordering. We purposely let the IT-pair ordering function in Line 6 remain unspecified. The usual manner of processing is in lexicographic order, but we can specify any other total order we want. The most promising approach is to sort the itemsets based on their support. The motivation is to increase opportunity for pruning elements from a class $[P]$. A quick look at Properties 1 and 2 tells us that these two cases are preferable over the other two. For Property 1, the closure of the two

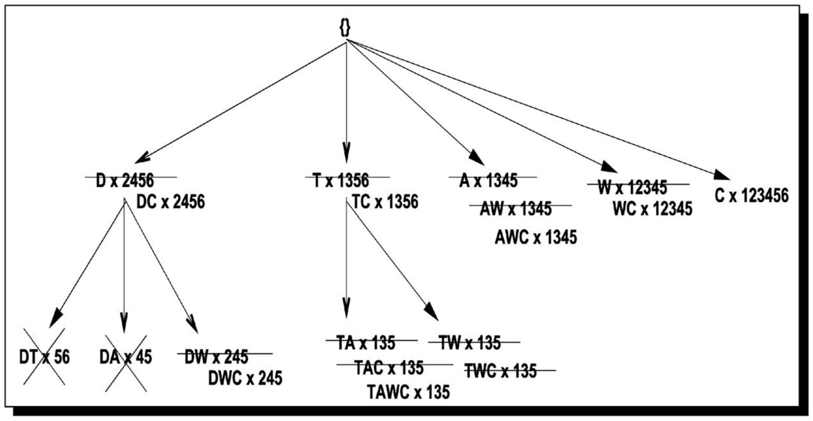


Fig. 7. Search process using tidsets.

itemsets is equal and, thus, we can discard l_j from $[P_i]$ and replace P_i with $P_i \cup l_j$. For Property 2, we can still replace P_i with $P_i \cup l_j$. Note that in both of these cases, we do not insert anything in the new class $[P_i]$! Thus, the more the occurrence of Cases 1 and 2, the fewer levels of search we perform. In contrast, the occurrence of Cases 3 and 4 results in additions to the set of new nodes, requiring additional levels of processing.

Since we want $t(l_i) = t(l_j)$ (Property 1) or $t(l_i) \subset t(l_j)$ (Property 2), it follows that we should sort the itemsets in increasing order of their support. At the root level of the IT-Tree, CHARM uses a slightly different node ordering. Let $x, y \in \mathcal{I}$, define the weight of an item x as $w(x) = \sum_{xy \in F_2} \sigma(xy)$, i.e., the sum of the support of frequent 2-itemsets that contain the item x. At the root level, we sort the items in increasing order of their weights. For the remaining levels, elements are added in sorted order of support to each new class $[P_i]$ (lines 20 and 22). Thus, the reordering is applied recursively at each node in the tree.

Example. Fig. 7 shows how CHARM works on our example database. If we look at the support of 2-itemsets containing A, we find that AC and AW have support 4, while AT has support 3, thus, $w(A) = 4 + 4 + 3 = 11$. The final sorted order of items is then D, T, A, W, and C (their weights are 7, 10, 11, 15, and 17, respectively).

We initialize the root class as $[\emptyset] = \{D \times 2,456, T \times 1,356, A \times 1,345, W \times 12,345, C \times 123,456\}$ in Line 1. At Line 4 we first process the node $D \times 2456$ (we set $X = D$ in Line 5); it will be combined with the remaining elements in Line 6. DT and DA are not frequent and are pruned. We next look at D and W; since $t(D) \neq t(W)$, Property 4 applies and we simply insert W in $[D]$ (line 22). We next find that $t(D) \subset t(C)$. Since Property 2 applies, we replace all occurrences of D with DC, which means that we also change $[D]$ to $[DC]$ and the element DW to DWC. We next make a recursive call to CHARM-EXTEND with class $[DC]$. Since there is only one element, we jump to line 9, where DWC is added to the frequent closed set $\mathcal{C}$ after subsumption checking, i.e., checking if there exists a closed superset with the same support as DWC (see Section 4.1). When we return the D (now DC) branch is complete, thus DC itself is added to $\mathcal{C}$.

When we process T, we find that $t(T) \neq t(A)$, thus we insert A in the new class $[T]$ (Property 4). Next, we find that $t(T) \neq t(W)$ and we get $[T] = \{A, W\}$. When we find $t(T) \subset t(C)$, we update all occurrences of T with TC (by Property 2). We thus get the class $[TC] = \{A, W\}$. CHARM then makes a recursive call on Line 10 to process $[TC]$. We try to combine TAC with TWC to find $t(TAC) = t(TWC)$. Since Property 1 is satisfied, we replace TAC with $TACW$, deleting TWC at the same time. Since $TACW$ cannot be extended further, we insert it in $\mathcal{C}$ and, when we are done processing branch TC, it too is added to $\mathcal{C}$. All other branches satisfy Property 2 and no new recursion is made; the final $\mathcal{C}$ consists of the uncrossed IT-pairs shown in Fig. 7.

4.1 Fast Subsumption Checking

Let X_i and X_j be two itemsets, we say that an itemset X_i *subsumes* another itemset X_j if and only if $X_j \subset X_i$ and $\sigma(X_j) = \sigma(X_i)$. Recall that, before adding a set P_i to the current set of closed patterns $\mathcal{C}$, CHARM makes a check in Line 9 (see Fig. 6) to see if P_i is subsumed by some closed set in $\mathcal{C}$. In other words, it may happen that, after adding a closed set Y to $\mathcal{C}$, when we explore subsequent branches, we may generate another set X, which cannot be extended further, with $X \subseteq Y$ and with $\sigma(Y) = \sigma(X)$. In this case, X is a nonclosed set subsumed by Y and it should not be added to $\mathcal{C}$. Since $\mathcal{C}$ dynamically expands during enumeration of closed patterns, we need a very fast approach to perform such subsumption checks.

Clearly, we want to avoid comparing P_i with all existing elements in $\mathcal{C}$ for this would lead to a $O(|\mathcal{C}|^2)$ complexity. To quickly retrieve relevant closed sets, the obvious solution is to store $\mathcal{C}$ in a hash table. But, what hash function to use? Since we want to perform subset checking, we cannot hash on the itemset. We could use the support of the itemsets for the hash function. But, many unrelated itemsets may have the same support. Since CHARM uses IT-pairs throughout its search, it seems reasonable to use the information from the tidsets to help identify if P_i is subsumed. Note that, if $t(X_j) = t(X_i)$, then, obviously, $\sigma(X_j) = \sigma(X_i)$. Thus, to check if P_i is subsumed, we can check if $t(P_i) = t(C)$ for some $C \in \mathcal{C}$. This check can be performed in $O(1)$ time using a hash table. But, obviously, we cannot afford to store the

```
SUBSUMPTION-CHECK($\mathcal{C}, P$):
1.  $h(P) = \sum_{T \in t(P)} T$
2.  for all $Y \in$ HASHTABLE$[h(P)]$ do
3.      if $\sigma(Y) \neq \sigma(P)$ or $P \not\subseteq Y$
4.          $\mathcal{C} = \mathcal{C} \cup P$
```

Fig. 8. Fast subsumption check.

actual tidset with each closed set in $\mathcal{C}$; the space requirements would be prohibitive.

CHARM adopts a compromise solution, as shown in Fig. 8. It computes a hash function on the tidset and stores in the hash table a closed set along with its support (in our implementation, we used the C++ STL—standard template library—hash_multimap container for the hash table). Let $h(X_i)$ denote a suitable chosen hash function on the tidset $t(X_i)$. Before adding P_i to $\mathcal{C}$, we retrieve from the hash table all entries with the hash key $h(P_i)$. For each matching, closed set C is then checked if $\sigma(P_i) = \sigma(C)$. If yes, we next check if $P_i \subset C$. If yes, then P_i is subsumed and we do not add it to hash table $\mathcal{C}$.

What is a good hash function on a tidset? CHARM uses the sum of the tids in the tidset as the hash function, i.e., $h(P_i) = \sum_{T \in t(P_i)} T$ (note, this is not the same as support, which is the cardinality of $t(P_i)$). We tried several other variations and found there to be no performance difference. This hash function is likely to be as good as any other due to several reasons. First, by definition, a closed set is one that does not have a superset with the same support; it follows that it must have some tids that do not appear in any other closed set. Thus, the hash keys of different closed sets will tend to be different. Second, even if there are several closed sets with the same hash key, the support check we perform (i.e., if $\sigma(P_i) = \sigma(C)$) will eliminate many closed sets whose keys are the same, but they, in fact, have different supports. Third, this hash function is easy to compute and it can easily be used with the diffset format we introduce next.

4.2 Diffsets for Fast Frequency Computations

Given that we are manipulating itemset-tidset pairs, CHARM uses a *vertical* data format, where we maintain a disk-based tidset for each item in the database. Mining algorithms using the vertical format have been shown to be very effective and usually outperform horizontal approaches [8], [18], [19], [22]. The main benefits of using a vertical format are: 1) Computing the supports is simpler and faster. Only intersections on tidsets are required, which are also well-supported by current databases. The horizontal approach, on the other hand, requires complex hash trees. 2) There is automatic pruning of irrelevant information as the intersections proceed; only tids relevant for frequency determination remain after each intersection. For databases with long transactions, it has been shown, using a simple cost model, that the vertical approach reduces the number of I/O operations [8]. Further, vertical bitmaps offer scope for compression [19].

Despite the many advantages of the vertical format, when the tidset cardinality gets very large (e.g., for very frequent items), the methods start to suffer since the intersection time starts to become inordinately large. Furthermore, the size of intermediate tidsets generated for frequent patterns can also become very large, requiring data compression and writing of temporary results to disk. Thus, (especially) in dense data sets, which are characterized by high item frequency and many patterns, the vertical approaches may quickly lose their advantages. In this paper, we utilize a vertical data representation, called *diffsets*, that we recently proposed [23]. Diffsets keep track of differences in the tids of a candidate pattern from its parent frequent pattern. These differences are propagated all the way from one node to its children starting from the root. We showed in [23] that diffsets drastically cut down (by orders of magnitude) the size of memory required to store intermediate results. Thus, even in dense domains, the entire working set of patterns of several vertical mining algorithms can fit entirely in main-memory. Since the diffsets are a small fraction of the size of tidsets, intersection operations are performed very efficiently.

More formally, consider a class with prefix P. Let $d(X)$ denote the diffset of X, with respect to a prefix tidset, which is the current universe of tids. In normal vertical methods, one has available for a given class the tidset for the prefix $t(P)$ as well as the tidsets of all class members $t(PX_i)$. Assume that PX and PY are any two class members of P. By the definition of support, it is true that $t(PX) \subseteq t(P)$ and $t(PY) \subseteq t(P)$. Furthermore, one obtains the support of PXY by checking the cardinality of

$$t(PX) \cap t(PY) = t(PXY).$$

Now, suppose instead that we have available to us not $t(PX)$ but rather $d(PX)$, which is given as $t(P) - t(X)$, i.e., the differences in the tids of X from P. Similarly, we have available $d(PY)$. The first thing to note is that the support of an itemset is no longer the cardinality of the diffset, but rather it must be stored separately and is given as follows: $\sigma(PX) = \sigma(P) - |d(PX)|$. So, given $d(PX)$ and $d(PY)$, how can we compute if PXY is frequent? We use the diffsets recursively as we mentioned above, i.e.,

$$\sigma(PXY) = \sigma(PX) - |d(PXY)|.$$

So, we have to compute $d(PXY)$. By our definition, $d(PXY) = t(PX) - t(PY)$. But, we only have diffsets and not tidsets as the expression requires. This is easy to fix since

$$\begin{aligned} d(PXY) &= t(PX) - t(PY) = t(PX) - t(PY) + t(P) - t(P) \\ &= (t(P) - t(PY)) - (t(P) - t(PX)) \\ &= d(PY) - d(PX). \end{aligned}$$

In other words, instead of computing $d(XY)$ as a difference of tidsets $t(PX) - t(PY)$, we compute it as the difference of the diffsets $d(PY) - d(PX)$. Fig. 9 shows the different regions for the tidsets and diffsets of a given prefix class and any two of its members. The tidset of P, the triangle marked $t(P)$, is the universe of relevant tids. The gray region denotes $d(PX)$, while the region with the solid black line denotes $d(PY)$. Note also that both $t(PXY)$ and $d(PXY)$ are subsets of the tidset of the new prefix PX. Diffsets are typically much smaller than storing the tidsets with each child since only the essential changes are propagated from a node to its children. Diffsets also shrink as longer itemsets are found.

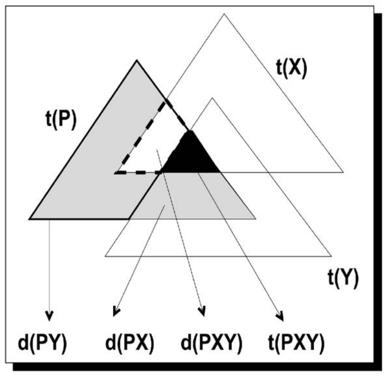


Fig. 9. Diffsets: prefix P.

Diffsets and subsumption checking. Notice that diffsets cannot be used directly for generating a hash key as was possible with tidsets. The reason is that, depending on the class prefix, nodes in different branches will have different diffsets, even though one is subsumed by the other. The solution is to keep track of the hash key $h(PXY)$ for PXY in the same way as we store $\sigma(PXY)$. In other words, assume that we have available $h(PX)$, then we can compute $h(PXY) = h(PX) - \sum_{T \in d(PXY)} T$. Of course, this is only possible because of our choice of hash function described in Section 4.1. Thus, we associate with each member of a class its hash key and the subsumption checking proceeds exactly as for tidsets.

Differences and subset testing. We assume that the initial database is stored in tidset format, but we use diffsets thereafter. Given the availability of diffsets for each itemset, the computation of the difference for a new combination is straightforward. All it takes is a linear scan through the two diffsets, storing tids in one but not the other. The main question is how to efficiently compute the subset information, while computing differences, required for applying the four IT-pair properties. At first, this might appear like an expensive operation, but, in fact, it comes for free as an outcome of the set difference operation. While taking the difference of two sets, we keep track of the number of mismatches in both the diffsets, i.e., the cases when a tid occurs in one list but not in the other. Let $m(X_i)$ and $m(X_j)$ denote the number of mismatches in the diffsets $d(X_i)$ and $d(X_j)$. There are four cases to consider:

1. Property 1. $m(X_i) = 0$ and $m(X_j) = 0$, then $d(X_i) = d(X_j)$ or $t(X_i) = t(X_j)$.
2. Property 2. $m(X_i) > 0$ and $m(X_j) = 0$, then $d(X_i) \supset d(X_j)$ or $t(X_i) \subset t(X_j)$.
3. Property 3. $m(X_i) = 0$ and $m(X_j) > 0$, then $d(X_i) \subset d(X_j)$ or $t(X_i) \supset t(X_j)$.
4. Property 4. $m(X_i) > 0$ and $m(X_j) > 0$, then $d(X_i) \neq d(X_j)$ or $t(X_i) \neq t(X_j)$.

Thus, CHARM performs support, subset, equality, and inequality testing simultaneously while computing the difference itself. Fig. 10 shows the search for closed sets using diffsets instead of tidsets. The exploration proceeds in exactly the same way as described in Example 1. However, this time we perform difference operations on diffsets (except for the root class, which uses tidsets). Consider an IT-pair like $TAWC \times 6$. Since this indicates that $TAWC$ differs from its parent $TC \times 1356$ only in the tid 6, we can infer that the real IT-pair should be $TAWC \times 135$.

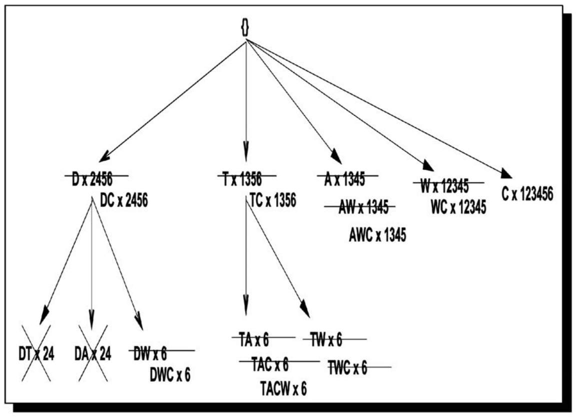


Fig. 10. Search process using diffsets.

4.3 Other Optimizations and Correctness

Optimized initialization. There is only one significant departure from the pseudocode in Fig. 6. Note that if we initialize the $[P]$ set in Line 1 with all frequent items and invoke CHARM-EXTEND, then, in the worst case, we might perform $n(n-1)/2$ difference operations, where n is the number of frequent items. It is well known that many itemsets of length 2 turn out to be infrequent, thus it is clearly wasteful to perform $O(n^2)$ operations. To solve this performance problem, we first compute the set of frequent itemsets of length 2 and then we add a simple check in Line 6 (not shown for clarity; it only applies to 2-itemsets) so that we combine two items X_i and X_j only if $X_i \cup X_j$ is known to be frequent. The number of operations performed after this check is equal to the number of frequent pairs, which in practice is closer to $O(n)$ rather than $O(n^2)$. To compute the frequent itemsets of length 2 using the vertical format, we perform a multistage vertical-to-horizontal transformation on-the-fly, as described in [22], over distinct ranges of tids. Given a recovered horizontal database chunk, it is straightforward to update the count of pairs of items using an upper triangular 2D array. We then process the next chunk. The horizontal chunks are thus temporarily materialized in memory and then discarded after processing [22].

Memory management. Since CHARM processes branches in a depth-first fashion, its memory requirements are not substantial. It has to retain all the itemset-diffsets pairs on the levels of the current left most branches in the search space. The use of diffsets also drastically reduces the memory consumption. For cases where even the memory requirement of depth-first search and diffsets exceed available memory, it is straightforward to modify CHARM to write/read temporary diffsets to/from disk, as in [19].

Theorem 2 (correctness). *CHARM enumerates all frequent closed itemsets.*

Proof. CHARM correctly identifies all and only the closed frequent itemsets since its search is based on a complete IT-tree search space. The only branches that are pruned are those that either do not have sufficient support or

```
CHARM-L ($\mathcal{D}$, $min\_sup$):
1.  $[\emptyset] = \{l_i \times t(l_i) : l_i \in \mathcal{I} \wedge \sigma(l_i) \geq min\_sup\}$
2.  CHARM-L-EXTEND ($[\emptyset]$, $\mathcal{L}_r = \{\emptyset\}$)
3.  return $\mathcal{L}$ //lattice of closed sets

CHARM-L-EXTEND ($[P]$, $\mathcal{L}_c$):
4.  for each $l_i \times t(l_i), \mathbb{C}(l_i)$ in $[P]$
5.      $P_i = P \cup l_i$ and $[P_i] = \emptyset$
6.      UPDATE-$\mathbb{C}$ ($l_i$, $[P]$)
7.      for each $l_j \times t(l_j), \mathbb{C}(l_j)$ in $[P]$, with $j > i$
8.          $X = l_j$ and $Y = t(l_i) \cap t(l_j)$ and $\mathbb{C}(X) = \mathbb{C}(l_i) \cap \mathbb{C}(l_j)$
9.          CHARM-PROPERTY($X \times Y, l_i, l_j, P_i, [P_i], [P]$)
10.     $\mathcal{L}_n$ = SUBSUMPTION-CHECK-LATTICE-GEN($\mathcal{L}_c, P_i, \mathbb{C}(P_i)$)
11.     CHARM-L-EXTEND ($[P_i]$, $\mathcal{L}_n$)
12.     delete $[P_i]$
```

Fig. 11. The CHARM-L Algorithm.

those that are subsumed by another closed set based on the properties of itemset-tidset pairs as outlined in Theorem 1. Finally, CHARM eliminates any nonclosed itemset that might be generated by performing subsumption checking before inserting anything in the set of all frequent closed itemsets $\mathcal{C}$. □

5 CHARM-L: Generating Closed Itemset Lattice

Consider the closed itemset lattice for our example database, shown in Fig. 2. All of the current closed set mining algorithms such as Close [16], Pascal [3], Closet [17], Closet+ [20], Mafia [6], as well as CHARM, do not output the lattice explicitly. Their output is simply a list of all the closed sets found. On the other hand, for efficient rule generation from the mined patterns, it is essential to know the lattice, i.e., the subset-superset relationship between the closed sets [26]. One approach to generate the lattice is to first mine the closed sets, $\mathcal{C}$, and to then construct the lattice for $\mathcal{C}$. Unfortunately, lattice construction has time complexity $O(|\mathcal{C}|^2)$ [14], which is too slow for a large number of closed itemsets, as we shall see in the experimental section.

We therefore decided to extend CHARM to directly compute the lattice while it generates the closed itemsets. The basic idea is that, when a new closed set X is found, we efficiently determine all its possible closed supersets, $\mathcal{S} = \{Y | Y \in \mathcal{C} \wedge X \subset Y\}$. The minimal elements in $\mathcal{S}$ form the "immediate" supersets or children of X in the closed itemset lattice. This approach leads to a very efficient algorithm, which we call CHARM-L.

Fig. 11 gives the pseudocode for CHARM-L. Let $\mathcal{L}$ denote the closed itemset lattice and $\mathcal{L}_r$ the root node of the lattice; we assume that $\mathcal{L}_r = \emptyset$. CHARM-L starts in the same manner as CHARM by initializing the parent class with the frequent items. It then makes a call to the extension subroutine, passing it the parent equivalence class and the lattice root as the current lattice node.

CHARM-L-EXTEND takes as input the current lattice node $\mathcal{L}_c$ (initially, the root node) and an equivalence class of IT-pairs. Whenever CHARM-L generates a new closed itemset, it assigns it a unique closed itemset identifier, called *cid*. In CHARM-L, each element $l_i \times t(l_i) \in [P]$ has

```
SUBSUMPTION-CHECK-LATTICE-GEN($\mathcal{L}_c, X, \mathbb{C}(X)$):
1.  $\mathcal{S} = \{Z \in \mathcal{C} | Z.\text{cid} \in \mathbb{C}(X)\}$
    //eliminate subsumed itemsets
2.  for each $Z \in \mathcal{S}$ do
3.      if $\sigma(X) = \sigma(Z)$ then return $\mathcal{L}_c$.
    //Insert $X$ as child of $\mathcal{L}_c$
4.  $\mathcal{L}_n = X$
5.  $\mathcal{L}_c$.children.add($\mathcal{L}_n$), $\mathcal{L}_n$.parents.add($\mathcal{L}_c$)
    //Adjust Lattice
6.  $\mathcal{S}^{\min} = \{Z \in \mathcal{S} | Z \text{ is Minimal}\}$
7.  for all $Z \in \mathcal{S}^{\min}$ do
8.      $\mathcal{L}_n$.children.add($Z$), $Z$.parents.add($\mathcal{L}_n$)
9.      for all $Z_p \in Z$.parents do
10.         if $Z_p \subset \mathcal{L}_n$ then
11.             $Z_p$.children.remove($Z$), $Z$.parents.remove($Z_p$)
12. return $\mathcal{L}_n$
```

Fig. 12. CHARM-L: Subsumption checking and lattice growth.

associated with it a *cidset*, denoted $\mathbb{C}$, which is the set of all cids of closed itemsets that are supersets of Pl_i. Given $\mathbb{C}(l_i)$ and $\mathbb{C}(l_j)$, one can obtain the set of closed itemsets that contain both Pl_i and Pl_j by simply intersecting the two cidsets, i.e., $\mathbb{C}(X) = \mathbb{C}(l_i) \cap \mathbb{C}(l_j)$, as done in Line 8. CHARM-L enumerates all closed sets which are not subsumed (Line 10), but, in addition, it also generates a new lattice node $\mathcal{L}_n$ for the new closed set and inserts it in the appropriate place in the closed itemset lattice $\mathcal{L}$. This new lattice node $\mathcal{L}_n$ becomes the current node in the next recursive call of the extension subroutine (Line 11). Since the list of closed supersets of l_i may change whenever a new closed itemset is added to the lattice, a check is made in Line 6 to update $\mathbb{C}(l_i)$ for each remaining element in the class. Note that CHARM-L shares the optimizations for computing length 2 itemsets mentioned in Section 4.3.

Subsumption check and lattice generation. To check if a new itemset P_i is closed (Fig. 11, Line 10), we apply SUBSUMPTION-CHECK-LATTICE-GEN, shown in Fig. 12. This routine takes as input the current lattice node, the new itemset X, and the cidset $\mathbb{C}(X)$. The first task is to check if X is subsumed. For this, we consider all closed itemsets $\mathcal{S}$ that are supersets of X (Line 1). If X has the same support as any superset $Z \in \mathcal{S}$ (Line 2), then X is subsumed and we return (Line 3). Otherwise, the new lattice node is initialized as $\mathcal{L}_n = X$ (Line 4). Each node in the lattice maintains a list of parents (immediate subsets) and children (immediate supersets). We add the new node $\mathcal{L}_n$ as a child of the current node $\mathcal{L}_c$ and $\mathcal{L}_c$ as the parent of $\mathcal{L}_n$ (Line 5). Out of all the closed supersets of $\mathcal{L}_n$ (i.e., X), the minimal supersets are found $\mathcal{S}^{\min}$ (Line 6). Each minimal superset $Z \in \mathcal{S}^{\min}$ becomes a child of $\mathcal{L}_n$ (and $\mathcal{L}_n$ a parent of Z) (Line 8). Finally, for every parent Z_p of Z, if $Z_p \subset \mathcal{L}_n$, then its children pointers have to be adjusted; we remove Z from Z_p's children (and Z_p from Z's parents) (Lines 9-10). Finally, we return the new lattice node $\mathcal{L}_n$ (Line 12).

Note that CHARM-L differs from CHARM in the way it does subsumption checking. In CHARM, we use the fast hash-based subsumption check. The hash-based approach hashes on the sum of tids for an itemset, locates closed sets having the same support, and then we

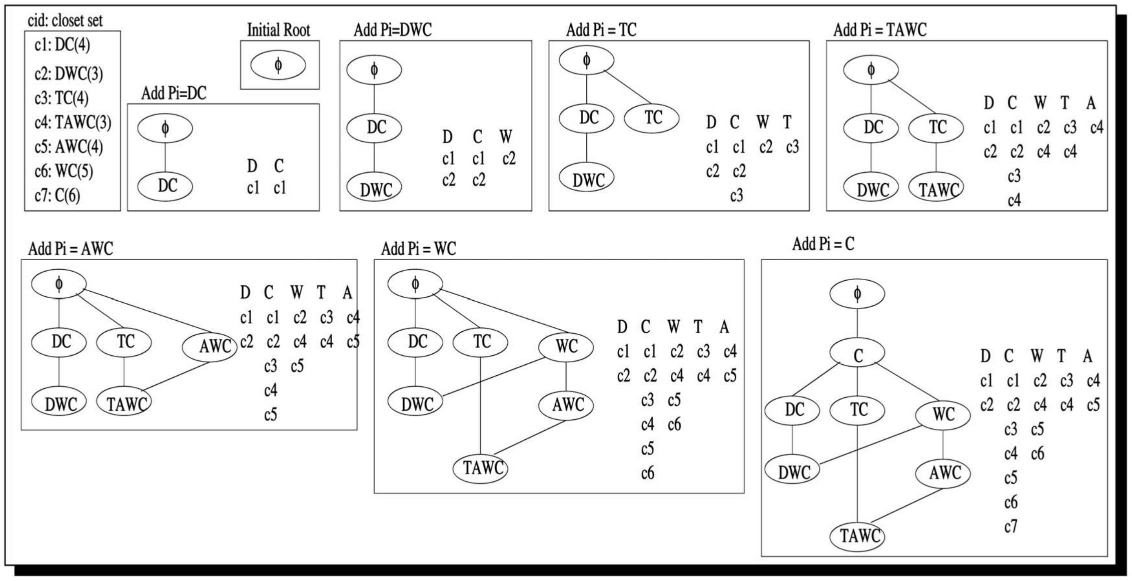


Fig. 13. CHARM-L: lattice growth and cidsets.

can use subset tests to check subsumption. On the other hand, for lattice generation, we need to know all the closed supersets (and the minimal elements among them) for a new closed set, and, since all of these will, by definition, have different supports, they will be in different hash cells. Thus, the hash-based approach is not suitable for lattice generation. CHARM-L thus uses the cidset intersection based subsumption checking, which is also used for lattice generation.

Updating $\mathbb{C}$. Consider the UPDATE-$\mathbb{C}$ routine in CHARM-L (Fig. 11, Line 6). After the recursive call to CHARM-L-EXTEND (Fig. 11, Line 11), new closed sets may have been generated, so we need to update the cidsets for all remaining IT-pairs in class $[P]$. That is, for all IT-pairs $l_j \times t(l_j) \in [P]$, with $l_j \geq_f l_i$, UPDATE-$\mathbb{C}$ adds the cids of all newly generated closed sets to $\mathbb{C}(l_j)$.

Example. Fig. 13 shows how CHARM-L works; we only consider the SUBSUMPTION-CHECK-LATTICE-GEN routine since the rest of the algorithm is similar to CHARM. As each closed set is found, it is assigned a new cid and inserted into a lookup table, as shown on the left most box. Initially, the lattice only has the root $\mathcal{L}_r = \emptyset$ and the cidsets of all items are empty. The first closed itemset to be added is $P_i = DC$, with the current node as $\mathcal{L}_c = \mathcal{L}_r$, so we add $\mathcal{L}_n = DC$ as a new child and update the cidsets of C and D, as shown (with $\mathbb{C}(C) = \{c_1\}, \mathbb{C}(D) = \{c_1\}$). Next, we add $P_i = DWC$, with $\mathcal{L}_c = DC$. DWC is added as the new node $\mathcal{L}_n$ and becomes a child of DC; the cidsets are also updated. The process continues and each new node becomes a child of the current node.

One interesting case is when we add $P_i = WC$, which requires adjusting the parent pointers of existing lattice nodes. We have

$$\mathbb{C}(WC) = \mathbb{C}(C) \cap \mathbb{C}(W) = \{c_1, c_2, c_3, c_4, c_5\} \cap \{c_2, c_4, c_5\} = \{c_2, c_4, c_5\}.$$

Note that the figure shows the cidsets only for the single items, but CHARM-L actually computes the cidsets of all itemsets via intersections (Fig. 11, Line 8). Thus, $\mathcal{S} = \{DWC, TAWC, AWC\}$. We find that WC is not subsumed since $\sigma(WC) = 5$ does not match the support of any closed superset. We then add $\mathcal{L}_n = WC$ as a child of $\mathcal{L}_c = \mathcal{L}_r$ and we compute the minimal elements, $\mathcal{S}^{\min} = \{DWC, AWC\}$. Each of these minimal elements becomes a child of WC. Next, we check if any parent of a set in $\mathcal{S}^{\min}$ is a subset of WC, in which case, we need to adjust the lattice. We do not have such a case with DWC since its only parent is DC (before adding WC), which is not a subset of WC. However, $\mathcal{L}_r = \emptyset$, the parent of AWC (before adding WC), is a subset of WC, so we remove the parent-child link between $\mathcal{L}_r$ and AWC. Finally, the last closed set to be added is C and we obtain the full closed itemset lattice as shown (bottom, right most box).

6 Experimental Evaluation

Experiments were performed on a 400MHz Pentium PC with 256MB of memory, running RedHat Linux 6.0. Algorithms were coded in C++. For performance comparison, we used the original source or object code for Close [16], Pascal [3], Closet [17], and Mafia [6], all provided to us by their authors. The original Closet code had a subtle bug, which affected the performance, but not the correctness. Our comparison below uses the new bug-free, optimized version of Closet obtained from its authors. Mafia has an option to mine only closed sets instead of maximal sets. We refer to this version of Mafia below. We also include a comparison with the base Apriori algorithm [1] for mining all itemsets. Timings in the figures below are based on total wall-clock time and include all preprocessing costs (such as vertical database creation in CHARM and Mafia).

Benchmark data sets. We chose several real and synthetic database benchmarks [1], [4], publicly available from IBM Almaden (www.almaden.ibm.com/cs/quest/demos.html), for the performance tests. The PUMS data

TABLE 1
Database Characteristics

Database	# Items	Avg. Length	Std. Dev.	# Records	Max. Pattern (sup)	Levels Searched
chess	76	37	0	3,196	23 (20%)	17
connect	130	43	0	67,557	29 (10%)	12
mushroom	120	23	0	8,124	21 (0.075%)	11
pumsb*	7117	50	2	49,046	40 (5%)	16
pumsb	7117	74	0	49,046	22 (60%)	17
gazelle	498	2.5	4.9	59,601	154 (0.01%)	11
T10I4D100K	1000	10	3.7	100,000	11 (0.025%)	11
T40I10D100K	1000	40	8.5	100,000	25 (0.001%)	19

sets (pumsb and pumsb*) contain census data. pumsb* is the same as pumsb without items with 80 percent or more support. The mushroom database contains characteristics of various species of mushrooms. The connect and chess data sets are derived from their respective game steps. The latter three data sets were originally taken from the UC Irvine Machine Learning Database Repository. The synthetic data sets (T10 and T40), using the IBM generator, mimic the transactions in a retailing environment.

The gazelle data set comes from click-stream data from a small dot-com company called Gazelle.com, a legware and legcare retailer, which no longer exists. A portion of this data set was used in the KDD-Cup 2000 competition. This data set was recently made publicly available by Blue Martini Software (download it from www.ecn.purdue.edu/KDDCUP).

Typically, the real data sets are very dense, i.e., they produce many long frequent itemsets even for very high values of support. The synthetic data sets mimic the transactions in a retailing environment. Usually, the synthetic data sets are sparser when compared to the real sets.

Table 1 shows the characteristics of the real and synthetic data sets used in our evaluation. It shows the number of items, the average transaction length, the standard deviation of transaction lengths, and the number of records in each database. The table additionally shows the length of the longest maximal pattern (at the lowest minimum support used in our experiments) for the different data sets, as well as the maximum level of search that CHARM performed to discover the longest pattern. For example, on gazelle, the longest closed pattern was of length 154 (any method that mines all frequent patterns will be impractical for such long patterns), yet the maximum recursion depth in CHARM was only 11! The number of levels skipped is also considerable for other real data sets. The synthetic data set $T10$ is extremely sparse and no levels are skipped, but for $T40$ six levels were skipped. These results give an indication of the effectiveness of CHARM in mining closed patterns, and are mainly due to repeated applications of Properties 1 and 2 in Theorem 1.

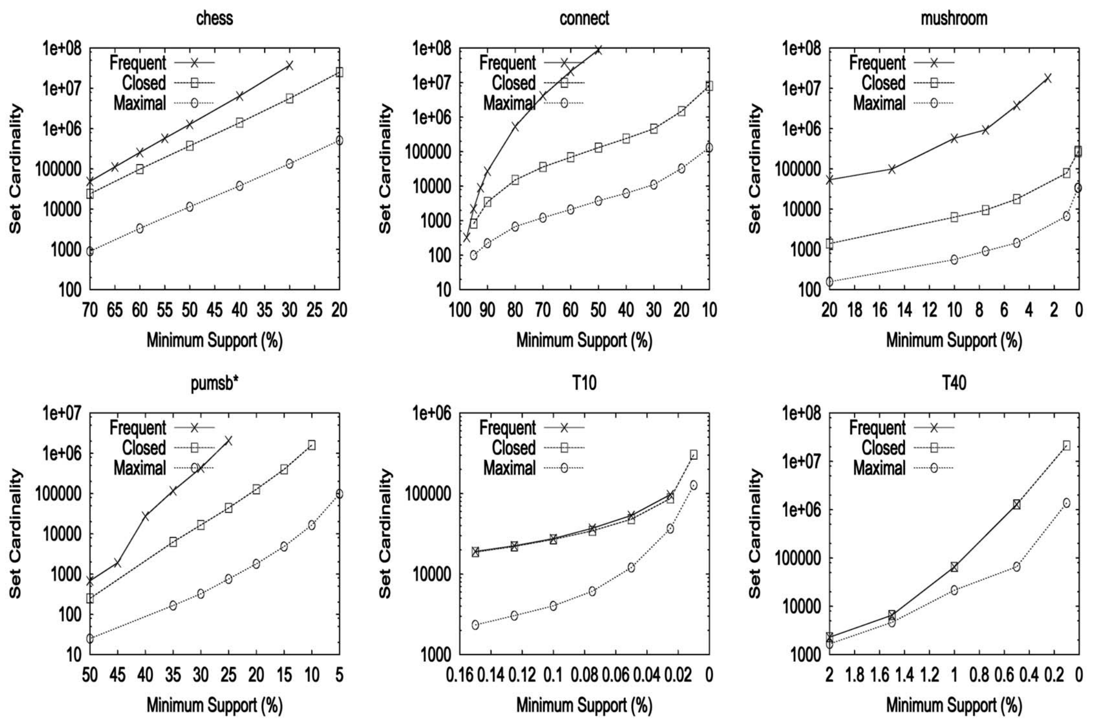


Fig. 14. Cardinality of frequent, closed, and maximal itemsets.

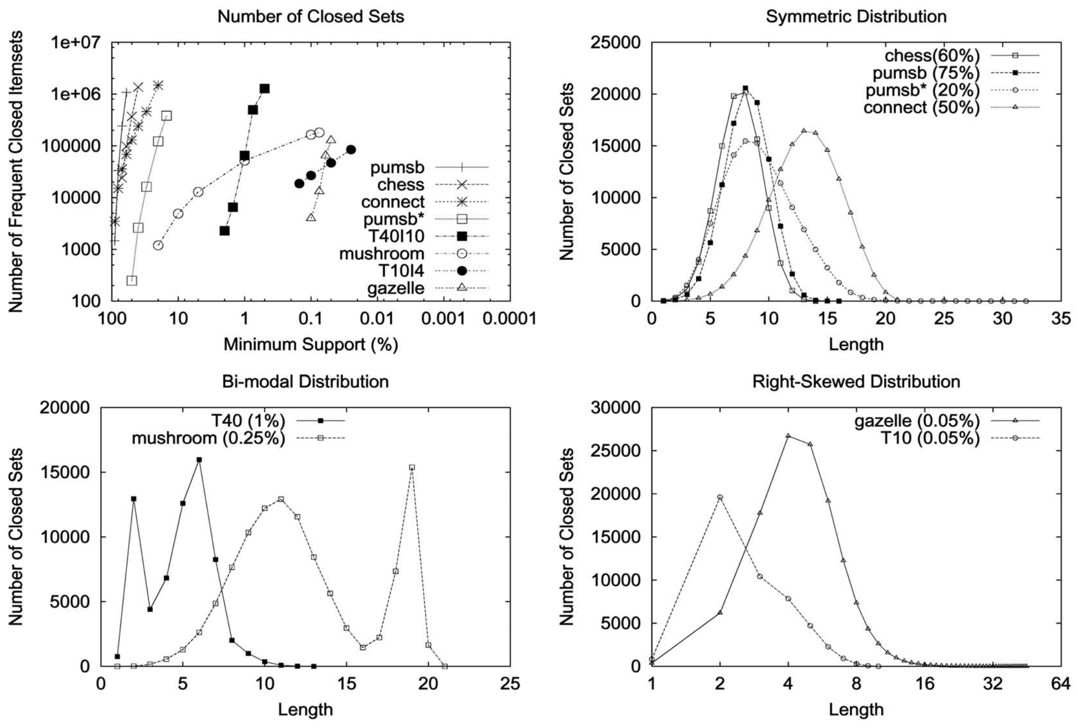


Fig. 15. Number of frequent closed itemsets and distribution by length.

Fig. 14 shows the total number of frequent, closed, and maximal itemsets found for various support values. The maximal frequent itemsets are a subset of the frequent closed itemsets (the maximal frequent itemsets must be closed, since, by definition, they cannot be extended by another item to yield a frequent itemset). The frequent closed itemsets are, of course, a subset of all frequent itemsets. Depending on the support value used, for the real data sets, the set of maximal itemsets is about an order of magnitude smaller than the set of closed itemsets, which, in turn, is an order of magnitude smaller than the set of all frequent itemsets. Even for very low support values we find that the difference between maximal and closed remains around a factor of 10. However, the gap between closed and all frequent itemsets grows more rapidly. On the other hand, in sparse data sets, the number of closed sets is only marginally smaller than the number of frequent sets; the number of maximal sets is still smaller, though the differences can narrow down for low support values.

Before we discuss the performance results of different algorithms, it is instructive to look at the total number of frequent closed itemsets and distribution of closed patterns by length for the various data sets, as shown in Fig. 15. We have grouped the data sets according to the type of distribution. chess, pumsb*, pumsb, and connect all display an almost symmetric distribution of the closed frequent patterns with different means. $T40$ and mushroom display an interesting bimodal distribution of closed sets. $T40$, like $T10$, has a many short patterns of length 2, but it also has another peak at length 6. mushroom has considerably longer patterns; its second peak occurs at 19. Finally, gazelle and T10 have a right-skewed distribution. gazelle tends to have many small patterns, with a very long right tail. $T10$ exhibits a similar distribution, with the majority of the closed patterns begin of length 2! The type of distribution tends to influence the behavior of different algorithms, as we will see below.

6.1 Performance Testing

We compare the performance of CHARM against Apriori, Close, Pascal, Mafia, Closet, and Closet+ in Fig. 16, Fig. 17, and Fig. 18. Since Closet and Closet+ were provided as a Windows executable by its authors, we compared them separately on a 3.06 MHz Pentium 4 processor with 1GB memory, running Windows XP. In [24], we tested a modified version of a maximal pattern finding algorithm (MaxMiner [4]) to discover closed sets in a postprocessing step and found it to be too slow for all except short patterns.

Symmetric data sets. Let us first compare how the methods perform on data sets which exhibit a symmetric distribution of closed itemsets, namely, chess, pumsb, connect, and pumsb*, as shown in Fig. 16. We observe that Apriori, Close, and Pascal work only for very high values of support on these data sets. The best among the three is Pascal, which can be twice as fast as Close and up to 4 times better than Apriori. On the other hand, CHARM is several orders of magnitude better than Pascal and it can be run on very low support values, where none of the former three methods can be run. Comparing with Mafia, we find that both CHARM and Mafia have a similar performance for higher support values. However, as we lower the minimum support, the performance gap between CHARM and Mafia widens. For example, at the lowest support value plotted, CHARM is about 30 times faster than Mafia on Chess, about three times faster on connect and pumsb, and four times faster on pumsb*. CHARM outperforms both Closet and Closet+ by an order of magnitude or more, especially as support is lowered (except for connect). Closet is always

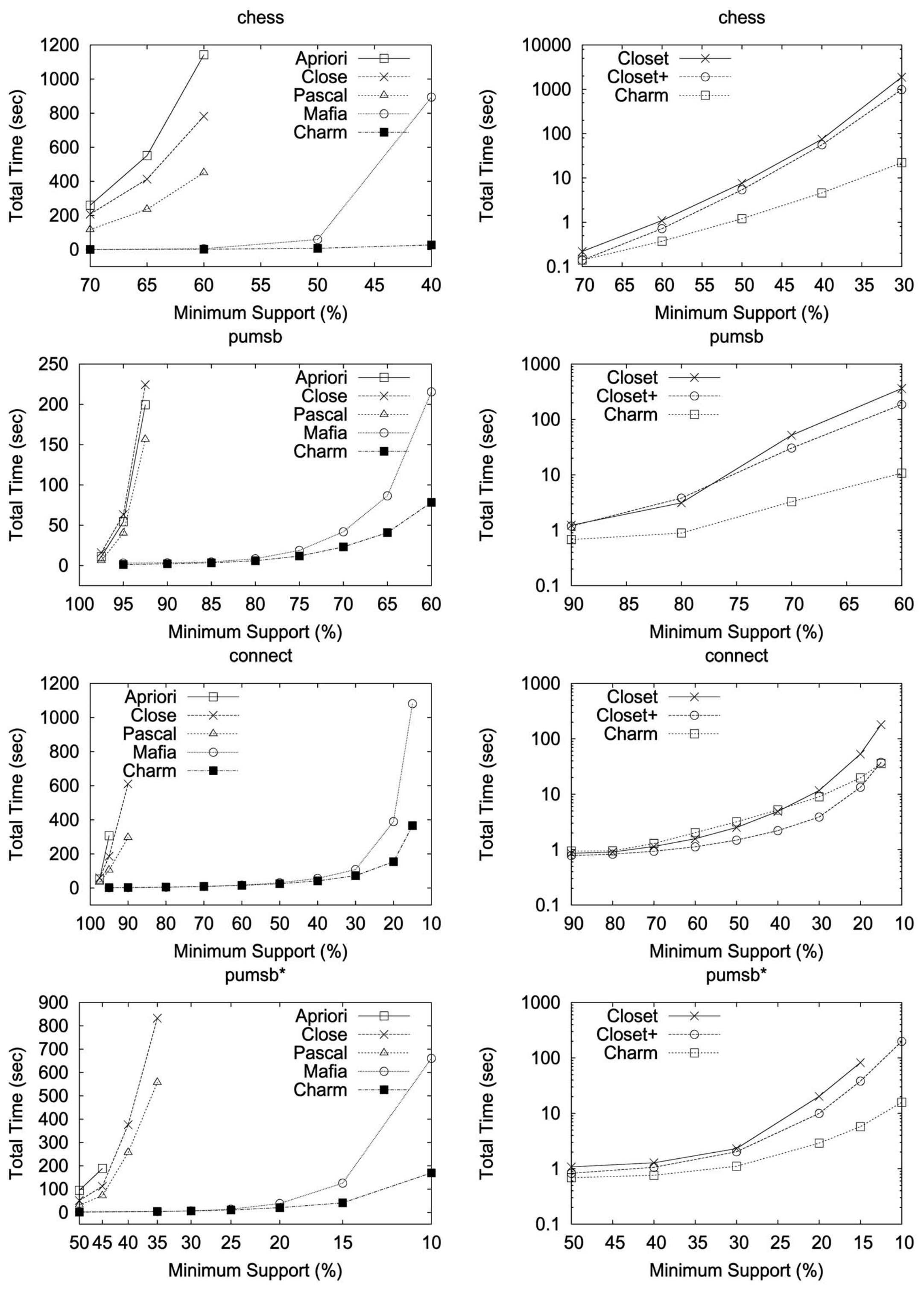


Fig. 16. Performance of CHARM on symmetric data sets.

slower than Closet+. On chess, CHARM is 30 times faster than Closet+ and, on pumsb and pumsb*, it is over 10 times faster than Closet+. On connect, Closet performs better, but both seem to become comparable at low support. The reason is that connect has transactions with a lot of overlap among items, leading to a compact FP-tree and to faster performance.

Bimodal Data Sets. On the two data sets with a bimodal distribution of frequent closed patterns, namely, mushroom and $T40$ (as shown in Fig. 17), we find that Pascal fares better than for symmetric distributions. For higher values of support, the maximum closed pattern length is relatively short and the distribution is dominated by the first mode. Apriori, Close, and Pascal can handle this case. However,

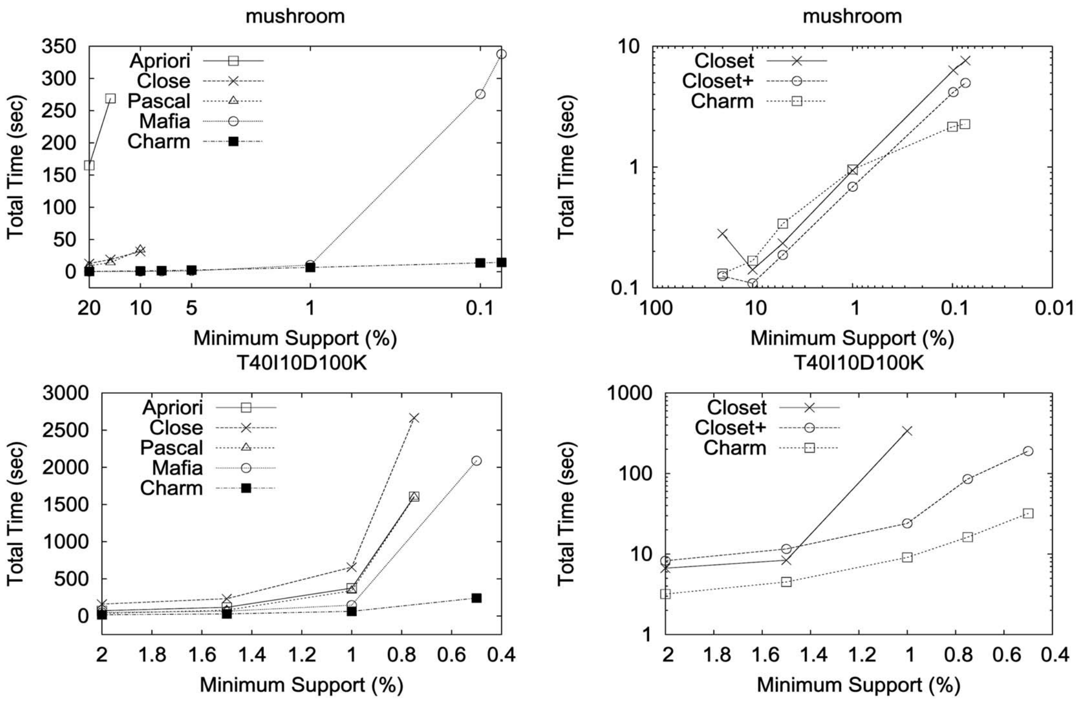


Fig. 17. Performance of CHARM on bimodal data sets.

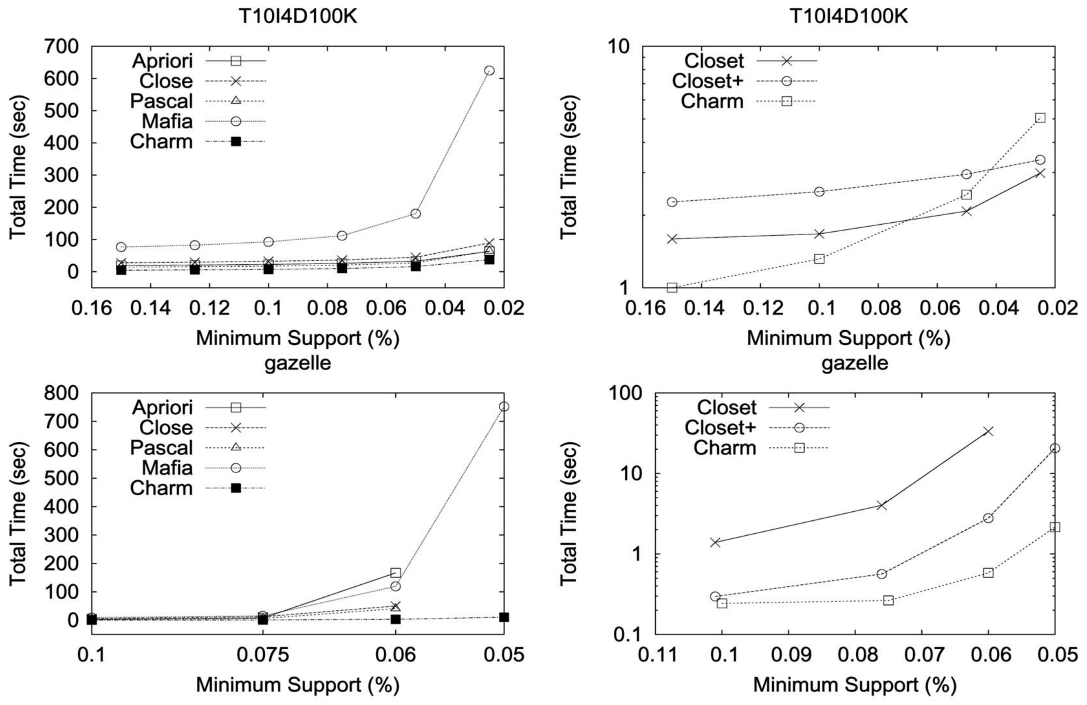


Fig. 18. Performance of CHARM on right-skewed data sets.

as one lowers the minimum support, the second mode starts to dominate, with longer patterns. These methods thus quickly lose steam and become uncompetitive. Between CHARM and Mafia, up to 1 percent minimum support, there is negligible difference, however, when the support is lowered, there is a huge difference in performance. CHARM is about 20 times faster on mushroom and 10 times faster on $T40$ for the lowest support shown. The gap continues to widen sharply. Closet is slower than Closet+, for all except very high supports. We find that CHARM outperforms Closet+ by a factor of 2 for mushroom and 5 for $T40$.

Right-skewed data sets. On gazelle and $T10$, which have a large number of very short closed patterns, followed by a sharp drop (as shown in Fig. 18), we find that Apriori, Close, and Pascal remain competitive even for relatively low supports. The reason is that $T10$ had a maximum pattern length of 11 at the lowest support shown. Also,

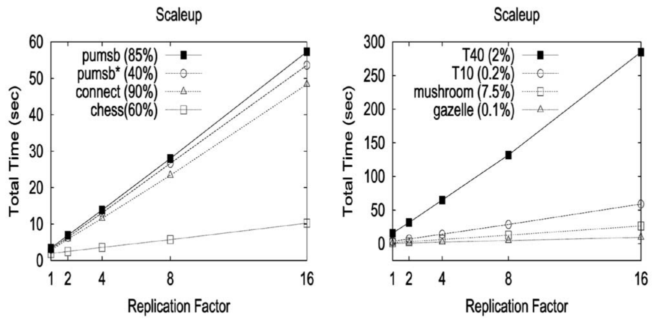


Fig. 19. Size scaleup on different data sets.

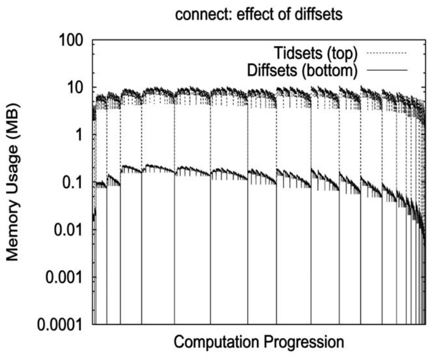


Fig. 20. Memory Usage (80 percent minsup).

gazelle at 0.06 percent support also had a maximum pattern length of 11. The level-wise search of these three methods is able to easily handle such short patterns. However, for gazelle, we found that, at 0.05 percent support, the maximum pattern length suddenly jumped to 45 and none of these three methods could be run.

$T10$, though a sparse data set, is problematic for Mafia. The reason is that $T10$ produces long sparse bitvectors for each item and offers little scope for bit-vector compression and projection that Mafia relies on for efficiency. This causes Mafia to be uncompetitive for such data sets. Similarly, Mafia fails to do well on gazelle. However, it is able to run on the lowest support value. The diffset format of CHARM is resilient to sparsity (as shown in [23]) and it continues to outperform other methods. For the lowest support, on $T10$, it is twice as fast as Pascal and 15 times better than Mafia and it is about 70 times faster than Mafia on gazelle. CHARM is about two times slower than Closet/Closet+ on $T10$. The reason is that the majority of closed sets are of length 2 and the tidset/diffsets operations in CHARM are relatively expensive compared to the compact FP-tree for short patterns (max length is only 11). However, for gazelle, which has much longer closed patterns, CHARM outperforms Closet+ by a factor of 5 and Closet by a factor of 40!

Scaleup experiments. Fig. 19 shows how CHARM scales with an increasing number of transactions. For this study, we kept all database parameters constant and replicated the transactions from 2 to 16 times. Thus, for example, for $T40$, which has 100K transactions initially, at a replication factor of 16, it will have 1.6 million transactions. At a given level of support, we find a linear increase in the running time with increasing number of transactions.

Memory usage. Fig. 20 shows how the memory usage for storing the tidsets and diffsets changes as computation progresses. The total usage for tidsets is generally under 10MB, but, for diffsets, it is under 0.2MB, a reduction by a factor of 50! The sharp drop to 0 (the vertical lines) indicates the beginning of a new prefix class. Table 2 shows the maximum memory usage for three data sets for different values of support. Here, we also see that the memory footprint using diffsets is extremely small even for low values of support. These results confirm that, for many data sets, the intermediate diffsets can easily fit in memory. However, as observed in [20], while CHARM outperforms Closet+ on most data sets, its memory consumption could still be high; this is mainly due to the tidsets of 2-itemsets, which do not benefit from diffsets.

6.2 CHARM-L Performance

Fig. 21 shows the performance of CHARM-L on different data sets for various support values. We compare its performance with an approach that first mines the closed sets and then constructs the lattice in a postprocessing step, labeled as POST-LAT in the figure. We also compare CHARM-L with CHARM to see how much more expensive lattice generation is versus just listing all closed itemsets. We find that CHARM-L is over 100 times faster than Post-Lat over the support values tested (except for very high support values). It is clear that this difference will only increase as support is lowered and more closed itemsets are found. We also find that CHARM-L compares favorably with CHARM, but the extra overhead in generating the lattice makes it slower than CHARM (the gap will widen for lower support values).

7 Conclusions

We presented and evaluated CHARM, an efficient algorithm for mining closed frequent itemsets, and CHARM-L, an efficient algorithm to generate the closed itemset lattice. These algorithms simultaneously explore both the itemset space and tidset space using the new IT-tree framework, which allows a novel search method that skips many levels to quickly identify the closed frequent itemsets, instead of having to enumerate many nonclosed subsets. We utilized a

TABLE 2
Maximum Memory Usage (Using Diffsets)

DB	50%	20%	DB	0.1%	0.05%	DB	1%	0.5%
connect	0.68MB	1.17MB	gazelle	0.13MB	1.24MB	T40I10D100K	0.39MB	0.52MB

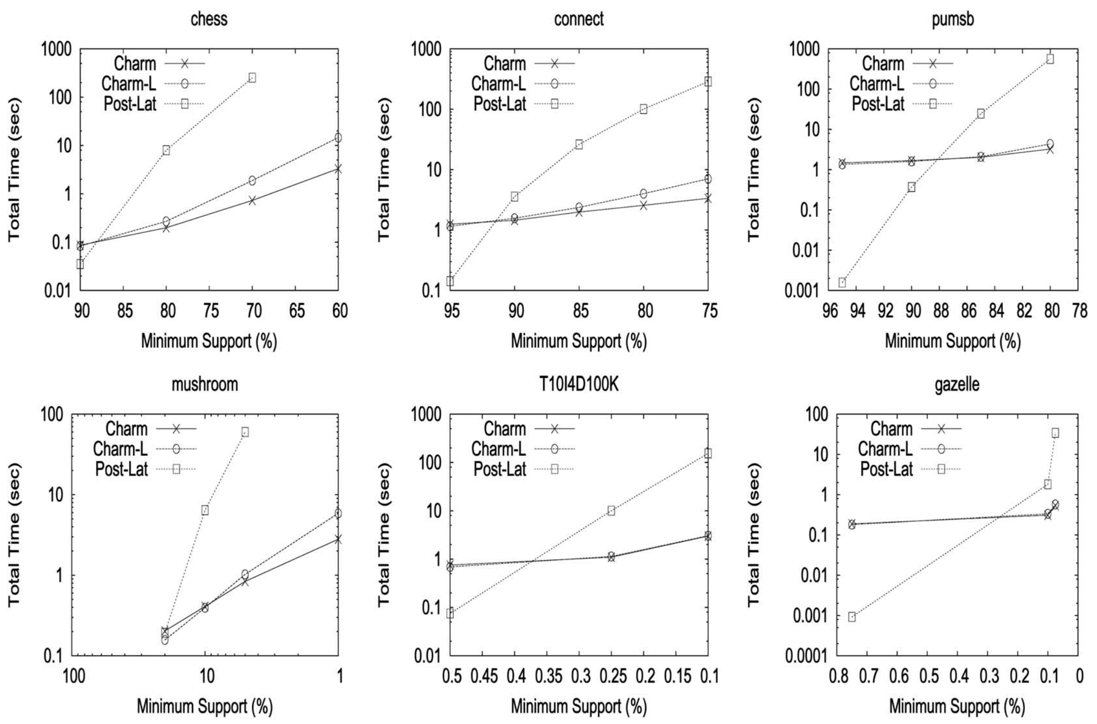


Fig. 21. CHARM-L performance.

new vertical format based on diffsets, i.e., storing the differences in the tids as the computation progresses. An extensive set of experiments confirm that CHARM and CHARM-L can provide orders of magnitude improvement over existing methods for mining closed itemsets. CHARM-L is a state-of-the-art algorithm that generates the frequent closed itemset lattice.

It has been shown in recent studies that closed itemsets can help in generating nonredundant rules sets, which are typically a lot smaller than the set of all association rules [21]. An interesting direction of future work is to develop efficient methods to mine closed patterns for other mining problems like sequences, episodes, multidimensional patterns, etc., and to study how much reduction in their respective rule sets is possible. It also seems worthwhile to explore if the concept of "closure" extends to metrics other than support. For example, for confidence, correlation, etc.

Acknowledgments

The authors would like to thank Lotfi Lakhal and Yves Bastide for providing the source code for Close and Pascal, Jiawei Han, Jian Pei, and Jianyong Wang for sending us the executable for Closet and Closet+, and Johannes Gehrke for the Mafia algorithm. They thank Roberto Bayardo for providing them the IBM real data sets and Ronny Kohavi and Zijian Zheng of Blue Martini Software for giving them access to the Gazelle data set. This work was supported in part by US National Science Foundation (NSF) CAREER Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, and NSF grant EIA-0103708. A previous version of this paper appeared in *the Second SIAM International Conference on Data Mining*, 2002.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo, "Fast Discovery of Association Rules," *Advances in Knowledge Discovery, and Data Mining*, U. Fayyad et al., eds., pp. 307-328, Menlo Park, Calif.: AAAI Press, 1996.

[2] R. Agrawal, C. Aggarwal, and V.V.V. Prasad, "Depth First Generation of Long Patterns," *Proc. Seventh Int'l Conf. Knowledge Discovery and Data Mining,* Aug. 2000.

[3] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal, "Mining Frequent Patterns with Counting Inference," *SIGKDD Explorations,* vol. 2, no. 2, Dec. 2000.

[4] R.J. Bayardo, "Efficiently Mining Long Patterns from Databases," *Proc. ACM SIGMOD Conf. Management of Data,* June 1998.

[5] S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," *Proc. ACM SIGMOD Conf. Management of Data,* May 1997.

[6] D. Burdick, M. Calimlim, and J. Gehrke, "MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases," *Proc. Proc. Int'l Conf. Data Eng.,* Apr. 2001.

[7] D. Cristofor, L. Cristofor, and D. Simovici, "Galois Connection and Data Mining," *J. Universal Computer Science,* vol. 6, no. 1, pp. 60-73, 2000.

[8] B. Dunkel and N. Soparkar, "Data Organization and Access for Efficient Data Mining," *Proc. 15th IEEE Int'l Conf. Data Eng.,* Mar. 1999.

[9] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations.* Springer-Verlag, 1999.

[10] K. Gouda and M.J. Zaki, "Efficiently Mining Maximal Frequent Itemsets," *Proc. First IEEE Int'l Conf. Data Mining,* Nov. 2001.

[11] J. Han and M. Kamber, *Data Mining: Concepts and Techniuqes.* Morgan Kaufmann, 2001.

[12] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. ACM SIGMOD Conf. Management of Data,* May 2000.

[13] D-I. Lin and Z.M. Kedem, "Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set," *Proc. Sixth Int'l Conf. Extending Database Technology,* Mar. 1998.

[14] L. Nourine and O. Raynaud, "A Fast Algorithm for Building Lattices," *Information Processing Letters,* vol. 71, pp. 199-204, 1999.

[15] J.S. Park, M. Chen, and P.S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* May 1995.

[16] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," *Proc. Seventh Int'l Conf. Database Theory,* Jan. 1999.

[17] J. Pei, J. Han, and R. Mao, "Closet: An Efficient Algorithm for Mining Frequent Closed Itemsets," *Proc. SIGMOD Int'l Workshop Data Mining and Knowledge Discovery,* May 2000.

[18] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. 21st Very Large Data Bases Conf.,* 1995.

[19] P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-Charging Vertical Mining of Large Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* May 2000.

[20] J. Wang, J. Han, and J. Pei, "Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* Aug. 2003.

[21] M.J. Zaki, "Generating Non-Redundant Association Rules," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* Aug. 2000.

[22] M.J. Zaki, "Scalable Algorithms for Association Mining," *IEEE Trans. Knowledge and Data Eng.,* vol. 12, no. 3, pp. 372-390, May-June 2000.

[23] M.J. Zaki and K. Gouda, "Fast Vertical Mining Using Diffsets," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* Aug. 2003.

[24] M.J. Zaki and C.-J. Hsiao, "ChARM: An Efficient Algorithm for Closed Association Rule Mining," Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Inst., Oct. 1999.

[25] M.J. Zaki and M. Ogihara, "Theoretical Foundations of Association Rules," *Proc. Third ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery,* June 1998.

[26] M.J. Zaki and B. Phoophakdee, "MIRAGE: A Framework for Mining, Exploring, and Visualizing Minimal Association Rules," Technical Report 03-4, Computer Science Dept., Rensselaer Polytechnic Inst., July 2003.

Mohammed J. Zaki received the PhD degree in computer science from the University of Rochester in 1998. He is an associate professor of computer science at Rensselaer Polytechnic Institute. His research interests focus on developing novel data mining techniques for intelligence applications, bioinformatics, performance mining, Web mining, etc. He has published more than 100 papers on data mining, coedited 11 books, served as guest editor for several journals, has served on the program committees of major international conferences, and has cochaired many workshops (BIOKDD, HPDM, DMKD, etc.) in data mining. He is currently an associate editor for the *IEEE Transactions on Knowledge and Data Engineering*, the *International Journal of Data Warehousing and Mining*, and the *ACM SIGMOD Digital Symposium Collection*. He received the US National Science Foundation CAREER Award in 2001 and the US Department of Energy Early Career Principal Investigator Award in 2002. He also received the ACM Recognition of Service Award in 2003. He is a member of the IEEE.

Ching-Jui Hsiao received the MS degree in computer science from Rensselaer Polytechnic Institute in December 1999. His research interest includes the problem of mining closed association rules.