Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 61 (2007) 93–113
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Association rules mining using heavy itemsets ☆

Girish K. Palshikar [a,*], Mandar S. Kale [b], Manoj M. Apte [b]

[a] *Tata Research Development and Design Centre, Pune 411013, India*
[b] *Engineering and Industrial Services, Tata Consultancy Services Ltd., Pune 411001, India*

Received 9 November 2005; received in revised form 17 February 2006; accepted 18 April 2006
Available online 2 June 2006

Abstract

A well-known problem that limits the practical usage of association rule mining algorithms is the extremely large number of rules generated. Such a large number of rules makes the algorithms inefficient and makes it difficult for the end users to comprehend the discovered rules. We present the concept of a *heavy itemset*. An itemset A is heavy (for given support and confidence values) if all possible association rules made up of items only in A are present. We prove a simple necessary and sufficient condition for an itemset to be heavy. We present a formula for the number of possible rules for a given heavy itemset, and show that a heavy itemset compactly represents an exponential number of association rules. Along with two simple search algorithms, we present an efficient greedy algorithm to generate a collection of disjoint heavy itemsets in a given transaction database. We then present a modified *apriori* algorithm that starts with a given collection of disjoint heavy itemsets and discovers more heavy itemsets, not necessarily disjoint with the given ones.

Keywords: Association rules; Data mining; Knowledge discovery in databases; Knowledge compression

1. Introduction

Association rule mining, as originally proposed in [1] with its *apriori* algorithm, has developed into an active research area. Many additional algorithms have been proposed for association rule mining [2,4,7,16]; see also [10]. Also, the concept of association rule has been extended in many different ways, such as generalized association rules, association rules with item constraints, sequence rules [11] etc. Apart from the earlier analysis of market basket data, these algorithms have been widely used in many other practical applications such as customer profiling, analysis of products and so on. We have used these algorithms in innovative applications such as warranty claims analysis and inventory analysis. Several commercial data mining tools now offer variants of the association rule mining algorithms.

End users of association rule mining tools encounter several well-known problems in practice. First, the algorithms do not always return the results in a reasonable time. Typically, this happens because the

☆ Preliminary version of this paper was published as [9].

* Corresponding author. Tel.: +91 20 403 1122; fax: +91 20 404 2399.
E-mail address: gk.palshikar@tcs.com (G.K. Palshikar).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.04.009

algorithms generate an exponential number of candidate frequent itemsets. Although several different strategies have been proposed to tackle efficiency issues, they are not always successful. Also, in many cases, the algorithms generate an extremely large number of association rules, often in thousands or even millions. Further, the association rules are sometimes very large. It is nearly impossible for the end users to comprehend or validate such large number of complex association rules, thereby limiting the usefulness of the data mining results. Several strategies have been proposed to reduce the number of association rules, such as generating only "interesting" rules, generating only "non-redundant" rules, or generating only those rules satisfying certain other criteria such as coverage, leverage, lift or strength. While these are promising strategies, none of them seem to sufficiently "compress" or reduce the generated association rules, for easy comprehension by end users.

In this paper, we propose a concept called a heavy itemset. An itemset A is heavy (for given support and confidence values) if all possible association rules made up of items only in A are present. We prove a simple necessary and sufficient condition for an itemset to be heavy. We present a formula for the number of possible rules for a given heavy itemset, and show that a heavy itemset compactly represents an exponential number of association rules. For example, we show later that if we are able to find a heavy itemset of 20 items in the given transaction dataset, it means that there is no need to output $O(3^{20})$ association rules, thereby leading to a huge (exponential) decrease in the output size. We present two simple search algorithms and an efficient greedy algorithm to generate a collection of disjoint heavy itemsets in a given transaction database. We then present a modified *apriori* algorithm that uses a given collection of heavy itemsets and detects more heavy itemsets, not necessarily disjoint with the given ones, and of course the remaining association rules. Table 1 shows an example, where the itemset {"97217", "99501-3095", "98055"} is heavy (see last 12 rules).

The concept of heavy itemsets is useful in several practical applications; e.g., in a database of inventory or warranty claims, a group of parts (which forms an assembly) will often occur together. For example, when one assembly is replaced, all the member parts in that assembly are actually replaced. Either these relationships among the items will be known beforehand, or they will constitute an interesting new knowledge. (a) If we already know that some parts always occur together, then they can be modeled as heavy itemsets. In this case, the association rules among the member items of an assembly (which are already known) will be suppressed. This is particularly useful when the assembly consists of a large number of parts. (b) Discovering unknown heavy itemsets is a potentially interesting and useful new knowledge. The algorithms presented in this paper can be used to discover only the heavy itemsets in a given transaction database.

The paper is organized as follows. Section 2 discusses some related work. The formal definition of the concept of heavy itemset and its theoretical properties are discussed in Section 3. Two simple search algorithms and an efficient algorithm to generate a collection of disjoint heavy itemsets are presented in Section 4. Section

Table 1
Some association rules generated from real data

"15238-2901"	⇒"33610"
"33610"	⇒"15238-2901"
"28241-7787"	⇒"33610"
"33610"	⇒"28241-7787"
"28241-7787"	⇒"98055"
"98055"	⇒"28241-7787"
"97217"	⇒"98055"
"98055"	⇒"97217"
"97217"	⇒"99501-3095"
"99501-3095"	⇒"97217"
"98055"	⇒"99501-3095"
"99501-3095"	⇒"98055"
"97217"	⇒"98055", "99501-3095"
"98055"	⇒"97217", " 99501-3095"
"99501-3095"	⇒"97217", " 98055"
"97217", "98055"	⇒"99501-3095"
"97217", " 99501-3095"	⇒"98055"
"98055", " 99501-3095"	⇒"97217"

5 contains the modified *apriori* algorithm, which uses a given collection of disjoint heavy itemsets as input, and generates "remaining" association rules and also finds more heavy itemsets, not necessarily disjoint from the given ones. Section 6 presents some experiments results. Section 7 presents conclusions and further work.

2. Related work

Association rule mining, as originally proposed in [1] with its *apriori* algorithm, has developed into an active research area. Many additional algorithms have been proposed for association rule mining [2,4,7,16]; see also [10]. End users of association rule mining tools encounter several problems in practice, such as too much time taken by the algorithms and too many rules generated as output. Since finding support requires a pass over the entire database (and thus results in high I/O cost), one strategy to improve the efficiency involves the use of random sampling to estimate the support of an itemset; see [6,14]. Use of hash-based efficient data structures [13] and mining of vertical (rather than horizontal) database [12] are some other approaches that have been tried to improve the efficiency of the association rule mining algorithms.

Ref. [17] compared five well-known association rule mining algorithms (viz., *apriori*, FP-tree, charm, Magnum-Opus and Closet) using three real-world data sets and observed super-exponential growth in the number of rules generated on these datasets. Several different strategies have been proposed to reduce the number of association rules generated. For example, we could use an interestingness measure to generate only interesting rules. See [5] for a survey of many interestingness measures proposed in the literature. A closely related work is [15], which proposed the concept of a closed frequent itemset and used it to generate an exponentially smaller number of non-redundant association rules. Several post-processing strategies have been proposed in [8] to reduce the number of generated association rules; e.g., prefer general association rules over specific ones, summarize and report only *direction-setting* (i.e., rules which indicate the type of correlation between specific itemsets) rules etc. In [3], algorithms are given to find itemsets containing highly correlated items i.e., association rules having high confidence but not constrained by any minimum support.

In this paper, we are *not* trying to reduce the number of association rules by choosing some of them and ignoring the others. Instead, we focus on the problem of compactly representing the association rules by means of heavy itemsets. Broadly, our approach is summarized as original association rules = heavy itemsets + remaining association rules. Note that all original association rules can be recovered, if desired, from the outputs of our algorithms.

3. Heavy itemset

When generating association rules over a set of items, we often find that all possible association rules over a subset A of items are generated. Each of these association rules over A has the form $X \Rightarrow Y$ where X and Y are disjoint non-empty subsets of A. These rules clutter the generated output. They can be actually summarized (and hence removed from the output) by stating that A is a heavy itemset.

Definition 1. Let L and R be any two non-empty disjoint itemsets. Let $0.0 < \sigma, \tau \leqslant 100.0$ be the given support and confidence values. We say that $L \Rightarrow R$ is a *valid association rule* (for given σ and τ) if (i) support$(L) \geqslant \sigma$ and (ii) support$(L \cup R) \geqslant \sigma$ and (iii) [support$(L \cup R)$/support(L)] $\geqslant \tau$.

Definition 2. Let $0.0 < \sigma, \tau \leqslant 100.0$ be the given support and confidence values. A non-empty itemset A ($|A| \geqslant 2$) is said to be a heavy itemset (for given σ and τ) if for every non-empty disjoint subsets X, Y of A, $X \Rightarrow Y$ is a valid association rule (for given σ and τ). An element of a heavy itemset is called a heavy item.

It is easy to see that a subset (of cardinality 2 or more) of a heavy itemset is itself a heavy itemset. This observation, along with the efficient condition for checking whether a given itemset is heavy or not (as proved below) leads to a straightforward application of the *apriori* algorithm for mining only heavy itemsets. In each step of *apriori*, instead of finding frequent itemsets, we find heavy itemsets. However, we run into the *same* difficulties of an exponential number of candidates, in case there exist heavy itemsets of large size in a given transaction database. Instead, we use special properties of the heavy itemsets and give below a polynomial time algorithm for generating disjoint heavy itemsets.

We first formalize the concept of heavy itemset, so as to facilitate counting of the rules a heavy itemset represents.

Definition 3. Given a non-empty finite set $A = \{a_1, a_2, \ldots, a_n\}$ of $n \geqslant 2$ items. A 2-*split* (or simply, a *split*) of A is a tuple (X, Y) where X, Y are non-empty disjoint subsets of A (i.e., $X, Y \subseteq A$, $X \neq \emptyset$, $Y \neq \emptyset$ and $X \cap Y = \emptyset$ but $X \cup Y$ need not be all of A). *A split-set* over A is a non-empty set of splits of A. Given a non-empty subset $B \subseteq A$, the *complete split-set* of B is the set of all splits of B.

For example, let $A = \{a, b, c, d\}$. Then $(\{a\}, \{b, d\})$ is a split of A and so are $(\{b\}, \{d\})$, $(\{d\}, \{b\})$ and $(\{b, c\}, \{a, d\})$. The complete split-set of $B = \{a, b, c\}$ containing 12 splits is

$$\{(\{a\},\{b\}),(\{a\},\{c\}),(\{b\},\{a\}),(\{b\},\{c\}),(\{c\},\{a\}),(\{c\},\{b\}),(\{a\},\{b,c\}),(\{b\},\{a,c\}),(\{c\},\{b,a\}),$$
$$(\{a,b\},\{c\}),(\{a,c\},\{b\}),(\{b,c\},\{a\})\}$$

Thus a split represents an association rule; e.g., the split $(\{a\}, \{b, c\})$ stands for the association rule $\{a\} \Rightarrow \{b, c\}$. Then an itemset A is heavy if all the association rules in the complete split-set of A are present with the required support and confidence.

Proposition 1. *The size of the complete split set (i.e., the number of all possible splits) for a given finite set A of size N is given by*

$$\sum_{m=1}^{N} \sum_{n=1}^{N-m} {}^{N}C_m \, {}^{N-m}C_n. \tag{1}$$

Here, m and n denote the sizes of the LHS and RHS sets in a spilt.

Proof. The term ${}^{N}C_m$ denotes the number of ways in which the m elements in the left hand side set can be selected out of the total N elements. The term ${}^{N-m}C_n$ denotes the number of ways in which the n elements in the right hand side set can be selected out of the remaining $N - m$ elements. Since both LHS and RHS need to be selected, we take the product of these two terms (rule of product). Since the number of element m varies from 1 to N and m varies from 1 to $N - m$, we take the summations over all possible values of m and n, to get the total number of splits sets (rule of sum). When $m = N$, we assume that the term ${}^{0}C_n = 0$. □

Thus, if $N = 3$ (e.g., $A = \{a, b, c\}$), the total number of all possible splits is $3\mathrm{C}_1 \cdot (2\mathrm{C}_2 + 2\mathrm{C}_1) + 3\mathrm{C}_2 \cdot (1\mathrm{C}_1) = 3 \cdot 3 + 3 \cdot 1 = 12$, as shown above. If $N = 4$, the total number of possible splits is

$$\begin{aligned} {}^{4}C_1 \cdot ({}^{3}C_3 + {}^{3}C_2 + {}^{3}C_1) + {}^{4}C_2 \cdot ({}^{2}C_2 + {}^{2}C_1) + {}^{4}C_3 \cdot {}^{1}C_1 &= 4 \cdot (1 + 3 + 3) + 6 \cdot (1 + 2) + 4 \cdot 1 \\ &= 28 + 18 + 4 = 50. \end{aligned}$$

It is easy to provide an upper bound on the summation in Proposition 1. Each item in the itemset A has three choices for each split: either it appears in the left side or it appears in the right side or it does not appear in either side. Thus the total number of possible rules for an itemset A of size n is bounded above by 3^n. Since neither the RHS nor the LHS of an association rule can be empty, we need to eliminate such association rules. The number of association rules where the RHS is empty and the LHS is any subset of the itemset is 2^n. Similarly, the number of association rules where LHS is empty is also 2^n. Since the empty association rule (LHS = RHS = $\emptyset$) is counted twice, an alternative expression for the number of association rules corresponding to a heavy itemset of n items: $3^n - 2^{n+1} + 1$. This expression is equivalent to the one in Proposition 1.

Corollary 2. *A heavy itemset compactly represents an exponential number of association rules.*

Proof. Immediately follows from the expression $3^n - 2^{n+1} + 1$ for the number of association rules corresponding to a heavy itemset of n items proved in the preceding paragraph. □

The significance of a heavy itemset is then as follows. If we identify an itemset H as heavy, then we need not report the association rules corresponding to H. Thus identifying a heavy itemset enables an exponential compression in the number of association rules reported by an algorithm for mining association rules.

We now prove a simple necessary and sufficient condition to decide whether an itemset is heavy or not.

Proposition 3. *Suppose* $H = \{a_1, a_2, \ldots, a_n\}$ *is an itemset of* $n \geqslant 2$ *items. Let* s_0 *be the support of* H. *Let* $s_1, s_2, \ldots, s_n$ *be the supports of the singleton itemsets* $\{a_1\}, \{a_2\}, \ldots, \{a_n\}$, *respectively. Let the minimum required support and confidence be* σ *and* τ. *Let* $mn = \min\{s_0, s_1, s_2, \ldots, s_n\}$ *and* $mx = \max\{s_0, s_1, s_2, \ldots, s_n\}$ *be the minimum and maximum of the support values. Then* H *is a heavy itemset iff* (i) $mn \geqslant \sigma$ *and* (ii) $(mn/mx) \geqslant \tau$.

Proof. (*only-if part*) Assume that (i) $mn \geqslant \sigma$ and (ii) $(mn/mx) \geqslant \tau$. H is a heavy itemset iff $L \Rightarrow R$ is a valid association rule for any disjoint non-empty subsets L and R of H. Let L and R be any two disjoint non-empty subsets of H. We need to prove that (i) support(L) $\geqslant \sigma$ and (ii) support($L \cup R$) $\geqslant \sigma$ and (iii) [support($L \cup R$)/support(L)] $\geqslant \tau$. Clearly, support(L) $\geqslant mn$. Given that $mn \geqslant \sigma$, it follows that support(L) $\geqslant \sigma$. Similarly, we can prove that support($L \cup R$) $\geqslant \sigma$. [support($L \cup R$)/support(L)] $\geqslant mn$/support(L) $\geqslant mn/mx$ which is given to be $\geqslant \tau$; hence [support($L \cup R$)/support(L)] $\geqslant \tau$ as required. Hence $L \Rightarrow R$ is a valid association rule. Since L, R are arbitrary disjoint subsets of H, we have proved that H is a heavy itemset.

(*if part*) Assume that H is a heavy itemset. We need to prove that (A) $mn \geqslant \sigma$ and (B) $(mn/mx) \geqslant \tau$. Since $mx = \max\{s_0, s_1, s_2, \ldots, s_n\}$, let $L = \{a_k\}$ be the singleton itemset such that support(L) = support($\{a_k\}$) = mx. Let $R = H \backslash L$ be the itemset of all items from H except a_k. Clearly, L, R are two disjoint subsets of H such that $L \cup R = H$ and $L \cap R = \emptyset$. Then since H is a heavy itemset, $L \Rightarrow R$ is a valid association rule. Hence, (i) support(L) $\geqslant \sigma$ and (ii) support($L \cup R$) $\geqslant \sigma$ and (iii) [support($L \cup R$)/support(L)] $\geqslant \tau$. Since support($L \cup R$) = $s_0 = mn$, it follows from (ii) that $mn \geqslant \sigma$, thus proving (A) we now prove that (B) $mn/mx \geqslant \tau$. In (iii), the numerator is the same as mn and the denominator is the same as mx and hence it follows by substitution that $mn/mx \geqslant \tau$. □

We present below an algorithm that returns yes or no depending on whether or not the given itemset is heavy. Note that the algorithm uses the condition proved in Proposition 3 and *does not* explicitly check all the possible association rules to decide whether H is heavy or not.

algorithm is_heavy // checks whether given itemset is heavy or not
input database D of N transactions
input itemset $H = \{a_1, a_2, \ldots, a_n\}$ of $n \geqslant 3$ items
input minimum support σ, minimum confidence τ
output yes if H is a heavy itemset; **no** otherwise
1. Let s_0 be the support of H in D;
2. Let $s_1, s_2, \ldots, s_n$ be the supports of $\{a_1\}, \{a_2\}, \ldots, \{a_n\}$ in D respectively;
3. Let $mn = \min\{s_0, s_1, s_2, \ldots, s_n\}$ and $mx = \max\{s_0, s_1, s_2, \ldots, s_n\}$ be the minimum and maximum of the support values;
4. **if** $mn < \sigma$ **then** *return*(**no**);
5. **if** $(mn/mx < \tau)$ **then** *return*(**no**);
6. *return*(**yes**);

Correctness of the algorithm follows from Proposition 3. Checking this condition for a given itemset A requires one pass over the database to obtain support for each of the singleton subset of A (which needs to be done once only) and for A itself. The algorithm's complexity is $O(N)$, where $N =$ no. of transactions.

4. Finding disjoint heavy itemsets

In this section, we present three different approaches to find out a disjoint collection of heavy itemsets. The first two approaches are more conceptual and the third is more suited for practical applications, since it is based on a well-known efficient data structure called FP-tree.

4.1. Searching the subset tree

In this section, we present a characterization of the search space of heavy itemsets and give a simple algorithm to find a collection of disjoint heavy itemsets.

Definition 4. Let F_1 be the set of $|F_1| = n$ frequent itemsets in a given transaction database (for a given support and confidence). A collection of pairwise disjoint subsets of F_1 is called a sub-partition of F_1. A heavy-sub-partition of F_1 is a sub-partition P of F_1, where each element of P is a heavy itemset. A heavy-sub-partition C of F_1 is maximal if no subset of $F_1 - \cup C$ is heavy, where $\cup C$ denotes the union of all elements of C.

Without loss of generality, assume that the items in F_1 are numbered $1, 2, \ldots, n$. As an example, $C = \{\{1,2,5\},\{4,8\}\}$ is a sub-partition of the set $F_1 = \{1,2,3,4,5,6,7,8,9\}$; here, $\cup C = \{1,2,5,4,8\}$. Given F_1, we need an algorithm to find a maximal heavy-sub-partition of F_1. We later give examples to illustrate the fact that there is no unique maximal heavy-sub-partition for a given F_1 i.e., a given F_1 may have many different maximal heavy-sub-partitions.

All possible subsets of F_1 can be arranged in a standard *subset tree*. A vertex u in the subset tree of F_1 is labeled with a subset $A \subseteq F_1$; this vertex u has at most $|A|$ children, each of which is labeled with a subset of A having size $|A| - 1$. These children of u are arranged from left to right, as per the numeric order of the subsets, assuming that elements within each subset are sorted in ascending order. If any of these $|A|$ subsets has already occurred in the subtree of the left siblings of u then it is not added as a child for u. The root of the subset tree is labeled with the entire set F_1. We stop expanding the tree when $|A| = 2$.

Fig. 1 shows the subset tree for $F_1 = \{1,2,3,4\}$. The following properties of a subset tree are obvious.

Proposition 4. *Each subset of F_1 of size $\geqslant 2$ appears exactly once in the subset tree of F_1. Also, the number of vertices in the subset tree of a set of size n is $2^n - n - 1 = O(2^n)$.*

We can devise a simple algorithm to find a maximal heavy-sub-partition of a given set F_1. This algorithm essentially performs an implicit breadth-first search of the subset tree of F_1; the search is implicit in the sense that the entire subset tree is not constructed *apriori* before the search. In each iteration, this algorithm constructs all children of a particular node and tests each of them for heavy itemset. If a heavy itemset is found, it is added to the collection of heavy itemsets found so far, removes the items in it from the set of items available for further construction of subsets. The algorithm stops when there are no more subsets to check.

algorithm BFS_subset_tree
input set F_1 of frequent 1-itemsets
input Transactions Database D
output a maximal heavy-sub-partition of F_1
$C := \emptyset$; // collection of heavy itemsets found so far
$R := F_1$; // $R = F_1 - \cup C$; remaining itemsets
$L := \langle F_1 \rangle$; // list of current subsets of F_1
1. **while** $R \neq \emptyset \wedge L \neq \emptyset$ **do**
2. **for** $j = 1$ to $|L|$ **do**
3. $newL := \emptyset$;
4. $A :=$ jth subset in L;
5. **if** $|A| \geqslant 2 \wedge$ is_heavy(A) **then**
6. $C := C \cup \{A\}$;
7. $R := R - A$;
8. **for each** B in L **do** $B := B - A$;
9. **else if** $|A| > 2$ **then**
10. append each of the $|A|$ subsets of A to $newL$ if it is not already in $newL$;
11. **end if**
12. **end for**
13. $L := newL$; // remove old stuff from L; copy $newL$ in it
14. **end while**

In the worst case, the above algorithm has to visit *all* vertices of the subset tree, which are $O(2^n)$, where $n = |F_1|$. Since the complexity of breadth-first search is $O(|V| + |E|)$ where V and E are sets of vertices and edges in the graph being searched, the above algorithm takes a time $O(2^n)$ in the worst case. The worst case occurs when there are no heavy itemsets in the given transaction database (for given support and confidence values).

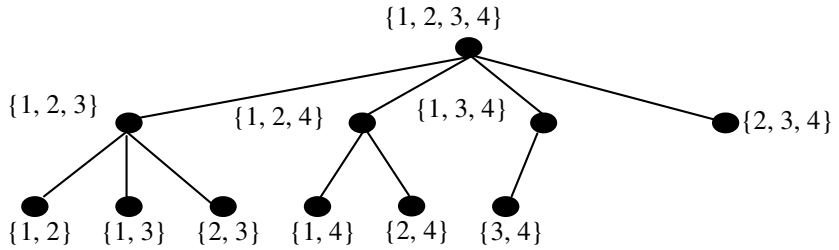


Fig. 1. The subset tree for $F_1 = \{1,2,3,4\}$.

It is easy to see the above algorithm has no "focus" while searching the subset tree. For example, if $\{3,4\}$ were a heavy itemset then the algorithm would reach it only after searching the previous (left) sub-trees, such as the sub-trees rooted at $\{1,2,3\}$ and $\{1,2,4\}$. It is possible to use different strategies, such as depth-first, to search the subset tree.

4.2. Searching the heavy itemset tree

We now present another algorithm that performs a depth-first search of the subset tree in a bottom up manner. Notice that each node in the subset tree is really a candidate heavy itemset. The next algorithm uses a pruning strategy to reduce the search space. Given a candidate heavy itemset H, the idea is to consider only those of the remaining items that can possibly be added to H.

algorithm *heavy*1 // try to find a heavy itemset of given size k
input D // transaction database
input σ, τ // support and confidence
input $F1$ // set of frequent items
input k // desired number of items in H
input L // items which can be added next to H
input CH // the heavy itemset (of size $< k$) constructed so far
1. $L1$:= set of all items J in L but not in CH such that support of CH $\cup \{J\} \geqslant \sigma$ arranged in descending order of support
2. **if** ($|L1 \cup \mathrm{CH}| < k$) **then return** $\emptyset$ **endif**
3. **for every** item $I \in L1$ **do**
4. **if** (get_confidence(CH $\cup \{I\}$) $\geqslant \tau$) **then**
5. **if** ($|\mathrm{CH} \cup \{I\}| == k$) **then**
6. **return** CH $\cup \{I\}$
7. **else**
8. $H := heavy1(D, \sigma, \tau, F1, k, L1 \backslash \{I\}, \mathrm{CH} \cup \{I\})$ // recursive call
9. **if** ($H! = \emptyset$) **then return** H **endif**
10. **endif**
11. **endif**
12. **endfor**
13. **return** $\emptyset$

algorithm find_disjoint_heavy_itemsets_simplified
input D // transaction database
input σ, τ // support and confidence
output S // collection of disjoint heavy itemsets
1. Let F_1 := the list of frequent 1-itemsets in D arranged in descending order of support
2. $S := \emptyset$
3. **for** $k := |F1|$ to 2 **do** // start with highest possible value for k

4. **while** $|F1| > k$ **do**
5. $H := heavy1(D, \sigma, \tau, F1, k, F1, \emptyset)$ // try to find a heavy itemset of size k
6. **if** $(H! = \emptyset)$ **then** $S := S \cup \{H\}$; $F1 := F1 \backslash H$ **else break endif** // add H to S
7. **end while**
8. **end for**

We recursively define a vertex-labeled, rooted tree called the *heavy itemset tree* T as follows.

Definition 5. Each node in *heavy itemset tree* T is labeled with a candidate heavy itemset CH and an itemset $L1$ disjoint from CH, indicating the items that can be added to CH. The root of T is labeled with $CH = \emptyset$ and $L1 = F1$, where $F1$ is the set of all frequent items in the given transaction database. Each node has a child for every item in $L1$. The candidate heavy itemset for a child is $CH_a = CH \cup \{a\}$, for an item $a \in L1$.

It is easy to see that the algorithm *heavy*1 implicitly constructs and performs a depth-first search of the heavy itemset tree T. At a particular level of recursion, the algorithm *heavy*1 constructs and searches all children of T at that level. The depth of this tree is at most k, because the size of CH is incremented by 1 at each level and the algorithm *heavy*1 stops when it reaches a vertex in T with $|CH| = k$ and when that CH is indeed a heavy itemset. For a vertex u in T at depth m, the algorithm does not create and search any of its child vertices, if size of the list $L1$ at u is not enough to reach the "remaining" depth $k - m$ i.e., no leaf in the sub-tree at u can have CH of size k (m is also equal to the size of CH at that level).

Lemma 5. *Every subset of size k of the set F*1 *at the root of the tree T either* (i) *occurs as a candidate heavy itemset* CH *for some leaf in T or* (ii) *it is certain that it is not heavy. Formally, we need to prove that every subset of size m of the set F*1 *is either the* CH *for some vertex of T at level m* + 1 *or it is not heavy.*

Proof. See Appendix A. □

Proposition 6. *Algorithm heavy*1 *terminates, returning a heavy itemset H where* $H \subseteq F1$ *and* $|H| = k$ *and returning* $\emptyset$ *if no such heavy itemset exists.*

Proof. See Appendix A. □

We can improve the performance of *heavy*1 by adopting the strategy used in *apriori*. Assume that each item has a unique integer ID. Then we can replace Line 1 in *heavy*1 by the following line, which would reduce repeated construction of already tested itemsets.

1. $L1$:= set of all items J in L but not in CH such that support of $CH \cup \{J\} \geqslant \sigma$ and $J >$ every ID in CH in ascending order of item IDs.

Proposition 7. *Algorithm find_disjoint_heavy_itemsets_simplified terminates, returning a collection S containing disjoint heavy itemsets. Moreover, the algorithm finds and reports one of the largest itemsets i.e., if the algorithm reports the largest heavy itemset of size say k, then there is no heavy itemsets of size k* + 1.

Proof. See Appendix A. □

We remark that the above algorithm finds and returns the heavy itemset having the maximum size; however, we do not say anything about the size of the other heavy itemsets returned.

The worst case for the above algorithm will be such that almost the entire heavy itemset tree is traversed. One way in which this can happen is as follows. Consider the situation where the set $F1$ contains M frequent items and every m-size subset (except one) of $F1$ is heavy and no subset of $F1$ of size $(m + 1)$ is heavy. In this case, almost the entire heavy itemset tree up to depth m will be traversed and this means that an exponential number of nodes are examined by the algorithm *heavy*1 (when called with $k = m + 1$).

Fig. 2 shows the heavy itemset tree searched by this algorithm on the transaction database in Table 2.

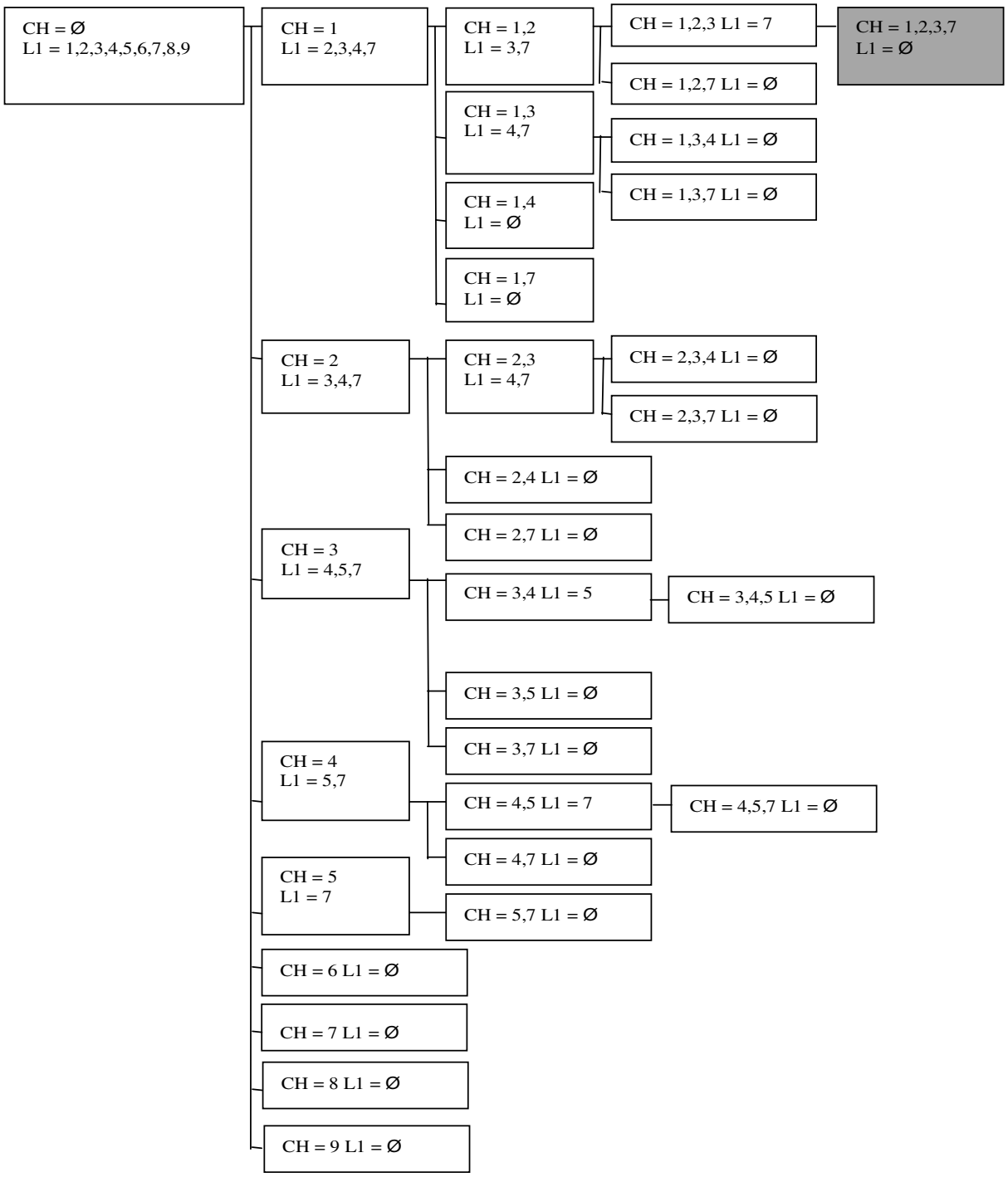


Fig. 2. Heavy itemset tree for the transaction database in Table 2.

4.3. FP-Tree

We can use any association rule mining algorithm to detect all heavy itemsets from the data. We only need to add a check for heavy itemset, once a frequent itemset is found. Different association rule mining algorithms have different strengths and weaknesses in practice, though all seem to have the same worst-case behaviour. Frequent pattern (FP) tree algorithm [4] seems more suitable for adaptation for fast detection of large heavy itemsets, because of the following reasons: (a) it is not a level-wise algorithm unlike *apriori* and others; (b) it

Table 2
Transaction, frequent 1-itemsets and pruned transactions

Original transactions	Frequent 1-itemsets item (support)	Pruned, sorted transactions
2,3,4,5,6,7,8	3 (10)	3,2,7,4,5,6,8
4,5,6,7,8,9	2 (9)	7,4,5,6,8,9
1,2,3,4,5	7 (9)	3,2,1,4,5
1,2,3,6,7	1 (7)	3,2,7,1,6
1,2,3,7	4 (6)	3,2,7,1
1,2,3,7,8	5 (5)	3,2,7,1,8
2,3,7,9,10,11	6 (5)	3,2,7,9
1,2,3,4,5,6,7	8 (5)	3,2,7,1,4,5,6
6,7,8,9,10,11	9 (4)	7,6,8,9
1,2,3,4,8,9,10,1		3,2,1,4,8,9
1		3,2
2,3		3,7,1,4,5
1,3,4,5,7		

works well on itemset space as well as transaction space and (c) seems suitable for storing and analyzing dense data.

We present a modified FP-tree algorithm to detect a disjoint collection of heavy itemsets in given transactions. Intuitively, the modifications are as follows:

- We force it to look for heavy itemsets of specific size, instead of discovering all heavy itemsets; and
- Once a heavy itemset h is found, we ignore all items from h while doing further analysis, thereby leading to a discovery of disjoint heavy itemsets.

Thus we modify the FP-tree algorithm to detect a collection H of large disjoint heavy itemsets from the given transactions. Once such a collection H is available, we show later how to modify of the *apriori* algorithm to discover all remaining association rules, along with any remaining heavy itemsets, not necessarily disjoint from those in H.

We first summarize the concept of the frequent-pattern tree, introduced in [4]. A *frequent-pattern tree* (*FP-tree*) consists of (i) a *header*, which is a sorted list of frequent itemsets in the descending order of their support, (ii) vertices consisting of items in the header, (iii) solid edge from an item u to item v having a *link count* c if support of $u \geqslant$ support of v and u and v co-occur in c transactions, (iv) dashed edge from each item u in the header to the first vertex in the tree for u, (v) dashed edge from each vertex u to the next vertex for u in the tree. Thus, all the vertices in the tree for the same item u are ordered from left to right and connected into a sequence by dashed edges. Dashed edges do not have any count values. An FP-tree is a representation of all the transactions pruned to contain only items in its header. Support for an item u in the header is equal to the sum of the link counts of all solid edges that come into a vertex in the tree labeled with u. The algorithm for constructing an FP-tree is as follows. For simplicity, we omit the construction of the header of FP-tree, which contains frequent items in the given transaction database and their links.

algorithm build_fp_tree
input Transactions Database D
output FP-Tree T
1. Create a root node T of FP-Tree and label it as *null*
2. **for** every transaction $t \in D$ **do**
3. **if** t is not empty **then**
4. insert (t, T)
5. link new nodes to other nodes with similar labels links originating from header list
6. **endif**
7. **end for**
8. **return** FP-Tree T

algorithm insert
input transaction t, any_node
1. **while** t is not empty **do**
2. **if** any_node has a child node with label *head*(t) **then**
3. increment link count between any_node and *head*(t) by 1
4. **else**
5. create a new child node T_0 of any_node with label *head*(t) and having link count 1
6. **endif**
7. **end while**
8. **call** insert(*body*(t), T_0)

First column of Table 2 shows a database containing $N = 12$ transactions of 11 items. For $\sigma = 33\%$, $\tau = 33\%$, the 9 frequent 1-itemsets are shown in column 2 of Table 2. Column 3 of Table 2 shows each of the original transactions after (i) pruning the infrequent items (10, 11) and (ii) sorting the remaining items in the transaction in descending order of their support. Fig. 3 shows the FP-tree for these pruned and sorted transactions.

The *path label* for a path in an FP-tree is the minimum link count in that path; e.g., in Fig. 3, the path label of the path $\langle \text{root}, 3, 2, 7, 1 \rangle$ is $\min\{10, 9, 6, 4\} = 4$.

Given an item u in the header of an FP-tree T, we construct a new conditional FP-tree T_u for u as follows. T_u is the same as T except that it does not contain any vertex labeled with v (and solid edges incident on them) where v comes "after" u in the header of T or there is no path in T from the root to the vertex v which passes through u. The later condition essentially removes those transactions in which u is not present. Such items v are also removed from the header of T_u. The link counts in T_u and support values for items in the header are different from those in T; these are initialized to 0 and re-calculated as follows. Consider the frequent item 1; there are three paths $\langle 3, 2, 7, 1 \rangle$, $\langle 3, 2, 1 \rangle$, $\langle 3, 7, 1 \rangle$ to the three vertices labeled 1 in the FP-tree T of Fig. 3. These paths represent a total of seven transactions, since support(1) = 7 in the header of T. The path label for $\langle 3, 2, 7, 1 \rangle$ is 4; so add 4 to each link count in this path. The path label for $\langle 3, 2, 1 \rangle$ is 2; so add 2 to each link count in this path. Finally, the path label for $\langle 3, 7, 1 \rangle$ is 1; so add 1 to each link count in this path. Now we find

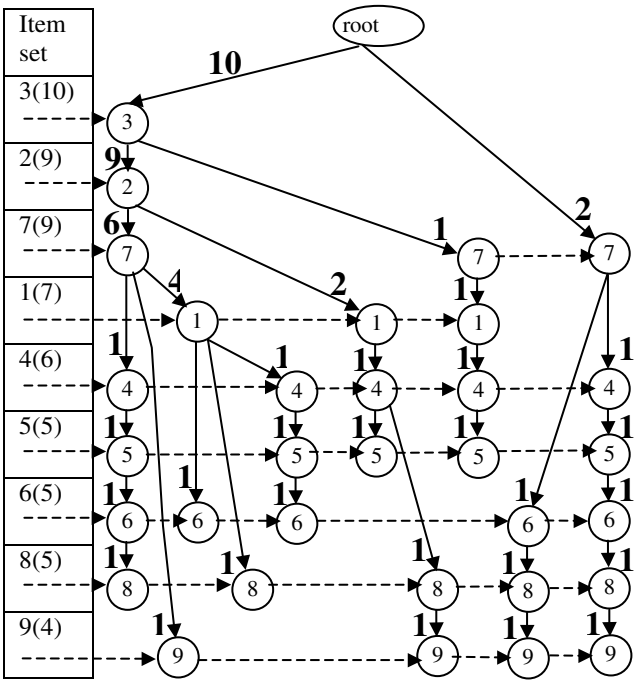


Fig. 3. FP-tree for the transactions in Table 2.

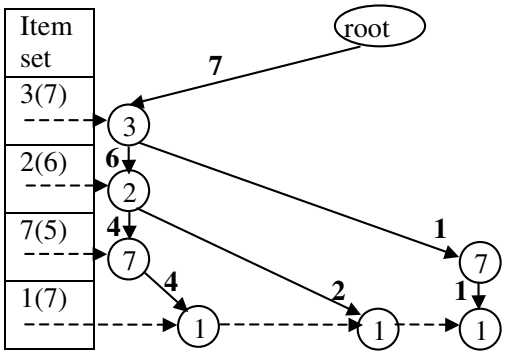


Fig. 4. Conditional FP-tree for frequent item 1.

the new support for an item u in the header of T_u as the sum of the link counts of all solid edges that come into a vertex in the conditional FP-tree labeled with u. Fig. 4 shows the conditional FP-tree for frequent item 1.

4.4. Finding disjoint heavy itemsets using FP-Tree

We now present an FP-tree based algorithm to find a collection of *disjoint* heavy itemsets for a given database D, support σ and confidence τ.

algorithm find_disjoint_heavy_itemsets
input database D of N transactions.
input support s, confidence t
output a collection S of disjoint heavy itemsets
1. $S = \emptyset$; $h = \emptyset$; // $h =$ current heavy itemset being built
2. $(HD, T) =$ build_fp_tree(D) // HD is the header
3. $nDepth = |HD|$; // size of heavy itemset we are looking for
4. $nHeavy = |HD|$; // size of largest possible heavy itemset
5. **while** ($nDepth > 1$) **do**
6. **if** ($|h| > 0$) **then** // found heavy itemset h
7. $S = S \cup \{h\}$; $h = \emptyset$; // add h to S, re-initialize h to empty
8. delete all elements of h from HD
9. re-initialize all links in HD to NULL
10. $nHeavy = nHeavy - |h|$
11. $nDepth = nHeavy$
12. Delete the old FP-tree T
13. $T =$ build_fp_tree(HD, D) // build new FP-tree
14. **endif**
15. **for** every item I in HD **do** // start from last item
16. conditional_treeR (T,HD, I, σ, τ, $nDepth$, $\emptyset$, h)
17. $nHeavy = nHeavy$–1
18. **if** $nHeavy < nDepth$ **then break**; **endif**
19. **end for**
20. $nDepth = nDepth$–1
21. **end while**
22. **return** S

algorithm conditional_treeR
input T, HD, I, σ, τ, $nDepth$ // FP_tree, header, item, support,...

input L // list of items for which conditional tree is built
output h; // heavy itemset found (h is both an input and output parameter)
1. $(HD1, T1) =$ build_conditional_fp_tree(T, HD, I)
2. Remove all elements of $HD1$ for which the support of the item (as per $T1$) in the element is $<\sigma$.
3. $L = L \cup \{I\}$
4. $\text{CH} = L \cup HD1$ // candidate heavy itemset
5. $nHeavy = |\text{CH}|$
6. **if** ($nHeavy < nDepth$) **then return** 1 **endif** // don't record this itemset on return path
7. **if** ($|HD1| > 1$) **then**
8. **if** (get_confidence$(L) < \tau$) **then return** 0 **endif**
9. $found = 0$;
10. **for** every item $I1 \in HD1$ such that $I1 \neq I$ **do**
11. $rec =$ conditional_treeR$(T1, HD1, I1, \sigma, \tau, nDepth, L, h)$
12. **if** ($rec! = 0$) **then** $found = 1$ **endif**;
13. $nHeavy = nHeavy - 1$
14. **if** ($nHeavy < nDepth$) **then return** 1 **endif**
15. **end for**
16. **if** ($found == 0$) **then**
17. **if** (get_confidence$(L) > \tau$) **then** $\{h = h \cup L$; **return** 1;$\}$
18. **else return** 0; **endif**
19. **else return** 1; **endif** // found something during recursion, so return 1
22. **else** // come here when $|HD1| \leqslant 1$
23. **if** (get_confidence$(L) > \tau$) **then** $\{h = h \cup \text{CH}; rec = 1;\}$
24. **else** $rec = 0$; **endif**
24. **endif**
25. **return** rec

The subroutine build_fp_tree is the same as the one in [10]. The subroutine get_confidence computes the confidence of an itemset A as per the formula $100 * (\text{support}(A)/mx)$, where $mx = \max\{\text{support of elements of } A\}$.

We now illustrate our algorithm for finding heavy itemsets. Initially, $nDepth = nHeavy = 9$, which is the number of frequent 1-itemsets. The subroutine conditional_treeR builds the conditional FP-tree $T1$ for item 9 and finds that there are no other items in the header $HD1$ of $T1$. The condition in line 6 is violated, the subroutine returns, $nHeavy$ is decremented, condition in line 17 is violated, the **for** loop is exited. Thus there is no possibility of getting a heavy itemset of size 9. Hence $nDepth$ is decremented and the algorithm starts looking for a heavy itemset of size 8. The algorithm continues and does not find any heavy itemsets of size upto 5 or more. Suppose $nDepth = nHeavy = 4$. The algorithm does not find any heavy itemset of size 4 for conditional FP-trees of items 9, 8, 6, 5, 4. Suppose now $I = 1$ in line 14 in find_disjoint_heavy_itemsets. We now enter the subroutine conditional_treeR and build the conditional FP-tree for 1 (Fig. 4).

No item is removed from $HD1$ in line 2. Now $L = \{1\}$ and $\text{CH} = \{3,2,7,1\}$. Since support$(\{1\}) = 4$, the get_confidence condition is satisfied in line 9. We now recursively enter subroutine conditional_tree, with $I1 = 7$. In the recursive call, conditional FP-tree for 7 is built using the conditional FP-tree for 1, whose header contains $HD1 = \{3,2,7\}$, none of which are removed in line 2. Now $L = \{1,7\}$, $\text{CH} = \{3,2,7,1\}$. Since support$(\{1,7\}) = 4$ (as per the conditional FP-tree of 7) and $\max\{\text{support}(\{1\}), \text{support}(\{7\}) = 9$, as per the header in original FP-tree T, confidence $= (4/9) * 100 = 44.44 > 33\%$. Continuing similarly, we reach and satisfy the base condition for recursion in line 23; thus finding and reporting the heavy itemset $\{3,2,7,1\}$. These four items are then removed from the header of the original FP-tree and the algorithm continues, finding another heavy itemset $\{4,5\}$. The algorithm stops after reaching $nDepth = 1$ in find_disjoint_heavy_itemsets. The worst-case complexity of the algorithm find_disjoint_heavy_itemsets is the same as that of the original FP-tree algorithm (exponential in the worst case).

5. The *apriori*_heavy algorithm

We have observed in practice that, in general, a given transaction database contains non-overlapping (i.e., non-disjoint) heavy itemsets; e.g., $\{3,9\}$ and $\{2,5,7,9\}$. Thus the goal is to detect *all* heavy itemsets in the given transactions, not only the disjoint heavy itemsets. We presented earlier a modified FP-tree algorithm to detect a collection H of large disjoint heavy itemsets from the given transactions. Once such a collection H is available, we now show a modification of the *apriori* algorithm to discover all remaining association rules, along with *all* remaining heavy itemsets, not necessarily disjoint from those in H.

Suppose A is the set of all frequent 1-itemsets in a given transaction database D. Suppose also that we find a collection $H = \{h_1, h_2, \ldots, h_k\}$ of k heavy itemsets in D using the above algorithm. Let B be the set of all frequent items in A, which do not occur in any heavy itemset in S. Apart from the association rules consisting of items only in B, there may be additional association rules involving (i) relationships between items in different heavy itemsets in H; and (ii) relationships between items in B and items in one or more heavy itemsets in H. We now give an association rule mining algorithm, which uses H and B as given inputs and finds the set of all other "missing" association rules. The algorithm also finds more heavy itemsets, not necessarily disjoint from the given ones and adds them to H. Thus the generated collection of heavy itemsets H and the generated association rules complete the mining process.

algorithm *apriori*_heavy
input set of heavy itemsets $H = \{h_1, h_2, \ldots, h_k\}$ // known heavy itemsets; H can be $\emptyset$
input set of frequent item $B = \{f_1, f_2, \ldots, f_m\}$ where each $\{f_i\}$ is a frequent 1-itemset and $f_i \notin h_j$, for any h_j in H
input transaction database D, support σ, confidence τ
output set of association rules $L \Rightarrow R$
output H // more heavy itemsets are added to H given as input
1. $C_2 = \{\{u,v\} |$ ($\exists 1 \leqslant i, j \leqslant k$ such that $u \in h_i \wedge v \in h_j \wedge i \neq j$) $\vee$ ($\exists 1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant k$ such that $u = f_i \wedge v \in h_j$) $\vee$ ($\exists 1 \leqslant i,j \leqslant m$ such that $i \neq j \wedge u = f_i \wedge v = f_j$)$\}$
2. Find support of all 2-itemsets in C_2 to determine L_2
3. move_to_heavy(L_2, H)
4. $k := 3$; $stop = 0$;
5. **while** $stop = 0$ **do**
6. C_k = gen_candidate_itemsets(H, B, k, L_{k-1})
7. prune(C_k, H, B)
8. L_k = set of all candidates in C_k having support $\geqslant \sigma$
9. **if** $L_k = \emptyset$ **then** $stop = 1$; **endif**
10. move_to_heavy(L_k, H)
11. **end while**
12. k++
13. answer $= \cup L_k$

algorithm move_to_heavy
input L_k, H // lists of frequent k-itemsets and heavy itemsets
1. **for all** itemsets $l \in L_k$ **do**
2. **if** is_heavy(l) **then**
3. $H = H \cup \{l\}$
4. Remove all strict subsets of l from H
5. $L_k = L_k - \{l\}$
6. **endif**
7. **end for**

algorithm gen_candidate_itemsets
input set of heavy itemsets $H = \{h_1, h_2, \ldots, h_k\}$

input set of frequent item $B = \{f_1, f_2, \ldots, f_m\}$ where each $\{f_i\}$ is a frequent 1-itemset and $f_i \notin h_j$, for any h_j in H
input k, set L_{k-1} of frequent itemsets of size $k-1$
1. $C_k = \emptyset$
2. **for all** itemsets $l_1 \in L_{k-1} \cup H$ such that $|l_1| = k-1$ **do**
3. **for all** itemsets $l_2 \in L_{k-1} \cup H$ such that $|l_2| = k-1$ **do**
4. **if** $(l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \cdots \wedge l_1[k-1] < l_2[k-1])$ **then**
5. $c = l_1[1], l_1[2], \ldots, l_1[k-1], l_2[k-1]$
6. **if** $c \not\subseteq h'$ for any $h' \in H$ **then** $C_k = C_k \cup \{c\}$ **endif**
7. **endif**
8. **end for**
9. **end for**
10. **for all** heavy itemsets $h \in H$ **do**
11. **for all** itemsets $l \in L_{k-1} \cup H$ such that $|l| = k-1$ **do**
12. **if** (l does not contain any item from h) **then**
13. **for all** items $i \in h$ **do**
14. $c = l \cup \{i\}$
15. **if** $c \not\subseteq h'$ for any $h' \in H$ **then** $C_k = C_k \cup \{c\}$
16. **else** // l and h are not disjoint
17. let i_0 be the largest item of h present in l
18. **for all** items $i \in h \wedge i > i_0$ **do**
19. $c = l \cup \{i\}$
20. **if** $c \not\subseteq h'$ for any $h' \in H$ **then** $C_k = C_k \cup \{c\}$ **endif**
21. **end for**
22. **endif**
23. **end for**
24. **endif**
25. **end for**
26. **end for**

The algorithm *apriori_heavy* is nearly the same as the original *apriori* algorithm, except for the following. The initial candidate itemsets are of size 2, obtained by taking pair-wise Cartesian product of the heavy itemsets in H among themselves and with the set B of "non-heavy" frequent items. After finding the frequent k-itemsets, the algorithm checks (using subroutine is_heavy) if any of them are heavy itemsets; if so, then it removes that set from L_k and adds it to H, taking care to remove all proper subsets of the newly added heavy itemset from H. Since the set L_k may become empty in this process, the terminating condition stated differently (stop when no new frequent itemsets are found).

In algorithm gen_candidate_itemsets, lines 2–7 are the nearly same as the original candidate generation scheme in *apriori*; except that heavy itemsets of size $k-1$ are also treated as frequent $(k-1)$-itemsets. Rest of the lines 7–17 pick a frequent $(k-1)$-itemset (or a $(k-1)$ size heavy itemset from H) and systematically add one heavy element from H to it. The itemsets are assumed to be sorted as per the (arbitrary) item IDs.

A small modification is needed to actually generate the association rules from the resulting heavy itemsets and frequent itemsets. We omit the details here.

The trace of this algorithm on the database of Fig. 3 is as follows. We show candidates generated by the algorithm gen_candidate_itemsets. Some of these will be removed by the prune algorithm.

$$k = 2,\ H = \{\{1,2,3,7\},\{4,5\}\},\ B = \{6,8,9\}$$

$$C_2 = \{\{1,4\},\{1,5\},\{1,6\},\{1,8\},\{1,\},\{2,4\},\{2,5\},\{2,\},\{2,8\},\{2,9\},\{3,4\},\{3,5\},\{3,6\},\{3,8\},\{3,9\},\{4,6\},$$
$$\{4,7\},\{4,8\},\{4,9\},\{5,6\},\{5,7\},\{5,8\},\{5,9\},\{6,7\},\{6,8\},\{6,9\},\{7,8\},\{7,9\},\{8,9\}\}$$

$$L_2 = \{\{1,4\},\{2,4\},\{3,4\},\{3,5\},\{4,7\},\{5,7\},\{6,7\},\{7,8\}\}$$

We find that each itemset in L_2 is a heavy itemset; so we add each of them to H. The new H is

$$H = \{\{1,2,3,7\},\{4,5\},\{1,4\},\{2,4\},\{3,4\},\{3,5\},\{4,7\},\{5,7\},\{6,7\},\{7,8\}\}$$

$$L_2 = \emptyset$$

$$k = 3$$

$$C_3 = \{\{1,2,4\},\{1,3,4\},\{1,3,5\},\{1,4,5\},\{1,4,6\},\{1,4,7\},\{1,4,8\},\{1,5,7\},\{1,6,7\},\{1,7,8\},$$
$$\{2,3,4\},\{2,3,5\},\{2,4,5\},\{2,4,6\},\{2,4,7\},\{2,4,8\},\{2,5,7\},\{2,6,7\},\{2,7,8\},\{3,4,5\},\{3,4,6\},$$
$$\{3,4,7\},\{3,4,8\},\{3,5,6\},\{3,5,7\},\{3,5,8\},\{3,6,7\},\{3,7,8\},\{4,5,6\},\{4,5,7\},\{4,5,8\},\{4,6,7\},$$
$$\{4,7,8\},\{5,6,7\},\{5,7,8\},\{6,7,8\}\}$$

$$L_3 = \{\{3,4,5\},\{1,3,4\},\{2,3,4\},\{4,5,7\}\}$$

We find that each of these is again a heavy itemset; so we add them to H and delete all their subsets already in H. The new H is

$$H = \{\{1,2,3,7\},\{3,4,5\},\{1,3,4\},\{2,3,4\},\{4,5,7\},\{6,7\},\{7,8\}\}$$

$$L_3 = \emptyset$$

$$k = 4$$

$$C_4 = \{\{1,3,4,5\},\{1,3,4,6\},\{1,3,4,7\},\{1,3,4,8\},\{2,3,4,5\},\{2,3,4,6\},\{2,3,4,7\},\{2,3,4,8\},$$
$$\{3,4,5,6\},\{3,4,5,7\},\{3,4,5,8\},\{4,5,7,8\}\}$$

$$L_4 = \emptyset$$

$$stop = 1$$

As an output of the algorithm *apriori_heavy*, we find the following heavy itemsets (note that the heavy itemsets are not disjoint):

$$H = \{\{1,2,3,7\},\{3,4,5\},\{1,3,4\},\{2,3,4\},\{4,5,7\},\{6,7\},\{7,8\}\}$$

These itemsets altogether represent 50, 12, 12, 12, 12, 2, 2 association rules respectively. Note that some of the association rules "belong to" multiple heavy itemsets; e.g., $3 \Rightarrow 4$ is represented by $\{3,4,5\}$, $\{1,3,4\}$ and $\{2,3,4\}$. Any standard algorithm for association rule mining (e.g., *apriori*) reports 92 association rules for the example in Fig. 3 for $\sigma = 33\%$, $\tau = 33\%$. We have "compactly represented" these 92 rules in terms of the six heavy itemsets, without loss of any information. Moreover, a heavy itemset represents a more meaningful relationship between the items, than in a single association rule.

There are no other rules left in this example, other than those represented by the heavy itemsets. However, in general, along with heavy itemsets, there would be other association rules as outputs of this algorithm. Note that the newly added heavy itemsets are *not* disjoint from the already known heavy itemsets $\{1,2,3,7\}$ and $\{4,5\}$.

As another example, for the following database of transactions,

1, 3, 5, 6, 8, 9
1, 2, 5, 7, 9
2, 3, 5, 6, 7, 9
1, 2, 5, 6, 8
2, 3, 4, 5, 7, 9
3, 4, 6, 8, 9
2, 3, 5, 7, 8, 9
1, 2, 4, 5, 6, 7, 9
1, 2, 3, 6, 8
1, 2, 3, 5, 7, 9

we get the following heavy itemsets

3, 9
2, 5, 7, 9

and the following association rules (for support = 50% and confidence = 65%).

support, 50.00, confidence, 83.33, 1, ⇒ ,5
support, 50.00, confidence, 83.33, 1, ⇒ ,2
support, 50.00, confidence, 71.43, 3, ⇒ ,5
support, 50.00, confidence, 71.43, 3, ⇒ ,2
support, 50.00, confidence, 71.43, 3, ⇒ ,5, 9
support, 50.00, confidence, 100.00, 3, 5, ⇒ ,9
support, 50.00, confidence, 83.33, 3, 9, ⇒ ,5
support, 50.00, confidence, 71.43, 5, 9, ⇒ ,3

Note that any standard association rule mining algorithm will report 60 rules in the above transaction database (for these support and confidence values). But *apriori_heavy* reports only 8 association rules and 2 heavy itemsets. Fifty association rules were suppressed because of the heavy itemset {2, 5, 7, 9} and 2 were suppressed because of the heavy itemset {3, 9}.

6. Experiments

We have conducted several experiments on several real life data sets. We have observed considerable reduction in the number of association rules generated by the *apriori_heavy* algorithm, as compared to the standard *apriori* algorithm. We present below some results (Table 3). We have consistently seen considerable decrease in the number of association rules reported. This decrease is also dependent on the support and confident values used; e.g., for dataset-2, we observed a 100.0% decrease when the values support = 8.0% and confidence = 60.0% were used. We have also found that the facility of only identifying the heavy itemsets is of considerable use in practice, because a heavy itemset almost always corresponds to some useful business concept.

In Table 4, we summarize the heavy itemsets identified by our algorithms in two well-known public domain datasets called BMS-WebView-1 and BMS-WebView-2 (http://www.ecn.purdue.edu/KDDCUP/) [17]. In these experiments we used *apriori_heavy* with no prior knowledge of any disjoint heavy itemsets ($H = \emptyset$). Both datasets were found to contain many heavy itemsets. Hence, when these heavy itemsets are identified and

Table 3
Experimental results

	Data-1	Data-2	Data-3
No. of transactions	209	134	22
No. of items	152,199	46	815
Min items/transaction	1	1	1
Max items/transaction	3388	17	615
Avg. items/transaction	972.87	4.1	316.27
Min transactions/item	1	11	1
Max transactions/item	15	15	17
Avg. transactions/item	1.34	11.96	8.54
Support %	5.0	8.0	60.0
Confidence %	90.0	60.0	83.0
No. of rules *apriori*	**2510**	**2692**	**62,485**
No. of rules *apriori_heavy*	**2142**	**0**	**41,593**
% Decrease in no. of rules	**14.66**	**100.0**	**33.44**
No. of heavy itemsets	**4**	**6**	**7**

Table 4
Heavy itemsets in two public domain datasets

apriori						*apriori_heavy*					
σ	τ	NF	LF	NR	T	NH	LH	NF	LF	NR	T
Dataset BMS-WebView-1											
1	10	77	2	20	7	10	2	57	1	0	26
0.9	10	90	2	28	9	14	2	62	1	0	35
0.8	10	105	2	34	10	17	2	73	1	0	43
0.7	10	133	2	48	13	24	2	91	1	0	52
0.6	10	162	3	68	15	29	3	107	1	0	68
0.5	10	201	3	121	21	35	3	128	3	7	104
0.4	10	286	3	231	35	63	3	163	3	37	212
0.3	10	435	4	481	56	106	3	235	3	184	399
0.2	10	798	4	1343	81	164	3	483	4	851	1033
Dataset BMS-WebView-2											
1	10	81	3	58	24	20	3	47	1	0	101
0.9	10	113	3	104	30	27	3	64	1	0	145
0.8	10	138	3	156	40	36	3	72	1	0	192
0.7	10	187	4	306	49	47	4	83	1	0	292
0.6	10	257	4	552	60	77	4	96	1	0	448
0.5	10	408	5	1220	94	124	5	127	1	0	818
0.4	10	676	5	2945	140	193	5	236	5	815	1521

σ, τ: the minimum support and minimum confidence, respectively.
NR, NF, NH: no. of reported association rules, frequent itemsets, heavy itemsets.
LF, LH: sizes of the largest reported frequent itemset and heavy itemset, respectively.
T: the execution time taken in seconds.

reported, the number of remaining association rules is much less than the association rules reported by the *apriori* algorithm. For example, for BMS-WebView-2, for support = 0.4% and confidence = 10%, the *apriori* algorithm reported 2945 association rules, whereas the *apriori_heavy* algorithm identified 193 heavy itemsets, thereby reducing the number of remaining association rules to 815 (72.33% reduction).

Table 4 also shows the time taken by naïve implementations of *apriori* and *apriori_heavy* on both datasets for different values of support and confidence. *Apriori_heavy* is a generalization of *apriori* to find association rules *and* heavy itemsets. Hence it is not surprising to see that *apriori_heavy* takes more time compared to *apriori*, since *apriori_heavy* is designed to report fewer association rules.

7. Conclusions and further work

Most association rule mining algorithms suffer from the twin problems of too much execution time and generating too many association rules. In this paper, we proposed a solution to address the latter problem. We proposed the concept of heavy itemset, which compactly represents an exponential number of rules. We gave an efficient theoretical characterization of a heavy itemset. Along with two simple search algorithms, we also presented an efficient greedy algorithm to generate a collection of disjoint heavy itemsets in a given transaction database. We then presented a modified *apriori* algorithm that uses given collection of heavy itemsets and detects more heavy itemsets, not necessarily disjoint with the given ones and of course the remaining association rules.

We have implemented the algorithms proposed in this paper. The heavy itemsets are a useful and informative abstraction, which is clearly understood by the end users in business terms. Typically, a heavy itemset represents a group of items, which logically belong together; e.g., as a unit or assembly. We have tried the algorithms on several real life data sets and there is a drastic reduction in the number of generated association rules, due to the use of heavy items. Thus the end users can make better sense and use of the outputs of the association rule mining algorithms. The *apriori_heavy* algorithm usually shows a substantial improvement over the performance of the *apriori* algorithm, due to its use of heavy itemsets.

As further work, we are looking for a more efficient algorithm to generate *all* heavy itemsets, disjoint or not. We are also looking into a generalization of the heavy itemset that associates a *degree of heaviness* with it. We are working on the concept of a *heavy association rule*, which is also an abstraction of a group of underlying association rules. For example, if a transaction database has *all* association rules (for a given support and confidence) of the form $L \Rightarrow R$, where $L \subseteq \{a,b,c\}$ and $R \subseteq \{p,q,r,s\}$, then *all* these rules can be compactly represented by stating that $\{a,b,c\} \Rightarrow \{p,q,r,s\}$ is a heavy association rule. Note that, in this case, $\{a,b,c,p,q,r,s\}$ *need not* be a heavy itemset. Finally, we are investigating information theoretic characterization of the heavy items and heavy association rules.

Acknowledgments

We sincerely thank Prof. Mathai Joseph for his support. We thank Dr. Sachin Lodha for many constructive suggestions. Thanks also to Dr. Gautam Sardar for useful discussions. The first author would like to thank Dr. Manasee Palshikar.

Appendix A. Proofs of theorems

Lemma 5. *Every subset of size k of the set F*1 *at the root of the tree T either* (i) *occurs as a candidate heavy itemset* CH *for some leaf in T or* (ii) *it is certain that it is not heavy. Formally, we need to prove that every subset of size m of the set F*1 *is either the* CH *for some vertex of T at level m* + 1 *or it is not heavy.*

Proof. We use induction on the level of recursion in the heavy itemset tree.

Basis: At the root (level $m = 0$), there is only one subset of $F1$ of size 0 viz., the empty set and the root is labeled with $\mathrm{CH} = \emptyset$.

Inductive step: Assume that every subset of $F1$ of size m either occurs as the label of some vertex at level m in T or it is not a heavy itemset of size m. Consider a vertex u at level m in T, having CH and $L1$ as its labels. If $|\mathrm{CH}| = m$, then $L1$ needs to contain at least $k - m$ items; if not, there is no possibility of getting a heavy itemset of size k. Hence none of the corresponding children of u are created i.e., there is no heavy itemset of size k containing CH. Note that this is really a effort-saving step (even if we did construct children of u, they will never result in a heavy itemset of size k). We now consider the case that $L1$ contains at least $k - m$ items. Clearly, the algorithm sequentially attempts to add every element I of $L1$ to CH at u. If the resulting itemset of size $m + 1$ ($\mathrm{CH} \cup \{I\}$) is not heavy then the corresponding child is not created; hence there is no heavy itemset containing $\mathrm{CH} \cup \{I\}$ as a subset and thus cannot be extended further. Thus at level m, the algorithm attempts all possible extensions of every candidate heavy itemset of size m. Thus all possible subsets of $F1$ of size $m + 1$ have been considered by the algorithm. □

Proposition 6. *Algorithm heavy*1 *terminates, returning a heavy itemset H where* $H \subseteq F1$ *and* $|H| = k$ *and returning* $\emptyset$ *if no such heavy itemset exists.*

Proof

(1) We first prove that the algorithm *heavy*1 terminates. At any particular level (iteration) of *heavy*1, as $L1$ contains a finite number of elements, the **for** loop in lines 3–12 terminates; it recursively calls *heavy*1 at most $|L1|$ times (i.e., at most once for each item I in $L1$). At any particular level and for a particular item I in $L1$, a recursive call to *heavy*1 is not made (for that item I) if the condition in line 4 is not satisfied. If it is satisfied, then we check whether $|\mathrm{CH}| + 1$ is equal to k. If yes, then the algorithm returns $\mathrm{CH} \cup \{I\}$ (without any further recursive calls). Otherwise, it makes a recursive call to *heavy*1; for this recursive call, the parameter L passed is $L1$ without the item I and parameter CH passed is CH added with item I. Note that by line 1, $L1$ is constructed (in each level of *heavy*1) so that $L1 \cap \mathrm{CH} = \emptyset$ i.e., none of the items I in $L1$ are in CH. Thus for the recursive call to *heavy*1, the size of the passed parameter CH is 1 more than the size of CH (as received at the current level). That is, the size of CH increases by 1 at each level of

*heavy*1 and will eventually reach k. The recursive call to *heavy*1 is not made if |CH|*cannot* be increased (condition in line 4 fails) or if $|\mathrm{CH} \cup \{I\}|$ has reached the given value k (which remains fixed across all recursive calls). Since |CH| increases by 1 whenever a recursive call is made, either the condition $|\mathrm{CH} \cup \{I\}| == k$ is eventually **true** or the condition in line 4 is not true for any item in $|L1|$ and there are no further recursive calls. Thus the algorithm *heavy*1 terminates.

(2) We now prove the correctness of the output. It is clear that if the algorithm returns an itemset H then H is a subset of $F1$. This follows from the observation that the "top-level" call to *heavy*1 is made with $L = F1$ and $\mathrm{CH} = \emptyset$. By the construction in line 1, the set $L1$ is always a subset of L and at each step only an item from $L1$ is added to CH.

(3) It is also easy to see that the algorithm returns an itemset H where either $H = \emptyset$ or $|H| = k$.

(4) We now prove that if the algorithm returns a non-empty itemset H then H is a heavy itemset. In line 1 it is ensured, for each item I in $L1$, the support of the set $\mathrm{CH} \cup \{I\}$ is $\geqslant$ the given value σ. Line 4 checks the necessary and sufficient condition for the itemset $\mathrm{CH} \cup \{I\}$ to be a heavy itemset. The subroutine get_confidence computes the confidence of an itemset $\mathrm{CH} \cup \{I\}$ (where $I \in L1$) as per the formula $100 * (\mathrm{support}(\mathrm{CH} \cup \{I\})/mx)$, where $mx = \max\{\text{support of single elements of } \mathrm{CH} \cup \{I\}\}$. Thus the itemset returned in line 6 satisfies the necessary and sufficient condition for a heavy itemset.

(5) We now need to prove that if the algorithm returns an empty itemset then the given database does not contain any heavy itemset of size k or more. This follows immediately from Lemma 5. □

Proposition 7. *Algorithm find_disjoint_heavy_itemsets_simplified terminates, returning a collection S containing disjoint heavy itemsets. Moreover, the algorithm finds and reports one of the largest itemsets i.e., if the algorithm reports the largest heavy itemset of size say k, then there is no heavy itemsets of size k + 1.*

Proof. We prove first that the algorithm *find_disjoint_heavy_itemsets_simplified* terminates. The outer **for** loop clearly terminates. The inner **while** loop calls the subroutine *heavy*1, which terminates, by Proposition 6, and returns a heavy itemset H which either has k elements (from the given set $F1$) or is empty. If the returned H is non-empty, the corresponding elements of H are removed from $F1$ and *heavy*1 is called again. Since the number of heavy itemsets in a finite database is finite, and since *heavy*1 returns only that heavy itemset which is a subset of given $F1$, we can say that eventually, there will not be any heavy itemset from given $F1$ (*heavy*1 will return $H = \emptyset$) or that $F1$ will be empty. In either case, the inner **while** loop will terminate.

Clearly, every itemset added to S is a heavy itemset returned by *heavy*1. That all itemsets in S are disjoint follows from the fact that once a heavy itemset H is returned by *heavy*1, the items in H are removed from $F1$ before the next call to *heavy*1.

Suppose the largest heavy itemset H_0 in the returned set S contains m elements. The index k for the outer **for** loop starts at the largest possible size for a heavy itemset and is decremented in each iteration. Hence, *heavy*1 must have returned H_0 when it was called with $k = m$ (and *heavy*1 must have returned $H = \emptyset$ for all values of $k > m$). Since *heavy*1 returned $H = H_0$ (of size m) when $k = m$ and returned $H = \emptyset$ for all values of $k > m$, it follows (by Proposition 6) that there is no heavy itemset of size $> m$ in the given database. □

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining associations between sets of items in massive databases, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD 1993), pp. 207–216.

[2] S. Brin, R. Motwani, J. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD 1997), 1997, pp. 255–264.

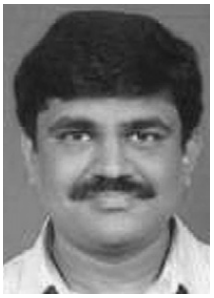
[3] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J.D. Ullman, C. Yang, Finding interesting associations without support pruning, Knowledge Data Eng. 13 (1) (2001) 64–78.

[4] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD 2000), 2000, pp. 1–12.

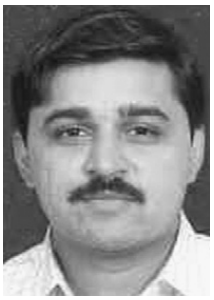
[5] R.J. Hilderman, H.J. Hamilton, Knowledge discovery and interestingness measures: a survey, Tech. Report CS-99-04, Dept. of Computer Science, Univ. of Regina, 1999.

[6] R.M. Karp, S. Shenker, C.H. Papadimitrious, A simple algorithm for finding frequent elements in streams and bags, ACM Trans. Database Syst. 28 (1) (2003) 51–55.

[7] D. Lin, Z.M. Kedem, Pincer-search: an efficient algorithm for discovering the maximum frequent set, IEEE Trans. Knowledge Data Eng. 14 (3) (2002) 553–556.
[8] B. Liu, W. Hsu, Y. Ma, Pruning and summarizing the discovered associations, in: Proc. 5th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, New York, 1999, pp. 125–134.
[9] G.K. Palshikar, M.S. Kale, M.M. Apte, Association rule mining using heavy itemsets, in: Proc. Int. Conf. Management of Data (COMAD 2005), Goa, India, 6–8 January 2005, pp. 148–155.
[10] A.K. Pujari, Data Mining Techniques, University Press, 2001.
[11] S. Ramaswamy, S. Mahajan, A. Silberschatz, On the discovery of interesting patterns in association rules, in: Proc. 24th Int. Conf. Very Large Data Bases (VLDB 1998), 1998, pp. 368–379.
[12] P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, D. Shah, Turbo-charging vertical mining of large databases, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD 2000), pp. 22–33.
[13] P.J. Soo, C. Ming-Syan, P.S. Yu, Using a Hash-based method with transaction trimming for mining association rules, IEEE Trans. Knowledge Data Eng. 9 (5) (1997) 813–825.
[14] H. Tiovonen, Sampling large databases for association rules, in: Proc. 22nd Int. Conf. Very Large Databases (VLDB 1996), 1996, pp. 134–145.
[15] M.J. Zaki, Generating non-redundant association rules, in: Proc. 6th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2000), 2000, pp. 34–43.
[16] M.J. Zaki, C. Hsiao, Charm: an efficient algorithm for closed itemset mining, in: Proc. SIAM International Conference on Data Mining, 2002.
[17] Z. Zheng, R. Kohavi, L. Mason, Real world performance of association rule algorithms, in: Proc. 7th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2001), 2001, pp. 401–406.



Girish Keshav Palshikar obtained an M.Sc. (Physics) from Indian Institute of Technology, Bombay in 1985 and an M.S. (Computer Science and Engineering) from Indian Institute of Technology, Chennai in 1988. He is a scientist at the Tata Research Development and Design Centre (TRDDC), Pune, India, since 1992. He has several research publications in international journals and conferences. His areas of research include machine learning and theoretical computer science.



Mandar S. Kale received the B.E. degree in Production Engineering from Kolhapur University, India in 1997 and M.Tech. degree in Industrial Engineering from V.R.C.E. Nagpur, Nagpur University in 2000. Currently he is working with Research and Development Group for Engineering and Industrial Services at Tata Consultancy Services. His research interests include Sequence Mining and application of data mining techniques for supply chain optimisation.

Manoj M. Apte received the B.E. degree in Industrial Engineering from University of Pune, India in 1994 and M. Tech. degree in Industrial Management from IIT Madras, in 1997. He is working with Tata Consultancy Services after completing his Masters and currently working with Research and Development Group for Engineering and Industrial Services at Tata Consultancy Services. His research interests include Association Rule Mining, Clustering and application of data mining to problems in Industry.