Efficient Incremental Mining of Top-K Frequent Closed Itemsets*

Andrea Pietracaprina and Fabio Vandin

Dipartimento di Ingegneria dell'Informazione, Università di Padova, Via Gradenigo 6/B, 35131, Padova, Italy. E-mails: {capri,vandinfa}@dei.unipd.it

Abstract. In this work we study the mining of top-K frequent closed itemsets, a recently proposed alternative to the classical frequent itemset mining problem, whose purpose is to provide better control on the output size by implicitly fixing the support threshold as the maximum value which yields at least K frequent closed itemsets. We first discuss the effectiveness of parameter K in controlling the output size. Then, we develop an efficient algorithm for mining top-K frequent closed itemsets in order of decreasing support, which exhibits consistently better performance than the best previously known one, attaining substantial improvements in some cases. A distinctive feature of our algorithm is that it allows the user to dynamically raise the value K with no need to restart the computation from scratch.

1 Introduction

The discovery of frequent itemsets is a fundamental primitive which arises in the mining of association rules and in many other mining problems. In its original formulation [1], the problem requires that given a dataset $\mathcal{D}$ of transactions over a set of items $\mathcal{I}$, and a support threshold σ, all frequent itemsets $X \subseteq \mathcal{I}$ be discovered and returned in output, where an itemset is considered frequent if it has *support* at least σ (i.e., it is contained in at least σ transactions). As well known, a challenging aspect of this formulation is related to the difficulty of predicting the actual number of frequent itemsets for a given dataset $\mathcal{D}$ and support threshold σ. Indeed, in some cases a small value of σ can yield a number of frequent itemsets exponential in the dataset size, while a large value may yield very few or no frequent itemsets.

In order to reduce the output size, a number of variants of the classical problem have been proposed, such as, for example, the mining of *maximal frequent itemsets* [2], that is, frequent itemsets that have no frequent supersets, or the mining of *frequent closed itemsets* [9], that is, frequent itemsets that have no supersets with the same support. Although in practice frequent closed itemsets and maximal itemsets are substantially less than all frequent itemsets, it has been shown that the number of these irredundant itemsets can still be exponential in the dataset size [16]. Hence, the choice of a suitable support threshold σ remains challenging even in these cases. In [7, 6] many state-of-art algorithms for these problems can be found.

Recently, an elegant variation of the original mining task has been proposed in [15] aiming at providing the user better control on the size of the output set. Specifically, this variation requires the discovery of the *top-K frequent closed itemsets*, that is, all closed itemsets of support at least σ_K, where K is an input parameter and σ_K, which is uniquely defined by K,

* This work was supported in part by MIUR of Italy under project MAINSTREAM.

is the maximum support threshold that yields at least K frequent closed itemsets. Although one is not guaranteed that top-K frequent closed itemsets are exactly K, it is conceivable that parameter K be more effective than the minimum support threshold in controlling the output size. It is important to remark that the top-K frequent closed itemsets can be employed in every application where frequent closed itemsets are needed.

In [15] the authors present an efficient algorithm, called TFP, for mining the top-K frequent closed itemsets. The main idea of the algorithm is to use an efficient depth-first mining process starting with an initially low support threshold σ which is progressively increased, as the execution proceeds, by means of several effective heuristics, until the final value σ_K is reached. When an itemset is generated it is inserted into a suitable data structure from which it can later be removed and discarded if found to be non-closed or infrequent. TFP has an additional feature which allows the user to specify a minimum length $\min_\ell$ for the closed itemsets to be returned. The authors provide experimental evidence of the efficiency of their algorithm. The main drawbacks of TFP are that no bound is given on the number of non-closed or infrequent itemsets that the algorithm must process, and that an involved itemset closure checking scheme is required. Moreover, TFP does not appear to be able to handle efficiently a dynamic scenario where the user is allowed to raise the value K.

The enumeration of closed itemsets in order of decreasing support is considered in [3], although no explicit algorithm is provided to implement this approach. The authors also show that the enumeration can be accomplished in polynomial incremental time. In [11] the mining the K itemsets of maximum density with respect to a fixed support threshold is studied, where the notion of density relaxes the requirement of strict containment of an itemset in its supporting transactions. The authors propose a priority-queue based approach for solving this problem, similar in spirit to the one adopted in our paper. Other results concerning somewhat related problems can be found in [12, 5, 4].

The mining of top-K frequent closed itemsets is the focus of this paper. We first discuss the effectiveness of parameter K in controlling the output size. As a corollary of a result in [3], we show that the number of closed itemsets returned in output is at most nK, where n is the number of items, and prove that such a bound is tight.

Then, we present a new algorithm, TopKMiner, for mining top-K frequent closed itemsets in order of decreasing support. The implementation of this strategy is challenging since during the mining process the frontier of explored itemsets may grow large, although somewhat controlled by parameter K. To face this challenge, TopKMiner, which uses a priority queue to store the frontier of explored itemsets, features a compact representation of the conditional datasets of the itemsets in the queue, combined with the clever method proposed in [14] for exploring only closed itemsets. In this fashion, TopKMiner, unlike algorithm TFP [15], is able to guarantee a provable bound on the number of itemsets touched during the mining process, and, moreover, it allows the user to dynamically raise the value K efficiently, without the need to restart the computation from scratch. Results of several experiments conducted to compare the performance of TopKMiner and TFP on both real and synthetic datasets, show that TopKMiner consistently exhibits better performance, with substantial improvements in some cases (more than two orders of magnitude). The efficiency of TopKMiner becomes considerably higher when used in a dynamic scenario where top-K frequent closed itemsets are sought for increasing values of K successively provided by the user.

The rest of the paper is organized as follows. Section 2 defines the problem and introduces a number of concepts and notations that will be used thereafter. The effectiveness of parameter K in controlling the output size is discussed Section 3. The algorithm TopKMiner

is described in Section 4, while the results of the experimental comparison between TFP and TopKMiner are reported and discussed in Section 5.

2 Preliminaries

Let $\mathcal{I}$ be a set of *items*, and $\mathcal{D}$ a (multi)set of *transactions*, where each transaction $t \in \mathcal{D}$ is a subset of $\mathcal{I}$. We use $||\mathcal{D}||$ to denote the dataset size, that is, $\sum_{t \in \mathcal{D}} |t|$. For an *itemset* $X \subseteq \mathcal{I}$ we define its *conditional dataset* $\mathcal{D}_X \subseteq \mathcal{D}$ as the (multi)set of transactions $t \in \mathcal{D}$ that contain X, and define the *support of* X *w.r.t.* $\mathcal{D}$, $\text{supp}_{\mathcal{D}}(X)$ for short, as the number of transactions in $\mathcal{D}_X$. An itemset X is *closed w.r.t.* $\mathcal{D}$ if there exists no itemset Y, with $X \subset Y \subseteq \mathcal{I}$, such that $\text{supp}_{\mathcal{D}}(Y) = \text{supp}_{\mathcal{D}}(X)$. In other words, if X is closed, then adding a single item to X decreases its support. For any itemset X, its *closure w.r.t.* $\mathcal{D}$, denoted by $\text{Clo}_{\mathcal{D}}(X)$, is the closed itemset $Y \supseteq X$ such that $Y = \bigcap_{t \in \mathcal{D}_X} t$[1].

The classical mining problem requires to discover all itemsets of support greater than or equal to a fixed threshold σ, which are referred to as *frequent itemsets* and denoted by the set $\mathcal{F}(\mathcal{D}, \sigma)$. In order to reduce redundancy in the output, the mining is often limited to the discovery of all *frequent closed itemsets* (set $\mathcal{FC}(\mathcal{D}, \sigma) \subseteq \mathcal{F}(\mathcal{D}, \sigma)$). It is easy to see that from the frequent closed itemsets and their supports, all frequent itemsets and their supports can be derived. Although the number of frequent closed itemsets is often much smaller than the number of all frequent itemsets, there are cases when $|\mathcal{FC}(\mathcal{D}, \sigma)|$ is still exponential in $||\mathcal{D}||$ [16].

As mentioned in the introduction, for a given dataset $\mathcal{D}$ and support threshold σ, it is hard to predict the $|\mathcal{F}(\mathcal{D}, \sigma)|$ or $|\mathcal{FC}(\mathcal{D}, \sigma)|$, and this is a problematic aspect of the classical frequent (closed) itemset mining task. Setting σ too large may exclude interesting itemsets from the output, while setting it too small may yield an impractically large output set. To overcome this problem, in [15] the authors propose to modify the mining task into that of discovering the *top-K frequent closed itemsets*, defined below.

Definition 1. *For a dataset $\mathcal{D}$ and an integer K, define the set of* top-K frequent closed itemsets *(*top-K f.c.i.*, for short) as $\mathcal{FC}_K(\mathcal{D}) = \mathcal{FC}(\mathcal{D}, \sigma_K)$, where σ_K is the maximum value such that $|\mathcal{FC}(\mathcal{D}, \sigma_K)| \geq K$.*

The top-5 frequent closed itemsets for a sample dataset $\mathcal{D}$ are shown in Figure 1. Note that when mining the top-K frequent closed itemsets the threshold σ_K is not given as part of the input and it is uniquely, although implicitly, defined as a function of K, which sets a more direct constraint on the output size. However, requiring the discovery of all closed itemsets of support at least σ_K may yield many itemsets (of support equal to σ_K) in excess of K, but these extra itemsets are necessary in case other patterns (e.g., association rules) must be derived from the frequent closed itemsets.

Since the number of frequent itemsets can be much larger than the number of frequent closed itemsets, when mining the latter it is convenient to avoid processing non-closed itemsets. To this aim, in [14] the authors propose a conceptual organization of the closed itemsets as nodes of a tree, with support decreasing with increasing depth. Specifically, let $\mathcal{D}$ be a dataset defined over the set of items $\mathcal{I} = \{a_1, a_2, \ldots, a_n\}$ (the indexing of the items is fixed but arbitrary). For an itemset X define its *j-th prefix* as $X(j) = X \cap \{a_i \ : \ 1 \leq i \leq j\}$,

[1] For simplicity, in what follows the terms support, closed itemset, and closure will be used without explicit reference to $\mathcal{D}$, when $\mathcal{D}$ is clear from the context.

$\mathcal{D}$	$\mathcal{I}$					
t_1	a_6		a_4			a_1
t_2	a_6		a_4			
t_3	a_6	a_5	a_4	a_3	a_2	
t_4	a_6	a_5				
t_5		a_5		a_3	a_2	
t_6		a_5		a_3		a_1
t_7	a_6	a_5	a_4	a_3	a_2	a_1

(a)

X	supp(X)
a_6	5
a_5	5
a_6 a_4	4
a_5 a_3	4
a_6 a_5	3
a_5 a_3 a_2	3
a_1	3

(b)

Fig. 1. (a) Sample dataset $\mathcal{D}$; (b) $\mathcal{FC}_5(\mathcal{D})$

for $1 \leq j \leq n$. The *core index* of a closed itemset X, denoted as $\text{core}_i(X)$, is defined as the minimum j such that $\mathcal{D}_X = \mathcal{D}_{X(j)}$.

Definition 2 ([14]). *A closed itemset X is a* prefix-preserving closure extension (ppc-extension) *of a closed itemset Y if: (1) $X = Clo_{\mathcal{D}}(Y \cup \{a_j\})$, for some $a_j \notin Y$ with $j > core_i(X)$; and (2) $X(j-1) = Y(j-1)$.*

Let $\perp = \text{Clo}_{\mathcal{D}}(\emptyset)$, which is the possibly empty closed itemset consisting of the items occurring in all transactions. The following theorem defines the tree structure over the set of closed itemsets, with $\perp$ being the root of the tree.

Theorem 1 ([14]). *Any closed itemset $X \neq \perp$ is the ppc-extension of exactly one closed itemset Y, and $supp(X) < supp(Y)$.*

A similar tree spanning all closed itemsets was independently discovered in [8]

3 Effectiveness of K in controlling the output size

Let $\Delta(n)$ be the family of all datasets $\mathcal{D}$ whose transactions comprise n distinct items. Define $\rho(n, K) = \max_{\mathcal{D} \in \Delta(n)}(|\mathcal{FC}_K(\mathcal{D})|/K)$, which provides a worst-case estimation of the deviation of the output size from K when mining top-K closed frequent itemsets. We have:

Theorem 2. *For every $n \geq 1$ and $K \geq 1$, we have $\rho(n, K) \leq n$. Moreover, for every $n \geq 1$ and every constant c, there are $\Omega(n^c)$ distinct values of K such that $\rho(n, K) \in \Omega(n)$.*

Proof (sketch). Consider an arbitrary dataset $\mathcal{D} \in \Delta(n)$ and a value $K \geq 1$. Let $\Phi = \{X_1, X_2, \ldots, X_K\}$ be the set of K most frequent non-empty closed itemsets numbered in decreasing order of support and let $\perp = \text{Clo}_{\mathcal{D}}(\emptyset)$. By Theorem 1 we know that any closed itemset $X \notin \Phi$ of support σ_K must be a ppc-extension of some closed itemset $Y \in (\Phi \backslash \{X_K\}) \cup \perp$. The upper bound on $\rho(n, K)$ follows directly from the argument in [3] which shows that any such itemset Y can generate at most $(n-1)$ ppc-extensions not belonging to Φ. Hence, the number of closed itemsets not included in Φ and of support σ_K is at most $K(n-1)$, which yields $\rho(n, K) \leq n$. The lower bound on $\rho(n, K)$ is provided by the dataset described in [16, Section 3.1]. In particular, that dataset shows that $\rho(n, 1) = n$.

Algorithm 1: **TopKMiner**

Input: Dataset $\mathcal{D}$, max value K^* for K
Output: Top-K f.c.i for any $K \leq K^*$ provided by the user

```
1  K ← input from user; /* K ≤ K* */
2  Initialize σ as a lower bound to σ_{K*}; σ' ← σ;
3  Q ← empty priority queue; extracted ← 0;
4  Compute ⊥= Clo_D(∅);
5  if ⊥≠ ∅ then {Output ⊥; extracted++; if K = 1 then σ' = |D|};
6  for each ppc-extension Y of ⊥ of support s ≥ σ do
7    Q.insert((D_Y, s, core_i(Y), Y(core_i(Y) − 1)));
8  while (Q ≠ ∅) and (Q.max() ≥ σ') do
9    (D_Y, s, i, Y(i − 1)) ← Q.removeMax();
10   extracted++; if extracted = K then σ' = s;
11   Generate and output closed itemset Y;
12   if s > σ then
13     for j ← i + 1 to n do /* Denote X = Clo_D(Y ∪ {j}) */
14       Compute X(j − 1), s' = supp_D(X), and D_X;
15       if X(j − 1) = Y(j − 1) and s' ≥ σ then
16         Q.insert(D_X, s', j, X(j − 1)) ;
17         if extracted+|Q| ≥ K* then
18           Update σ and remove from Q all entries of support < σ;
19 if user wants to raise K then
20   K ← new input from user;
21   if K > extracted then σ' ← σ;
22   goto line 8;
```

Fig. 2. Algorithm TopKMiner: pseudocode

We note that in practice the ratio $|\mathcal{FC}_K(\mathcal{D})|/K$ is much smaller than n. On several real and synthetic datasets we tested for values of K between 100 and 10000, the ratio is always very close to 1. In fact, it can be shown that $\rho(n, K) = n$ is attained only for $K = 1$ and we conjecture that $\rho(n, K)$ be a decreasing function of K (more details will be reported in the full version of this paper). Finally, we remark that the bounds of Theorem 2 crucially rely on the fact that the mining task is limited to closed itemsets. Indeed, if we lift the closedness requirement and mine the top-K frequent itemsets, the ratio between the number of itemsets returned in output and the value K can be exponentially large in n. A non-trivial dataset where this happens is shown in [14, Theorem 1].

4 TopKMiner

In the following subsections we describe our algorithm *TopKMiner*. Subsection 4.1 introduces the algorithm's high-level strategy, while Subsection 4.2 describes a number of crucial implementation details. In the description of the algorithm we let $\mathcal{I} = \{a_1, a_2, \ldots, a_n\}$ denote the set of items and assume that they are ordered by non-decreasing support.

4.1 Main strategy

TopKMiner, whose pseudocode is given in Figure 2, is based on the following conceptually simple strategy. The algorithm receives in input the dataset $\mathcal{D}$ and a value K^* that represents

the maximum K for which the user may request the mining of top-K frequent closed itemsets. The value K is provided by the user at the beginning of the algorithm. Two variables σ and σ' are used to store dynamic approximations from below to σ_{K^*} and σ_K, respectively. Variable σ can be initially set using the same heuristics as in [15]. Instead, σ' is initially set equal to σ, and is raised to the final value σ_K as soon as the K-th frequent closed itemset is discovered. The algorithm makes use of a priority queue Q whose entries correspond to closed itemsets. An entry for a closed itemset Y is a quadruple $(\mathcal{D}_Y, s, i, Y(i-1))$, where $\mathcal{D}_Y$ is the conditional dataset for Y, s its support (and the key for the entry), i its core index, and $Y(i-1)$ its $(i-1)$-st prefix. Q is initialized with the ppc-extensions of $\perp = \mathrm{Clo}_{\mathcal{D}}(\emptyset)$. Then a main loop is executed (lines $8 \div 18$), where in each iteration the entry $(\mathcal{D}_Y, s, i, Y(i-1))$ with maximum support s is extracted from Q, the itemset Y is generated and returned in output, and entries for all ppc-extensions of Y with support $s' \geq \sigma$ are inserted into Q. Once K^* closed itemsets have been seen, after every insertion into Q σ is updated by setting it to the support of the K^*-th most frequent itemset seen so far, and all entries in Q corresponding to infrequent itemsets are removed from Q. The loop ends when all top-K frequent closed itemsets have been generated or Q becomes empty. Finally (lines $19 \div 22$) if the user raises K to a new value $K_{\mathrm{new}} \leq K^*$, and more closed itemsets need to be discovered, the main loop is started again resetting σ' equal to σ as a lower bound to $\sigma_{K_{\mathrm{new}}}$.

Note that an entry $(\mathcal{D}_Y, s, i, Y(i-1))$ in Q for a closed itemset Y does not contain Y itself but only sufficient information to generate the itemset. The actual generation of Y, which is a time-consuming task, is done only when strictly necessary, that is, when the entry $(\mathcal{D}_Y, s, i, Y(i-1))$ is extracted from Q and Y is guaranteed to belong to the output set. In fact, as it will be shown in the following subsection, entries for all ppc-extensions of Y to insert into Q can be produced efficiently without generating the corresponding itemsets.

We remark that the upper limit K^* on the value K is not needed for correctness, but it is useful to provide a bound on the maximum number of entries inserted into the priority queue Q. Indeed, it can be argued that for a dataset $\mathcal{D}$ over a set of n items, algorithm TopKMiner will insert a total of at most nK^* entries into Q during the entire course of the computation. In fact, with a slightly modification of the algorithm it is possible to reduce this bound to $nK_{\max}$, where $K_{\max} \leq K^*$ is the maximum K requested by the user. No bound is known on the number of itemsets processed by algorithm TFP [15]. Moreover, unlike TFP, TopKMiner allows the user to dynamically raise K without the need to restart the computation from scratch.

4.2 Implementation details

The efficient implementation of TopKMiner is challenging and non-trivial. For lack of space, we will describe here only a few crucial aspects of our implementation. As in [10] the dataset $\mathcal{D}$ is represented through a Patricia trie $T_{\mathcal{D}}$ built on the set of transactions regarded as strings of items, with items sorted by decreasing support. The Patricia trie $T_{\mathcal{D}}$ for the sample dataset of Figure 1 is shown in Figure 3. Each node v is identified by a unique id (shown in a circle) and stores a count that indicates how many transactions of $\mathcal{D}$ contain the itemset encountered along the path from v to the root.

The priority queue employed by TopKMiner is implemented as a standard max-heap vector. As introduced before, an entry, corresponding to some closed itemset Y, consists of the quadruple $(\mathcal{D}_Y, s, i, Y(i-1))$. While the last three components are stored in a natural way, a suitable representation of $\mathcal{D}_Y$ is required for both space and time efficiency. We represent $\mathcal{D}_Y$ through a list $L_{\mathcal{D}}(Y)$ of nodes of $T_{\mathcal{D}}$ such that a node v is included in $L_{\mathcal{D}}(Y)$

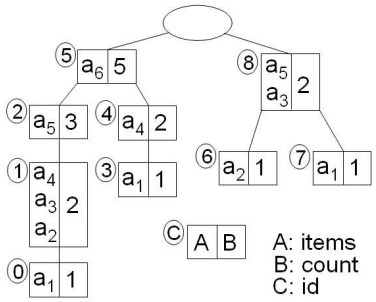


Fig. 3. Patricia trie for the sample dataset of Figure 1

if and only if v contains the core index item a_i of Y and belongs to a path associated with one or more transactions in $\mathcal{D}_Y$. Let $\mathcal{D}_{Y,v}$ denote the (multi)set of transactions in $\mathcal{D}_Y$ whose associated paths in $T_\mathcal{D}$ contain the node v, and let $Z_{Y,v} = \bigcap_{t \in \mathcal{D}_{Y,v}} t$. In the list $L_\mathcal{D}(Y)$, together with each node v, we store the prefix $Z_{Y,v}(i-1)$, that is the intersection of all transactions in $\mathcal{D}_{Y,v}$ limited to the items of index less than i. Such a prefix turn out to be useful in the implementation of the while loop described next. Moreover we associate with every node v the number $s_{Y,v}$ of transactions in $\mathcal{D}_Y$ which share this node, that is $s_{Y,v} = |\mathcal{D}_{Y,v}|$. For very large and sparse datasets, the list $L_\mathcal{D}(Y)$ may be very long. If its length exceeds a certain fixed threshold (5MB in our experiments) the list is stored on disk rather than in main memory. In this fashion we can considerably reduce the amount of main memory required by the algorithm.

Consider an arbitrary iteration of the while loop (lines $8 \div 18$) where entry $(\mathcal{D}_Y, s, i, Y(i-1))$ is extracted from Q. All of the operations prescribed by the iteration can be executed through a simple bottom-up traversal of the sub-trie T' of $T_\mathcal{D}$, whose leaves are the nodes in the list $L_\mathcal{D}(Y)$ which represents $\mathcal{D}_Y$. The purpose of the traversal is to fill the rows of a *header table* HT, whose j-th row, denoted by HT$[j]$, is associated with item a_j and contains a record with three fields: HT$[j]$.supp, HT$[j]$.pref, and HT$[j]$.list (the contents of these fields will be described below). By using a strategy similar to the one introduced in [10], the subtrie T' can be traversed in such a way to process each node only once. Let $X^{(j)}$ denote the itemset $\mathrm{Clo}_\mathcal{D}(Y \cup \{j\})$. During the traversal of T', by percolating upwards the prefixes $Z_{Y,v}(i-1)$ initially stored with the leaves of T' we can update the header table so that, at the end of the traversal, for every $j > i$ we have that: HT$[j]$.supp $= \mathrm{supp}_\mathcal{D}(\mathrm{Clo}_\mathcal{D}(Y \cup \{j\}))$; HT$[j]$.pref $= X^{(j)}(j-1)$; and HT$[j]$.list is the head of the list of all nodes of T' containing item a_j. With each node v in HT$[j]$.list we store the count $s_{X^{(j)},v}$ and the prefix $Z_{X^{(j)},v}(j-1)$.

In Figure 4 the HT filled after a traversal is shown for sample dataset of Figure 1. It is easy to see that once the header table is filled as described above, the information stored in its rows is sufficient to fully compute the itemset Y, and to identify each ppc-extension X of Y determining also its support s', its core index j, its prefix $X(j-1)$ and the representation $L_\mathcal{D}(X)$ of its conditional dataset. We observe that, at this point, determining for each ppc-extension X of Y all of its constituent items would require an extra non-trivial computation which would be useless in case X turn out not to belong to the output set. For this reason, TopKMiner postpones the actual determination of a closed itemset X to the time when the entry corresponding to X is extracted from Q, hence ensuring that X belong to the output set.

j	**supp**	**pref**	**list**
6	2	$a_4\ a_1$	<5, 2,{$a_4\ a_1$}>
5	2	$a_3\ a_1$	<2, 1,{$a_4\ a_3\ a_2\ a_1$}> <8, 1, {$a_3\ a_1$}>
4	2	a_1	<1, 1,{$a_3\ a_2\ a_1$}> <4, 1,{a_1}>
3	2	a_1	<1, 1,{$a_2\ a_1$}> <8, 1,{a_1}>
2	1	a_1	<1, 1,{a_1}>
1	-	-	-

Fig. 4. HT at the end of the traversal of the Patricia trie of Figure 3, starting from nodes of $L_{\mathcal{D}}(\{a_1\})$, namely the nodes with id's 0, 3 and 7. For every j and every node v in HT[j].list, we show within angular brackets its id, $s_{X^{(j)},v}$, and $Z_{X^{(j)},v}(j-1)$

5 Experimental evaluation

We experimentally evaluated the performance of TopKMiner on both real and synthetic datasets from the FIMI repository (`http://fimi.cs.helsinki.fi`). In this section we report the results for a representative sample of the datasets, whose characteristics are illustrated in Figure 5. The results for the other datasets are consistent with those reported here.

Dataset	#Items	Avg. Trans. Length	# Transactions
T40	1,000	39.5	100,000
accidents	468	33.8	340,183
pos	1,658	7.5	515,597
kosarac	41,270	8.1	990,002
webdocs	5,267,656	177	1,692,082

Fig. 5. Datasets characteristics

The experiments have been conducted on a HP Proliant, using one AMD Opteron 2.2GHz processor, with 8GB main memory, 64KB L1 cache and 1MB L2 cache. The main objective of the experiments has been to compare the performance of TopKMiner with that of TFP [15], which is, to the best of our knowledge, the only known algorithm for mining the top-K frequent closed itemsets. Both TopKMiner and TFP have been coded in `C++` and the source code for TFP has been provided to us by its authors. It must be recalled that TFP has an additional feature which enables the mining of the top-K frequent closed itemsets of length greater than or equal to a minimum value $\min_\ell$ specified in input. We did not implement a similar feature in TopKMiner, hence in the experiments TFP has always been executed with $\min_\ell = 1$.

5.1 Comparison between TFP and TopKMiner without dynamic raising of K

We run both TopKMiner and TFP for values of K ranging from 1000 to 10000 with step 1000. For TopKMiner, we imposed $K = K^*$ so to assess the relative performance of the two algorithms when focused on the basic task of mining top-K frequent closed itemsets. The running times exhibited by the two algorithms on the datasets of Figure 5 (except webdocs)

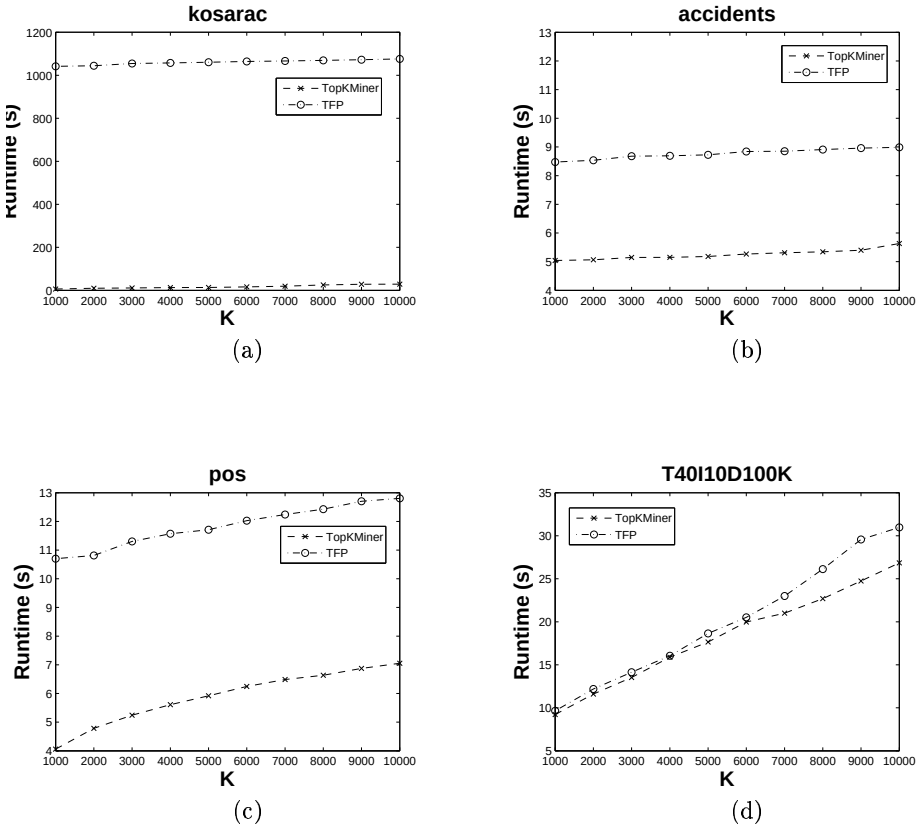


Fig. 6. Running times of TopKMiner and TFP for (a) kosarac, (b) accidents, (c) pos, and (d) T40I10D100K for various values of K

are shown in Figure 6. It can be seen that TopKMiner runs always faster than TFP, with a performance improvement of more than two orders of magnitude for kosarac. One reason that explains the superior performance of TopKMiner is that it processes exclusively closed itemsets and generates in full only those itemsets that surely belong to the output set, unlike TFP which may happen to process and generate non-closed and/or infrequent itemsets. Indeed, in the above experiments we discovered that TFP generates, without being able to discard immediately, almost twice the number of itemsets that TopKMiner processes (i.e., inserts into the priority queue). This discrepancy can grow up to a factor proportional to the number of items in certain extreme cases such as the artificial dataset defined in [16, Section 3.1]. Moreover, TFP is penalized by the involved yet necessary closure-checking mechanism.

For dataset webdocs, TFP aborted after a few hours of execution even for $K = 100$ and not because of memory problems. Thus, we compared the running time achieved by TopKMiner with the one achieved by algorithm LCM, [13], one of the best algorithms at the FIMI'03 competition for mining frequent closed itemsets, feeding LCM with the

exact support threshold. As shown in Figure 7, TopKMiner surprisingly achieved better performance. In this case, because of the large size of the dataset, it has been crucial for TopKMiner to use external memory to store the conditional dataset representations.

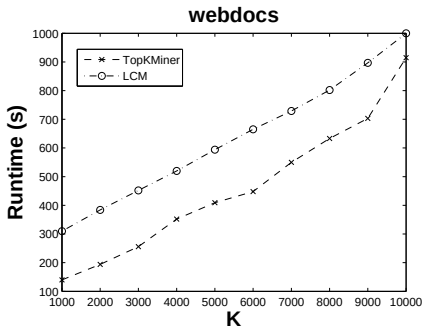


Fig. 7. Running times of TopKMiner and LCM on webdocs, for various values of K

We also compared the memory usage of TFP and TopKMiner on all datasets, except for webdocs. While TFP adopts a depth-first mining strategy, which is known to be generally space-efficient, TopKMiner employs a support-driven exploration which may require more space due to the need to store the frontier of explored closed itemset. However, for not too large values of K the actual number of itemsets that TopKMiner must concurrently maintain in the queue is somewhat limited. For K between 1000 and 10000, TopKMiner requires less memory than TFP in all cases except for T40I10D100K with $K > 5000$ for which it requires, in the worst case, a factor 1.5 more memory.

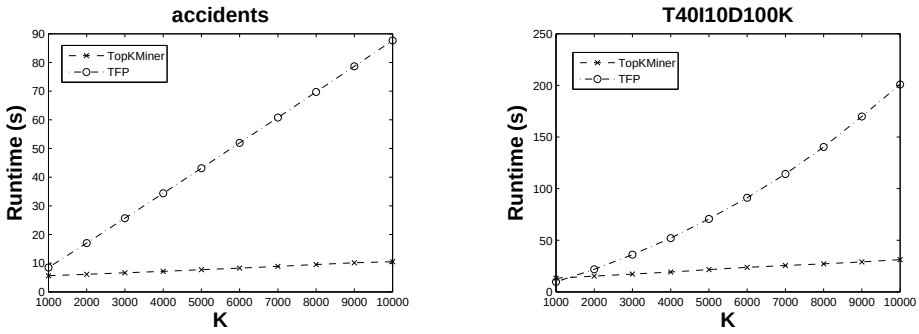
The high memory usage exhibited by TFP can be in part accounted for by the conditional datasets that it creates during execution, while the lower memory usage exhibited by TopKMiner in several cases is due to the efficient representation chosen for the priority queue entries. We remark that although the machine we used for the experiments features a very large RAM (8 GB), in all of the experiments (except for webdocs) the actual total RAM required never exceeded 450 MB, which is a reasonable quantity even for a low-end PC. Moreover, in these experiments TopKMiner did not resort to disk for storing the conditional datasets, hence its memory usage would have considerably improved by externalizing the conditional datasets, slightly increasing the running time. As for webdocs, TopKMiner used between 1.4GB to 6.5GB of main memory, but in this case the dataset itself occupies about 1.5GB.

5.2 Comparison between TFP and TopKMiner with dynamic raising of K

In a final set of experiments, we tested the effectiveness of the TopKMiner's feature which allows the user to dynamically raise the value K up to a maximum value K^*. To this purpose we simulated a scenario where K is raised from 1000 to 10000 with step 1000 and run TopKMiner with $K^* = 10000$ measuring the running time after the computation for each value K ended. We compared these running times with those attainable by TFP if used

in a similar scenario, by running, for each K, the algorithm from scratch and accumulating the running times of previous executions. The results are shown in Figure 8 only for two datasets (the time for the user's input is not accounted for). Results for the other datasets are similar and are omitted for brevity.

As expected, the time required by TopKMiner for each value of K is considerably lower than the cumulative time required by TFP, which is a clear evidence of the effectiveness of TopKMiner dynamic feature. Moreover, we remark that the provision of such a feature adds only a negligible slowdown. Indeed, even if the computation is stopped after the first value $K = 1000$, TopKMiner remains still faster than TFP with the exception of the T40I10D100K dataset, where, however, the difference between the two algorithms is very small. This means that the flexibility of TopKMiner (in the raising of K) results in better performance compared with TFP, even if the potential of this flexibility is not fully exploited.



(a) (b)

Fig. 8. Running times of TopKMiner and TFP for (a) accidents, and (b) T40I10D100K with dynamic update of K from 1000 to 10000.

6 Acknowledgments

The authors gratefully acknowledge the authors of TFP for providing the source code, and Geppino Pucci for fruitful discussions.

7 Conclusions

We studied the mining of top-K frequent closed itemsets. We discussed the effectiveness of K in controlling the output size and developed an efficient algorithm, TopKMiner for this problem. The algorithm incrementally generates the closed itemsets in order of decreasing support and allows the user to dynamically raise the value K without the need to restart the computation from scratch. The experimental analysis has shown the superior performance

of TopKMiner compared to the one of the best previously known algorithm. A number of interesting problems remain open. It would be useful to prove or disprove our conjecture that as K increases, the bound on the ratio between the output size and K (maximized over all datasets on n items) decreases. Also, it would be interesting to provide a full external memory implementation of TopKMiner and study the inherent time-space tradeoffs.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, pages 207–216, 1993.
2. R. Bayardo. Efficiently mining long patterns from databases. In *Proc. ot the ACM SIGMOD Intl. Conference on Management of Data*, pages 85–93, 1998.
3. E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On maximal frequent and minimal infrequent sets in binary matrices. *Annals of Mathematics and Artificial Intelligence*, 39:211–221, 2003.
4. Y. Cheung and A. Fu. Mining frequent itemsets without support threshold: with and without item constraints. *IEEE Trans. on Knowledge and Data Engineering*, 16(9):1052–1069, 2004.
5. A. Fu, R. Kwong, and J. Tang. Mining n-most interesting itemsets. In *Proc. of the Intl. Symp. on Methodologies for Intellingent Systems*, pages 59–67, 2000.
6. B. Goethals, R. Bayardo, and M. J. Zaki, editors. *Proc. of the 2nd Workshop on Frequent Itemset Mining Implementations (FIMI04)*, volume 126. CEUR-WS Workshop On-line Proceedings, Nov. 2004.
7. B. Goethals and M. J. Zaki, editors. *Proc. of the 1st Workshop on Frequent Itemset Mining Implementations (FIMI03)*, volume 90. CEUR-WS Workshop On-line Proceedings, Nov. 2003.
8. C. Lucchese, S. Orlando, and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Trans. on Knowledge and Data Engineering*, 18(1):21–36, 2005.
9. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the 7th Int. Conference on Database Theory*, pages 398–416, Jan. 1999.
10. A. Pietracaprina and D. Zandolin. Mining frequent itemsets using Patricia tries. In *Proc. of the Workshop on Frequent Itemset Mining Implementations (FIMI03)*, Vol. 90, Melbourne, USA, Nov. 2003. CEUR-WS Workshop On-line Proceedings.
11. J. Seppanen and H. Mannila. Dense itemsets. In *Proc. of the 10th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, pages 683–688, 2004.
12. L. Shen, H. Shen, P. Pritchard, and R. Topor. Finding the n largest itemsets. In *Proc. of the IEEE Intl. Conference on Data Mining*, 1998.
13. T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating frequent closed item sets. In *Proc. of the Workshop on Frequent Itemset Mining Implementations (FIMI03)*, Vol. 90. CEUR-WS Workshop On-line Proceedings, Nov. 2003.
14. T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorith for enumerating closed patterns in transaction databases. In *Proc. of 7th Intl. Conf. Discovery Science*, pages 16–31, 2004.
15. J. Wang, J. Han, Y. Lu, and P. Tzvetkov. TFP: An efficient algorithm for mining top-k frequent closed itemsets. *IEEE Trans. on Knowledge and Data Engineering*, 17(5):652–664, 2005.
16. G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proc. of the 10th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, pages 344–353, 2004.