


# ETARM: an efficient top-k association rule mining algorithm

Linh T. T. Nguyen<sup>1</sup> · Bay Vo<sup>2</sup> · Loan T. T. Nguyen<sup>3,4</sup>  ·  
Philippe Fournier-Viger<sup>5</sup> · Ali Selamat<sup>6</sup>

Published online: 15 August 2017  
© Springer Science+Business Media, LLC 2017

**Abstract** Mining association rules plays an important role in data mining and knowledge discovery since it can reveal strong associations between items in databases. Nevertheless, an important problem with traditional association rule mining methods is that they can generate a huge amount of association rules depending on how parameters are set.

However, users are often only interested in finding the strongest rules, and do not want to go through a large amount of rules or wait for these rules to be generated. To address those needs, algorithms have been proposed to mine the top-k association rules in databases, where users can directly set a parameter  $k$  to obtain the  $k$  most frequent rules. However, a major issue with these techniques is that they remain very costly in terms of execution time and memory. To address this issue, this paper presents a novel algorithm named ETARM (Efficient Top-k Association Rule Miner) to efficiently find the complete set of top-k association rules. The proposed algorithm integrates two novel candidate pruning properties to more effectively reduce the search space. These properties are applied during the candidate selection process to identify items that should not be used to expand a rule based on its confidence, to reduce the number of candidates. An extensive experimental evaluation on six standard benchmark datasets show that the proposed approach outperforms the state-of-the-art TopKRules algorithm both in terms of runtime and memory usage.

---

✉ Loan T. T. Nguyen  
nguyenthithuyloan@tdt.edu.vn

Linh T. T. Nguyen  
thuylinh@dongan.edu.vn

Bay Vo  
bayvodinh@gmail.com

Philippe Fournier-Viger  
philfv@hit.edu.cn

Ali Selamat  
aselamat@utm.my

<sup>1</sup> Faculty of Information Technology, Dong An Polytechnic, Binh Duong, Vietnam

<sup>2</sup> Faculty of Information Technology, Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam

<sup>3</sup> Division of Knowledge and System Engineering for ICT, Ton Duc Thang University, Ho Chi Minh City, Vietnam

<sup>4</sup> Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam

<sup>5</sup> School of Humanities and Social Sciences, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, 518055, China

<sup>6</sup> Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia

**Keywords** Data mining · Association rule mining · Top-k association rules · Rule Expansion

## 1 Introduction

Association rule mining is a fundamental data mining task, which consists of discovering patterns representing strong associations between items in transactional databases [1]. Association rules are useful and important for decision-making and business planning, and are used in many fields such as marketing, business administration and e-learning. There exist several variations of the association rule mining problem [1, 24] such as non-redundant association rule

mining [17, 31], most generalization association rule mining [27] and association rule mining with various interestingness measures [25]. The traditional task of association rule mining consists of finding all associations having a support and confidence that is no less than a user-specified minimum support threshold (*minsup*) and minimum confidence threshold (*minconf*), respectively. An association rule meeting these criteria is said to be a *high confidence association rule*. The process of association rule mining is performed in two steps: (i) Step 1: Finding all frequent itemsets, i.e. groups of items (itemsets) having a support that is no less than a user-specified *minsup*; (ii) Step 2: Using the frequent itemsets found in Step 1 to generate all high confidence rules.

Although mining association rules is desirable, an important issue of traditional association rule mining algorithms is that setting their parameters is often unintuitive. Thus, users often find appropriate values for these parameters on a given dataset by trial and error. This is an important issue because on one hand, traditional algorithms can generate a huge amount of association rules and have very long execution times when the user specifies small values for the *minsup* and/or *minconf* thresholds. And on the other hand, algorithms can generate no or very few association rules if the parameters are set too high. Thus, the process of adjusting the parameters of association rule mining algorithms to just find enough but not too many rules can be very time-consuming [9]. As a solution to this problem, the TopKRules algorithm was proposed for mining the top-k association rules, in which  $k$  is a parameter set by the user indicating the number of association rules to be discovered. The algorithm then outputs the  $k$  most frequent association rules having a high confidence. To discover the top-k rules, TopKRules utilizes a search procedure that is different from traditional association rule mining algorithms. It performs two operations called left expansion and right expansion to generate new candidate rules by appending items to the antecedents and consequents of rules, respectively. Although the algorithm is correct and complete, discovering the top-k association rules remains a computationally expensive task because of the very large search space. It is thus an important research issue to design algorithms for this task that are more efficient in terms of runtime and memory usage.

In this paper, we address this issue by proposing a novel algorithm named ETARM (Efficient Top-k Association Rule Miner) to find the complete set of top-k association rules. The proposed algorithm extends the current state-of-the-art TopKRules algorithm with two novel properties to avoid performing rule expansions when some conditions are met, and thus reduce the number of candidate rules in the search space. More specifically, the two proposed properties are based on the following ideas.

- The first pruning property is applied during candidate generation to avoid expanding a given rule with items to reduce the number of new candidates. In general, an item can be used to expand a rule if the item appears in some transactions where the rule appears, is lexicographically greater than all items in the expanded side of the rule, and if the item does not appear in the other side. The novel property states that if the largest item in the antecedent (consequent) of a rule according to the lexicographical order is also the largest item in the database, the rule antecedent should not be expanded by a left (right) expansion.
- The second pruning property relies on a property of the confidence measure to reduce the search space. Based on the definition of confidence, if the confidence of a rule is less than *minconf*, no items should be appended to its consequent to generate candidate rules, as those rules are not top-k rules.

An extensive experimental evaluation on six standard benchmark datasets show that the proposed ETARM algorithm outperforms TopKRules both in terms of runtime and memory usage, thanks to the two novel pruning properties.

The rest of this paper is organized as follows. Section 2 presents important definitions related to top-k association rule mining. Related work on mining top-(rank)-k frequent (closed) itemsets and top-k association rules are reviewed in Section 3. Section 4 presents the proposed ETARM algorithm, and includes a detailed example of the algorithm's execution. Section 5 then presents an experimental evaluation on four standard benchmark datasets to evaluate the algorithm's performance both in terms of runtime and memory usage. Finally, a conclusion is drawn and future work is discussed in Section 6.

## 2 Definitions and problem statement

In this section, key concepts related to top-k association rule mining are introduced.

### Definition 1 Transaction database [2]

Let there be a finite set of items  $I = \{i_1, i_2, \dots, i_n\}$  representing all items (products) sold in a retail store. A transaction database  $D$  is a set of transactions  $T = \{T_1, T_2, \dots, T_m\}$  where each transaction  $T_d (1 \leq d \leq m)$  is a subset of the set of items  $I$ , and has a unique identifier  $d$ , called its *Tid* (*Transaction identifier*). For instance, Table 1 shows a transaction database, which will be used as running example. This database contains six transactions  $T = \{T_1, T_2, T_3, T_4, T_5, T_6\}$  and five items  $I = \{1, 2, 3, 4, 5\}$ .

**Table 1** A transaction database

Transaction	Items
$T_1$	{1, 2, 4, 5}
$T_2$	{2, 3, 5}
$T_3$	{1, 2, 4, 5}
$T_4$	{1, 2, 3, 5}
$T_5$	{1, 2, 3, 4, 5}
$T_6$	{2, 3, 4}

**Definition 2** Association rule [1]

Let there be two itemsets  $X$  and  $Y$  ( $X, Y \subset I$ ) such that  $X \cap Y = \emptyset$ . An association rule is a pattern of the form  $X \rightarrow Y$  indicating that there is an association between the presence of the itemsets  $X$  and  $Y$  in transactions.

In association rule mining, various measures have been proposed to evaluate how interesting rules are, and thus which rules should be output [25]. The two most popular measures are the support and confidence, which are defined as follows [9].

**Definition 3** Support

The support of an itemset  $X$ , denoted as  $sup(X)$ , is the number of transactions containing  $X$ . The support of an association rule  $X \rightarrow Y$ , denoted as  $sup(X \rightarrow Y)$ , is the ratio between the number of transactions containing  $X \cup Y$  and the number of transactions in the database:

$$sup(X \rightarrow Y) = \frac{sup(X \cup Y)}{|T|}$$

**Definition 4** Confidence

The confidence of an association rule  $X \rightarrow Y$ , denoted as  $conf(X \rightarrow Y)$ , is the ratio between the number of transactions containing  $X \cup Y$  and the number of transactions containing  $X$ :

$$conf(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)}$$

**Table 2** Some association rules extracted from the database of Table 1

Rule	Support	Confidence
$\{1\} \rightarrow \{2\}$	0.67	1.00
$\{2\} \rightarrow \{5\}$	0.83	0.83
$\{1, 2\} \rightarrow \{5\}$	0.67	1.00
$\{5\} \rightarrow \{1, 2\}$	0.67	0.80
$\{3\} \rightarrow \{4\}$	0.33	0.50

For example, some association rules are listed in Table 2, with their support and confidence values for the database of Table 1.

Given two user-specified threshold  $minsup$  and  $minconf$ , the problem of association rule mining consists of discovering all valid association rule in a transaction database [1]. The concept of valid association rule is defined as follows.

**Definition 5** Frequent association rule

An association rule  $r$  is frequent if  $sup(r) \geq minsup$ .

**Definition 6** Valid association rule

An association rule  $r$  is valid if it is frequent ( $sup(r) \geq minsup$ ) and  $conf(r) \geq minconf$ .

Because it can be difficult to set the parameters of traditional association rule mining algorithms, the problem of association rule mining has been redefined as the problem of top-k association rule mining [9].

**Problem statement** Let there be a transaction database  $D$ , and two user-specified parameters  $k$  (an integer) and  $minconf$  (a value in the  $[0,1]$  interval). The problem of top-k association rule mining is to discover the set  $L$  defined as

$$L = \{r \in D \mid conf(r) \geq minconf \wedge \neg \exists s \in L \mid conf(s) \geq minconf \wedge sup(s) > sup(r)\}.$$

Thus, the goal of top-k association rule mining is to find the  $k$  most frequent association rules having a confidence no less than  $minconf$ . However, note that the set  $L$  may contain more than  $k$  rules if several rules have exactly the same support.

To describe association rules, several definition and properties have been introduced. The following paragraphs present some definitions and properties that will be used in the rest of this paper [9].

**Definition 7** Size of a rule

The size of an association rule  $X \rightarrow Y$  is  $p * q$ , where  $p = |X|$  and  $q = |Y|$ . A rule of size  $p * q$  is larger than a rule of size  $r * s$  if  $p > r$  and  $q \geq s$ , or  $p \geq r$  and  $q > s$ .

**Definition 8** Tidset of an itemset

The tidset of an itemset  $X$  is denoted as  $tids(X)$  and defined as  $tids(X) = \{t \mid t \in T \wedge X \subseteq t\}$ .

For example, in the database of Table 1,  $tids(\{1, 2\}) = \{T_1, T_3, T_4, T_5\}$ .

**Definition 9** Tidset of a rule

The tidset of a rule  $X \rightarrow Y$  is denoted as  $tids(X \rightarrow Y)$  and is defined as  $tids(X \cup Y)$ . The support and confidence of a rule  $X \rightarrow Y$  can be also expressed in terms of tidsets:

$$sup(X \rightarrow Y) = \frac{|tids(X \cup Y)|}{|T|}$$

$$conf(X \rightarrow Y) = \frac{|tids(X \cup Y)|}{|tids(X)|}$$

**Property 1** For any rule  $X \rightarrow Y$ ,  $tids(X \rightarrow Y) = tids(X) \cap tids(Y)$ .

To discover the top-k association rules in a database, two processes called left expansion and right expansion have been proposed [9].

**Definition 10** Left expansion of a rule

A left expansion is the process of adding an item  $i \in I$  to the left hand side of a rule  $X \rightarrow Y$  to form a new rule  $X \cup \{i\} \rightarrow Y$  of greater size.

**Definition 11** Right expansion of a rule

A right expansion is the process of adding an item  $i \in I$  to the right hand side of a rule  $X \rightarrow Y$  to form a new rule  $X \rightarrow Y \cup \{i\}$  of greater size.

Left and right expansions have the following properties [9].

**Property 2** Let there be an item  $i \in I$  and two association rules  $r : X \rightarrow Y$  and  $r' : X \cup \{i\} \rightarrow Y$ . It follows that  $sup(r) \geq sup(r')$ .

**Property 3** Let there be an item  $i \in I$  and two association rules  $r : X \rightarrow Y$  and  $r' : X \rightarrow Y \cup \{i\}$ . It follows that  $sup(r) \geq sup(r')$ .

Property 2 and property 3 state that the support of an expanded rule is less than or equal to that of the original rule. This means that expanding a rule having a support less than *minsup* will always produce an infrequent rule (cf. Definition 5). Hence, all the frequent rules can be found by recursively performing left and right expansions, starting from frequent rules of size  $1 * 1$ .

**Property 4** Let there be an item  $i \in I$  and two rules  $r : X \rightarrow Y$  and  $r' : X \rightarrow Y \cup \{i\}$ . It follows that  $conf(r) \geq conf(r')$ .

**3 Related work**

In data mining, several algorithms have been proposed to discover various types of top-k patterns in databases. This section first reviews basic techniques for itemset mining, algorithms for mining top-k or top-rank-k frequent itemsets, and then algorithms for association rule mining and top-k association rule mining.

**3.1 Frequent itemset mining**

Frequent itemset mining is the first step of the traditional association rule mining process [2]. It consists of discovering the frequent itemsets in a transaction database, i.e. the sets of items having a support no less than a user-specified *minsup* threshold. The first algorithm for this task is Apriori [2]. Let an itemset containing  $z$  items be called a  $z$ -itemset. Apriori discovers all frequent itemsets in a database using a recursive level-wise search starting from 1-itemsets. During the search, Apriori utilizes frequent itemsets of size  $z$  to generate those of size  $z + 1$ . Apriori first scans the database to find frequent items ( $L_1$ ). Then, Apriori utilizes  $L_1$  to find  $L_2$ , the frequent 2-itemsets. Then,  $L_2$  is used to find  $L_3$ , and this process is repeated until no more frequent itemsets are found. During each of these steps (levels), Apriori scans the database to calculate the support of patterns. The main drawbacks of the Apriori algorithm are that (i) it can generate a huge amount of candidate itemsets, and (ii) it scans the database multiple times to evaluate each of those candidates. Hence, if the database contains a huge number of transactions, the performance of Apriori deteriorates.

To reduce the number of database scans performed by Apriori, the AprioriTid algorithm was proposed [2]. AprioriTid uses the same candidate generation function as Apriori (Apriori-Gen). But to reduce the cost of database scans, AprioriTid uses the concept of tidsets of itemsets. AprioriTid only scans the database once for counting the support of 1-itemsets. Then, to calculate the support of any candidate  $z$ -itemsets ( $z > 1$ ) without scanning the database, an intersection operation is performed with the tidsets of some of its subsets. It was shown that Apriori generally outperforms AprioriTid for the first levels of the level-wise search, while AprioriTid performs better for subsequent levels. Hence, Agrawal and Srikant proposed the AprioriHybrid algorithm [2], which combines both Apriori and AprioriTid. This algorithm uses Apriori for the first levels and then switches to AprioriTid for the remaining levels. AprioriHybrid was shown to outperform both Apriori and AprioriTid.

Despite the above improvements, Apriori still suffers from the problem of generating too many candidates. To address this problem, Han et al. proposed the FP-Growth algorithm [11]. It relies on a data structure called FP-Tree

to discover all frequent itemsets without generating candidates. The FP-Tree is an extended prefix tree structure that stores a compressed representation of a transaction database. In an FP-tree, nodes represent items and paths represent transactions. All items in transactions are sorted by descending order of support in an FP-Tree to have more chances that paths overlap and thus of obtaining a more compact tree. To build an FP-Tree, FP-Growth scans a database twice. In the first scan, FP-Growth calculates the support of each item. Then, FP-Growth sorts frequent items in descending order of support, and discards infrequent items. In the second scan, FP-Growth inserts each transaction in the FP-tree. If a transaction is unique, it will form a new path in the tree and the counter associated to each path node is set to 1. Otherwise, the transaction shares a common prefix itemset with other transaction(s) in the tree. Hence, the counters of the nodes representing this prefix will be incremented by one, and each other node will be created with a counter set to 1. After the construction of the FP-Tree, FP-Growth performs a depth-first search to enumerate all frequent itemsets without scanning the database. During this process, FP-Growth recursively constructs projected FP-trees.

### 3.2 Top-k and top-rank-k frequent itemset mining

Numerous approaches have been proposed for mining frequent (closed) itemsets in transaction databases [1, 2, 6, 7, 10, 11, 14, 15, 20, 26]. However, a key issue with traditional frequent itemset mining algorithms is that setting the *minsup* parameter is unintuitive despite that it greatly influences the process of frequent itemset mining. On one hand, if a user specifies a *minsup* value that is too small, there can be a huge number of frequent itemsets, and as a result mining algorithms may become much slower and consume much more memory. On the other hand, if a user specifies a *minsup* value that is too large, too few or no frequent itemsets may be found.

To address this problem, the task of top-k frequent itemset mining was proposed. It consists of discovering the  $k$  most frequent itemsets in a transaction database. Setting the parameter  $k$  is more intuitive than setting the *minsup* parameter of traditional frequent itemset mining since  $k$  is the number of itemsets that the user wants to analyze. Han et al. proposed the TFP algorithm for mining top-k closed frequent itemsets [12], where  $k$  is the number of closed frequent itemsets to be found. This algorithm lets users specify a constraint on the minimum size of patterns. Pietracaprina and Vandin proposed the TopKMiner algorithm for mining the top-k closed frequent itemsets in descending order of support [18]. This algorithm utilizes a priority queue to speed up the mining process. Tzvetkov et al. proposed the

TSP algorithm for mining the top-k closed frequent itemsets respecting a minimum length constraint [23]. Pyun and Yun proposed the CRM (Combination Reducing Method) for mining top-k frequent patterns [19]. CRM converts patterns obtained from single-paths into composite patterns during the mining process and recovers them as the original patterns when the top-k frequent patterns are extracted. Chuang et al. proposed the MTK (Memory-constraint Top-k Mining) and MTK\_Close algorithms for mining the top-k (closed) itemsets in the presence of a memory constraint [3].

A variation of the top-k itemset mining problem called top-rank-k frequent itemset mining has also been studied. Mining top-rank-k frequent itemsets consists of finding itemsets that belong to the first  $k$  ranks when itemsets are ordered by decreasing order of support. FAE (Filtering and Extending) [4] is the first algorithm for mining top-rank-k frequent itemsets. Fang and Deng also proposed the VTK (Vertical mining of top-rank-k frequent patterns) algorithm [8]. It is more efficient than FAE since it calculates the support of itemsets without scanning the database. In 2014, Deng proposed the NTK algorithm for mining the top-rank-k frequent itemsets using the Node-List structure. This algorithm outperforms both FAE and VTK [5]. Huynh-Thi-Le et al. then proposed the iNTK algorithm to improve the performance of frequent itemset mining [13]. Saif-Ur-Rehman et al. proposed the Top-k Miner algorithm for mining the top-rank-k frequent itemsets using the CIS-tree (candidate-itemsets-search tree) [22]. Nguyen et al. proposed an efficient strategy for mining top-rank-k frequent closed itemsets [16] by modifying the DCI-plus algorithm [21] and adopting the dynamic bit vector technique [26].

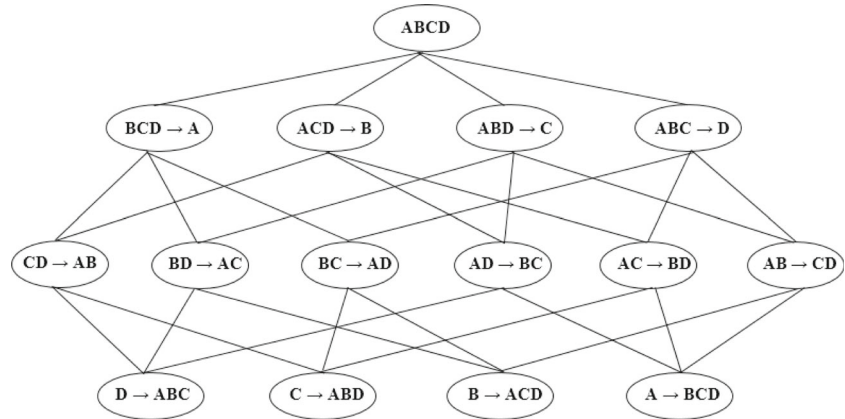
### 3.3 Association rule mining and top-k association rule mining

As previously explained, association rule mining [1, 2] is the task of enumerating all valid association rules appearing in a transaction database. Traditional association rule is performed in two steps: (i) finding all non empty frequent itemsets, and then (ii) for each frequent itemset  $l$  and non empty subset  $a$  of  $l$ , output the rule  $r : a \rightarrow (l - a)$ , if  $conf(r) \geq minconf$  (if  $r$  is a valid rule). For example, Fig. 1 shows all the association rules that can be generated from the itemset {A, B, C, D}.

Although association rule mining is useful, a limitation of traditional algorithms is that setting their parameters can be unintuitive for users. To address this issue, several studies have been carried related to top-k association rule mining such as k-optimal rule discovery [28], filtered top-k association rule discovery [29], and mining the top-k fault tolerant association rules by redundant pattern disambiguation in data streams [30]. A limitation of the two first studies is that



**Fig. 1** Rules that can be generated from the itemset {A, B, C, D}



they cannot find rules containing more than one item in the consequent, while the third study is designed for handling streams rather than a transaction database as in traditional association rule mining. Recently, an algorithm named TopKRules was proposed for mining the top-k association rules in a transaction database without restriction on the number of items [9]. This algorithm extracts the  $k$  most frequent association rules having a confidence no less than a *minconf* threshold. Thus, the TopKRules algorithm replaces the *minsup* parameter by the parameter  $k$ , which is more intuitive for users. However, an issue with this algorithm is that discovering the top-k association rules can be very costly in terms of runtime and memory usage because many candidate rules may be generated by the process of rule expansion. Besides, memory consumption of this algorithm should be optimized to achieve better performance.

### 3.4 TopKRules algorithm

The main steps of the TopKRules algorithm are as follows [9].

- (i). Initialize *minsup* = 0
- (ii). Generate all rules of size 1\*1, that is rules having an antecedent and consequent containing one item. During this process, generated rules satisfying the top-k condition are saved into a list  $L$ . The list  $L$  contains at any given moment the  $k$  most frequent rules satisfying the *minconf* threshold, found until now. Note that if several rules have the same support, the list  $L$  may contain more than  $k$  rules. When updating the list  $L$  by inserting a new rule, if the number of rules is greater than  $k$ , rules having a support less than the support of the  $k$  most frequent rules are removed from  $L$ . After that, the *minsup* threshold is set to the support of the least frequent rule in  $L$ . Besides that, for each rule of size 1\*1, if the rule can be expanded to potentially generate some other frequent rules, it

will be added to a set of candidates  $R$ . A rule is said to potentially generate some other frequent rules if it has a support greater than or equal to *minsup*.

- (iii). Then a recursive exploration of the search space of rules is performed by expanding rules from the set of candidates  $R$ . During this process, the first rule to be expanded is always the rule having the highest support that is greater than *minsup*. Rules are expanded using left or right expansions (by adding candidate items to the left or right sides of rules). A candidate item is an item that is lexicographically greater than all items in the expanded side and do not appear in the other side. After generating a rule by left or right expansion, it is added to the set  $L$  and  $R$  if the required conditions for these sets are met, respectively. After a rule has been expanded to generate one or more rule(s), it is removed from the set  $R$ . All rules having a support less than the current *minsup* threshold are removed from  $R$ . The expansion process ends when  $R$  is empty. The sets  $R$  and  $L$  are sorted by decreasing support, and are implemented using the heap data structure to increase performance.

## 4 The proposed algorithm

The next subsection explains how TopKRules has been modified to obtain the proposed ETARM algorithm.

### 4.1 The ETARM algorithm

The proposed ETARM algorithm improves TopKRules by integrating two novel propositions:

**Proposition 1** *If the largest item in the antecedent (consequent) of a rule according to the lexicographical order is also the largest item in the database, the rule antecedent should not be expanded by a left (right) expansion.*

**Proposition 2** *If the confidence of a rule is less than minconf, it should not be expanded by right expansion, as the resulting rule will not be a top-k rule. Let  $r'$  be a rule obtained by performing a right expansion of a rule  $r$ , such that the confidence of the rule  $r$  is less than minconf. It can be easily proven that the confidence of the rule  $r'$  is less than minconf.*

**Reasonable** By the definition of confidence, we have that:

$$\text{conf}(X \rightarrow Y) = \frac{|\text{tids}(X \cup Y)|}{|\text{tids}(X)|}$$

$$\text{and } \text{conf}(X \rightarrow Y \cup \{c\}) = \frac{|\text{tids}(X \cup Y \cup \{c\})|}{|\text{tids}(X)|}.$$

Since  $|\text{tids}(X \cup Y \cup \{c\})| \leq |\text{tids}(X \cup Y)|$ , it follows that  $\text{conf}(X \rightarrow Y \cup \{c\}) \leq \text{conf}(X \rightarrow Y)$ .

Because  $\text{conf}(X \rightarrow Y) < \text{minconf}$ , it follows that  $\text{conf}(X \rightarrow Y \cup \{c\}) < \text{minconf}$ .

The proposed ETARM algorithm is presented in Fig. 2. Initially, the *minsup* threshold is set to 0. Then, the algorithm scans the database  $D$  to construct the tidset of each item. Then, items are sorted according to a total order such as the lexicographical order. The algorithm performs two main steps.

- Step 1: Generate all frequent rules of size 1\*1 by considering each pair of items  $i, j$  to determine if the support of  $\{i, j\}$  is no less than *minsup*. For each frequent pair  $\{i, j\}$ , the rules  $\{i\} \rightarrow \{j\}$  and  $\{j\} \rightarrow \{i\}$  are created and valid rules are added to  $L$ , which stores the current top-k rules (step 1.1 and step 1.2). Moreover, for each rule  $r$ , if its left hand side contains the largest item according to the total order, the rule  $r$  can only be expanded on the right hand side (by proposition 1). Otherwise, the rule  $r$  can be expanded on both the left and right hand sides and the two rules are added to  $R$  (the set of candidate rules to be considered for generating other rules by expansions) (Step 1.3).
- Step 2: After generating rules of size 1\*1, the algorithm recursively expands rules using the set of candidates  $R$  to generate larger rules. This process ends when there are no remaining candidates in the set  $R$ . During the expansion process, the algorithm always expand the rule having the highest support in  $R$  first, on the left hand side or right hand side by calling the procedures EXPANDL and EXPANDR (Step 2.1). Then, the algorithm removes the considered rule out of the set of candidates  $R$  (Step 2.2) and removes rules having a support less than *minsup* from  $R$  (Step 2.3).

---

ETARM(*minconf*,  $k$ ,  $T$ )

*minsup* := 0;  $R := \emptyset$ ;  $L := \emptyset$

Scan the database  $D$  and store the *tidset* of each single item.

Step 1. For each pair of items  $(i, j)$  having a support greater than or equal to *minsup*, construct the two rules:  $r_1: \{i\} \rightarrow \{j\}$  and  $r_2: \{j\} \rightarrow \{i\}$

IF  $\text{sup}(r_1) \geq \text{minsup}$  THEN

Step 1.1. IF  $\text{conf}(r_1) \geq \text{minconf}$  THEN

SAVE( $r_1, L, k, \text{minsup}$ )

Step 1.2. IF  $\text{conf}(r_2) \geq \text{minconf}$  THEN

SAVE( $r_2, L, k, \text{minsup}$ )

Step 1.3. For every pair of rules  $(r_1 \text{ and } r_2)$ , if the largest item in the left hand side according to the lexicographical order is equal to the largest item in the database, an expansion flag for the rule is set to false (indicating that the rule should only be expanded by right expansions). Otherwise, the expansion flag is set to true (indicating that both sides can be expanded).

$R := R \cup \{r_1, r_2\}$

Step 2. Expansions are performed recursively using rules in  $R$ . During this process, the rule  $r$  having the highest support in  $R$  is used first to generate other rules. This process is repeated until  $R$  is empty:

Step 2.1. IF  $\text{sup}(r) \geq \text{minsup}$  THEN

IF  $r.\text{ExLR} = \text{true}$  THEN

EXPANDL( $r, L, R, k, \text{minsup}, \text{minconf}$ )

EXPANDR( $r, L, R, k, \text{minsup}, \text{minconf}$ )

ELSE

EXPANDR( $r, L, R, k, \text{minsup}, \text{minconf}$ )

Step 2.2. Remove  $r$  from  $R$

Step 2.3. Remove all rules having a support less than *minsup* in  $R$

---

**Fig. 2** The ETARM Algorithm

**The SAVE Procedure** (Fig. 3) is used to update the set  $L$  with a new rule. It initially adds a valid rule  $r$  to the set  $L$ . If the number of rules in  $L$  is greater than  $k$ , the algorithm removes rules from  $L$  that have a support less than the current top-k rules in  $L$ . After that, the procedure set the *minsup* value to the support of the least frequent rule in  $L$ .

**EXPANDL and EXPANDR procedures** (Figs. 4 and 5). These procedures are used to recursively generate larger rules using a candidate rule  $r$ . The input parameters of these procedures are the rule  $r$ , the sets  $L$  and  $R$ ,  $k$ , *minsup* and *minconf*. An expansion of a rule  $r$  is done by adding an item  $p$  to the left or right side of  $r$ . An item can be added to a rule  $r$  if the item appears in some transactions where the rule

**SAVE**( $r, L, k, minsup$ )

- Step 1. Add  $r$  to  $L$ .
- Step 2. If the number of rules in  $L$  is greater than  $k$ , the algorithm removes the rules having a support less than the top- $k$  rules in  $L$ . After that,  $minsup$  is set to the support of the least frequent rule in  $L$ .

**Fig. 3** The SAVE procedure

$r$  appears, is lexicographically greater than all items in the expanded side of the rule, and if the item does not appear in the other side.

The left expansion procedure **EXPANDL** is presented in Fig. 4. It adds items one by one to the left hand side of a rule  $r$  to form new rules, where each the size of rule  $r'$  is larger than its of rule  $r$ . If  $r'$  has a support greater than or equal to  $minsup$  and its confidence is not less than  $minconf$  then  $r'$  is inserted into  $L$  using the SAVE procedure. If the lexicographically largest item in the left hand side of  $r'$  is the largest item in the database, the rule should only be expanded by right expansion(s) (by proposition 1). Otherwise, this rule can be considered for left and right expansions. The rule  $r'$  is added to  $R$ .

The procedure for expanding the right hand side of a rule, **EXPANDR** (Fig. 5), scans transactions containing a rule  $r$  to find all candidate items that can expand the right hand side size of  $r$  to form a new rule  $r'$ . An item is a candidate item if it appears in some transactions where the rule  $r$  appears, is lexicographically greater than the largest item in the right hand side of the rule  $r$ , and do not appear in the left hand side of  $r$ . If the new rule  $r'$  satisfies the  $minsup$  and  $minconf$  thresholds, it is added to  $L$ . If the lexicographically largest item in the rule  $r'$  is the largest item in the database, this new

**EXPANDL**( $r, L, R, k, minsup, minconf$ )

- Step 1. Find all items that can be added to the left hand side of  $r$ . These candidate items are the ones appearing in some transactions where  $r$  appears, are lexicographically greater than all items in the left hand side of  $r$ , and do not appear in the right hand side.
- Step 2. For each candidate item  $p$ , try to add  $p$  to the left hand side of  $r$  to form a new rule  $r'$ . If  $sup(r') \geq minsup$  then
- IF  $conf(r') \geq minconf$  THEN
- SAVE( $r', L, k, minsup$ )
- If the lexicographically largest item in the left hand side of  $r'$  is the largest item in the database,  $r'.ExLR = false$ . Otherwise  $r'.ExLR = true$
- $R := R \cup \{r'\}$

**Fig. 4** The EXPANDL procedure**EXPANDR**( $r, L, R, k, minsup, minconf$ )

- Step 1. Find all candidate items that can be used to expand the right hand side of the rule  $r$ . A candidate item appears in some transactions where  $r$  appears, is lexicographically larger than all items in the right hand side of  $r$ , and do not appear in the left hand side of  $r$ .
- Step 2. Each candidate item  $p$  is used to generate a new rule  $r'$  by performing a right expansion of  $r$  with  $p$ . If  $sup(r') \geq minsup$  then:

IF  $conf(r') \geq minconf$  THEN

SAVE( $r', L, k, minsup$ )

If the lexicographically largest item in the right hand side is not the largest item in the database.

$r'.ExLR = false$

$R := R \cup \{r'\}$

**Fig. 5** The EXPANDR procedure

rule should not be expanded by right expansion (by proposition 1). Otherwise, it can be considered for right expansion. If  $r'$  only satisfies  $minsup$  and does not satisfy  $minconf$ , the rule should not be expanded (by proposition 2). Thus, it is not added to  $R$ .

**4.2 Illustration**

Consider the database of Table 1,  $minconf = 0.8$ , and  $k = 10$ . The following paragraphs explain how the proposed ETARM algorithm is applied step by step to extract the top- $k$  association rules.

Initially, variables are initialized as:  $minsup := 0$ ;  $R := \emptyset$ ;  $L := \emptyset$ .

Then, the algorithm scans the database to create the tidsets (list of tids) of all items, as shown in Table 3.

- Step 1. The rules of size  $1*1$  are then generated. The algorithm utilizes the pairs of items (1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5) to create and extract valid rules. To illustrate this process, consider the pair of items (1, 2). This pair is used to generate the rules  $r_1 : \{1\} \rightarrow \{2\}$  ( $support = 0.667$ ) and  $r_2 :$

**Table 3** Tidsets of items

Item	Tidset
1	{1, 3, 4, 5}
2	{1, 2, 3, 4, 5, 6}
3	{2, 4, 5, 6}
4	{1, 3, 5, 6}
5	{1, 2, 3, 4, 5}



$\{2\} \rightarrow \{1\}$  ( $support = 0.667$ ). The support of these rules is no less than the  $minsup$  threshold.

Step 1.1. The algorithm calculates the confidence of the rule  $r_1$ . Since the confidence is 1, which is not less than  $minconf$ ,  $r_1$  is added to  $L$ .

Step 1.2. The same process is repeated for  $r_2$ .  $r_2$  is added to  $L$ .

Step 1.3.

$r_1.ExLR := true$

$r_2.ExLR := true$

$R := R \cup \{r_1, r_2\}$

This process is then repeated for all the other pairs of items. The set  $L$  obtained after this step is shown in Table 4, and the set  $R$  of candidate rules is presented in Table 5.

Step 2. The algorithm selects the rules having the highest support in  $R$  that satisfy the current  $minsup$  value, to perform expansions.

- To illustrate this process, consider the rule  $\{5\} \rightarrow \{2\}$ ,

Step 2.1. Because this rule can only be expanded on the right hand side, the procedure EXPANDR is called.

Step 2.2. Consider the candidate items  $\{3, 4\}$ . These items are used to generate two rules:  $\{5\} \rightarrow \{2, 3\}$  and  $\{5\} \rightarrow \{2, 4\}$ . It is found that:  $sup(\{5\} \rightarrow \{2, 3\}) = 0.5$  and  $conf(\{5\} \rightarrow \{2, 3\}) = 0.6$ . Since the rule  $\{5\} \rightarrow \{2, 3\}$  has a support that is no less than the current  $minsup$  value, and its confidence value is less than  $minconf$ , this rule is not added to  $L$  and  $R$ . The

same process is done for the rule  $\{5\} \rightarrow \{2, 4\}$ .

Step 2.3. The rule  $\{5\} \rightarrow \{2\}$  is removed from  $R$ .

Step 2.4. All rule having a support less than  $minsup$  are removed from  $R$ .

- This process is then repeated for all remaining rules, until  $R = \emptyset$
- After finishing step 2, the algorithm has identified the set of top-k rules for  $minsup = 0.667$ . These 10 rules are listed in Table 6.

**Table 4** Rules of size 1\*1

Rule	Support	Confidence
$\{1\} \rightarrow \{2\}$	0.667	1.000
$\{1\} \rightarrow \{5\}$	0.667	1.000
$\{5\} \rightarrow \{1\}$	0.667	0.800
$\{3\} \rightarrow \{2\}$	0.667	1.000
$\{4\} \rightarrow \{2\}$	0.667	1.000
$\{2\} \rightarrow \{5\}$	0.833	0.833
$\{5\} \rightarrow \{2\}$	0.833	1.000

**Table 5** Set of candidate rules

Rule	Support	Confidence	Is considered for expansion?
$\{1\} \rightarrow \{3\}$	0.333	0.500	True
$\{3\} \rightarrow \{1\}$	0.333	0.500	True
$\{3\} \rightarrow \{4\}$	0.333	0.500	True
$\{4\} \rightarrow \{3\}$	0.333	0.500	True
$\{1\} \rightarrow \{4\}$	0.500	0.750	True
$\{4\} \rightarrow \{1\}$	0.500	0.750	True
$\{3\} \rightarrow \{5\}$	0.500	0.750	True
$\{5\} \rightarrow \{3\}$	0.500	0.600	False
$\{4\} \rightarrow \{5\}$	0.500	0.750	True
$\{5\} \rightarrow \{4\}$	0.500	0.600	False
$\{1\} \rightarrow \{2\}$	0.667	1.000	True
$\{2\} \rightarrow \{1\}$	0.667	0.667	True
$\{1\} \rightarrow \{5\}$	0.667	1.000	True
$\{5\} \rightarrow \{1\}$	0.667	0.800	False
$\{2\} \rightarrow \{3\}$	0.667	0.667	True
$\{3\} \rightarrow \{2\}$	0.667	1.000	True
$\{2\} \rightarrow \{4\}$	0.667	0.667	True
$\{4\} \rightarrow \{2\}$	0.667	1.000	True
$\{2\} \rightarrow \{5\}$	0.833	0.833	True
$\{5\} \rightarrow \{2\}$	0.833	1.000	False

**Table 6** Set of top-k rules mined for  $k = 10$  and  $minconf = 0.8$

Rule	Support	Confidence
$\{5\} \rightarrow \{1\}$	0.667	0.800
$\{3\} \rightarrow \{2\}$	0.667	1.000
$\{4\} \rightarrow \{2\}$	0.667	1.000
$\{5\} \rightarrow \{1, 2\}$	0.667	0.800
$\{1, 2\} \rightarrow \{5\}$	0.667	1.000
$\{2, 5\} \rightarrow \{1\}$	0.667	0.800
$\{1\} \rightarrow \{2, 5\}$	0.667	1.000
$\{1, 5\} \rightarrow \{2\}$	0.667	1.000
$\{2\} \rightarrow \{5\}$	0.833	0.833
$\{5\} \rightarrow \{2\}$	0.833	1.000

**Table 7** Testing datasets

Name	Number of transactions	Number of items	Average length of transactions
Chess	3,196	75	37
Connect	67,557	129	43
Mushroom	8,416	128	23
Pumsb	49,046	7,116	74
T25I10D10K	10,000	1,000	25
Retail	88,162	16,470	10.3

## 5 Experimental results

To evaluate the performance of the proposed algorithm, experiments have been conducted. The first subsection describes the datasets and the testing environment. Then, the second subsection presents the results.

### 5.1 Datasets and testing environment

The experiments were carried on a computer having an Intel Core I5-6200U 2.3 GHz processor, 8 GB of RAM, running Windows 10 (64 bit version). The proposed ETARM algorithm and the TopKRules algorithms were implemented in Java. Six standard datasets were used for testing, which were downloaded from <http://www.philippe-fournier-viger.com/spmf/>. The characteristics of these datasets are presented in Table 7. These datasets have been chosen because they have varied characteristics.

### 5.2 Experimental results

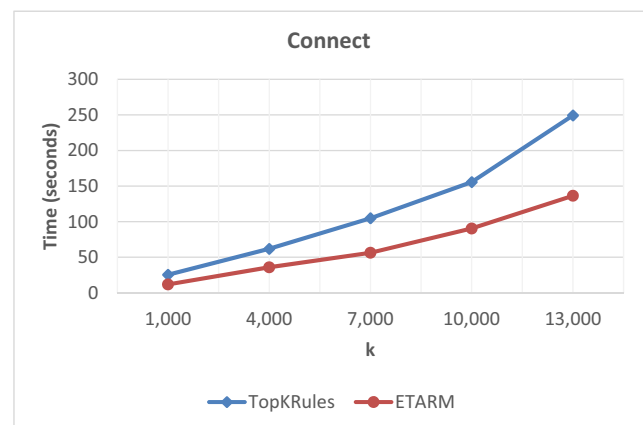
The proposed ETARM and TopKRules algorithms were executed on the six standard datasets presented in Table 7. The execution time and maximum memory usage were recorded for  $minconf = 0.8$ , while  $k$  was varied from 1,000

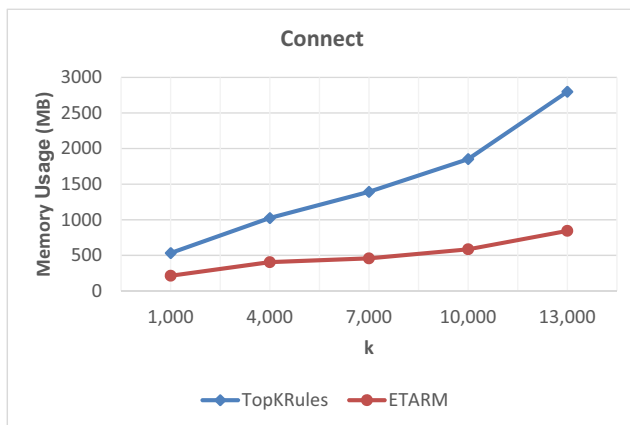
**Fig. 7** Memory usage of TopKRules and ETARM on the Chess dataset

to 13,000. The results are presented in Figs. 6, 7, 8, 9, 10, 11, 12 and 13. Note that these two algorithms produce the same rule set for each dataset when the value of  $k$  is the same.

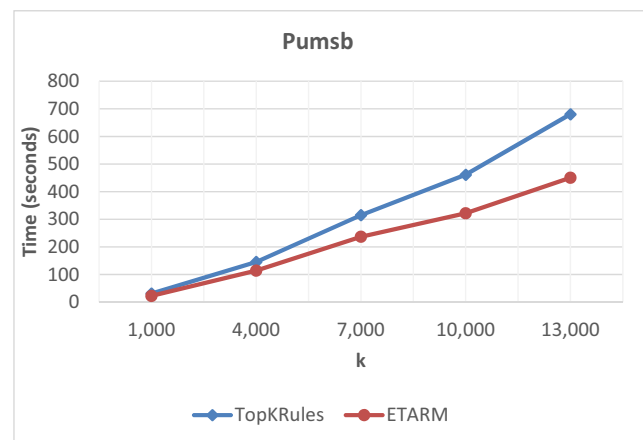
Figures 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17 show that ETARM outperforms TopKRules both in terms of runtime and memory usage, on all datasets and for all parameter settings. The difference in terms of runtime and memory usage is not very large for small values of  $k$ . But when large values of  $k$  are used to mine rules, the difference in terms of runtime and memory usage is quite considerable.

Figures 6, 8, 10, 12, 14, 16 compare the runtime of the ETARM and TopKRules algorithms on the six datasets. It can be observed that as  $k$  is increased, the gap in terms of execution time between ETARM and TopKRules increases. In Fig. 6, for  $k = 1,000$ , both algorithms have nearly the same execution time. However, for  $k = 4,000, 7,000$ , and  $10,000$ , ETARM is faster than TopKRules. In particular, for  $k = 13,000$ , ETARM is twice faster than TopKRules. In Fig. 10, for  $k = 13,000$ , on the Mushroom dataset, ETARM is about five times faster than TopKRules.

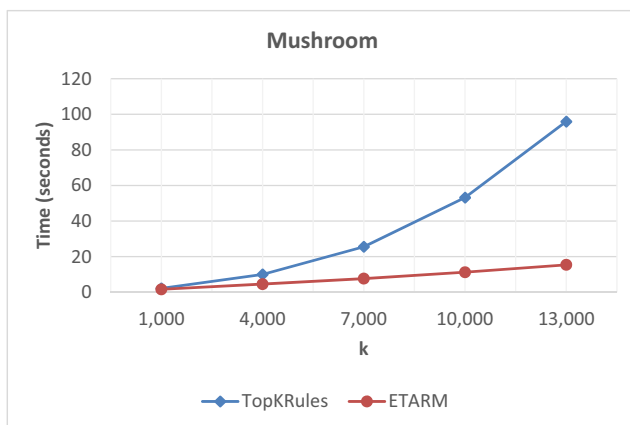
**Fig. 6** Runtime of TopKRules and ETARM on the Chess dataset**Fig. 8** Runtime of TopKRules and ETARM on the Connect dataset



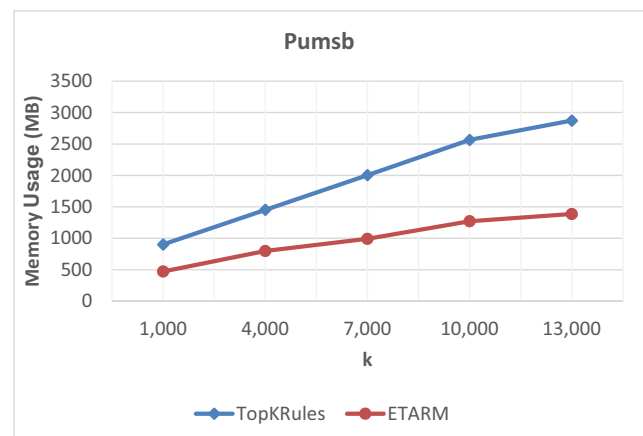
**Fig. 9** Memory usage of TopKRules and ETARM on the Connect dataset



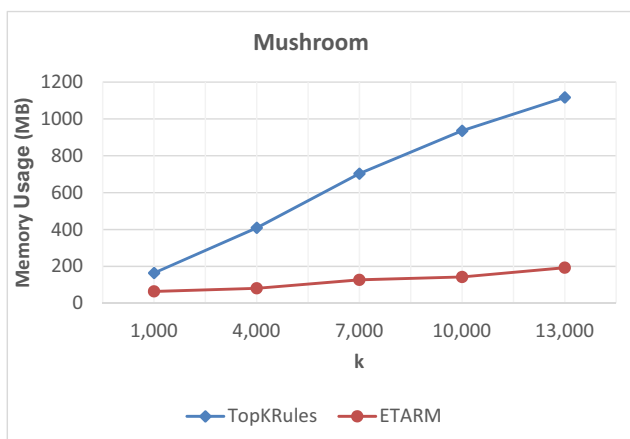
**Fig. 12** Runtime of TopKRules and ETARM on the Pumsb dataset



**Fig. 10** Runtime of TopKRules and ETARM on the Mushroom dataset



**Fig. 13** Memory usage of TopKRules and ETARM on the Pumsb dataset



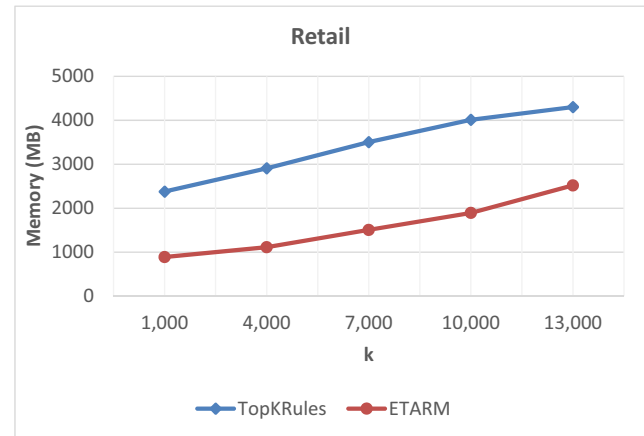
**Fig. 11** Memory usage of TopKRules and ETARM on the Mushroom dataset



**Fig. 14** Runtime of TopKRules and ETARM on the T25I10D10K dataset

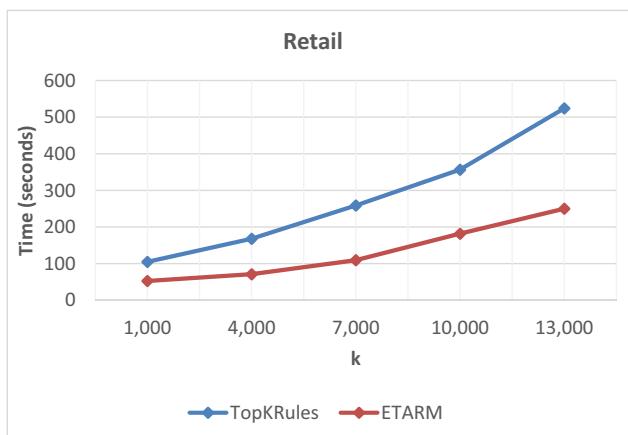


**Fig. 15** Memory usage of TopKRules and ETARM on the T25I10D10K dataset



**Fig. 17** Memory usage of TopKRules and ETARM on the Retail dataset

Figures 7, 9, 11, 13, 15 and 17 show that ETARM consumes less memory than TopKRules. Figure 11 compares the memory usage of TopKRules and ETARM for mining the top-k rules in the Mushroom dataset. It can be observed that the memory usage of ETARM increases slightly when  $k$  is increased, and did not exceed 200 MB. On the other hand, the memory usage of TopKRules increased from 200 MB to 1.1 GB when  $k$  is increased from 1,000 to 13,000. The results are similar on the Chess and Connect datasets. ETARM consumed about twice less memory than TopKRules. On the Pumsb, T25I10D10K and Retail dataset, ETARM consumes half the amount of memory used by TopKRules. By comparing Figs. 7, 9 and 11 to Figs. 13, 15, 17 and Table 7, it can be observed that memory usage is higher for datasets having a large number of items. Datasets such as T25I10D10K (1,000 items) contain up to eight times more items than datasets such as Connect (129 items).



**Fig. 16** Runtime of TopKRules and ETARM on the Retail dataset

## 6 Conclusion and future work

In this paper, we have proposed an efficient algorithm to mine the top-k association rules in transactional databases. The proposed ETARM algorithm addresses the problem that setting the *minsup* threshold in traditional association rule mining to find enough useful association rules is unintuitive. The proposed algorithm mines the top-k association rules having the highest support that satisfy a *minconf* threshold. Two novel pruning properties were integrated in the algorithm to reduce the search space, and thus decrease the runtime and memory usage. An experimental evaluation was carried on four benchmark datasets. Experimental results have shown that ETARM outperforms the state-of-the-art TopKRules algorithm both in terms of runtime and memory usage to mine the top-k association rules, and performs very well for large values of  $k$ .

The proposed algorithm still requires that users set the *minconf* threshold for mining association rules. In future work, we will investigate how to mine the top-k association rules without using the *minconf* threshold to find rules that have the highest confidence. Moreover, we will also investigate the problem of mining the top-k non-redundant association rules.

**Acknowledgement** This work was carried out during the tenure of an ERCIM ‘Alain Bensoussan’ Fellowship Programme.

## References

1. Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings ACM international conference on management of data. ACM Press, pp 207–216

2. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large data bases, pp 487–499
3. Chuang KT, Huang JL, Chen MS (2008) Mining top-k frequent patterns in the presence of the memory constraint. *Vldb J* 17(5):1321–1344
4. Deng Z, Fang G (2007) Mining top-rank-k frequent patterns. In: *ICMLC'07*, pp 851–856
5. Deng ZH (2014) Fast mining top-rank-k frequent patterns by using node-lists. *Expert Syst Appl* 41(4):1763–1768
6. Deng ZH, Lv SL (2015) PrePost<sup>+</sup>: an efficient N-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning. *Expert Syst Appl* 42(13):5424–5432
7. Deng ZH (2016) DiffNodesets: an efficient structure for fast mining frequent itemsets. *Appl Soft Comput* 41:214–223
8. Fang G, Deng ZH (2008) VTK: vertical mining of top-rank-k frequent patterns. In: *FSKD'08*, pp 620–624
9. Fournier-Viger P, Wu C-W, Tseng VS (2012) Mining top-k association rules. In: Proceedings of the 25th Canadian conference on artificial intelligence AI (2012). Springer, LNAI 7310, pp 61–73
10. Han J, Dong G, Yin Y (1999) Efficient mining of partial periodic patterns in time series database. In: *ICDE'99*, pp 106–115
11. Han J, Pei H, Yin Y (2004) Mining frequent patterns without candidate generation. In: Proceedings ACM international conference on management of data (SIGMOD'00, Dallas, TX), vol 8(1). ACM Press, pp 53–87
12. Han J, Wang J, Lu Y, Tzvetkov P (2002) Mining top-k frequent closed patterns without minimum support. In: *ICDM'02*, pp 211–218
13. Huynh-Thi-Le Q, Le T, Vo B, Le B (2015) An efficient and effective algorithm for mining top-rank-k frequent patterns. *Expert Syst Appl* 42(1):156–164
14. Le T, Vo B (2015) An N-list-based algorithm for mining frequent closed patterns. *Expert Syst Appl* 42(19):6648–6657
15. Lucchese C, Orlando S, Perego R (2006) Fast and memory efficient mining of frequent closed itemsets. *IEEE Trans Knowl Data Eng* 18(1):21–36
16. Nguyen LTT, Trinh T, Nguyen NT, Vo B (2017) A method for mining top-rank-k frequent closed itemsets. *J Intell Fuzzy Syst* 32(2):1297–1305
17. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Efficient mining of association rules using closed itemset lattices. *Inf Syst* 24(1):25–46
18. Pietracaprina A, Vandin F (2004) Efficient incremental mining of top-k frequent closed itemsets. In: Tenth international conference discovery science. Springer, Berlin, pp 275–280
19. Pyun G, Yun U (2014) Mining top-k frequent patterns with combination reducing techniques. *Appl Intell* 41(1):76–98
20. Pyun G, Yun U, Ryu KH (2014) Efficient frequent pattern mining base on linear prefix tree. *Knowl-Based Syst* 55:125–139
21. Sahoo J, Das AK, Goswami A (2015) An effective association rule mining scheme using a new generic basis. *Knowl Inf Syst* 43(1):127–156
22. Saif-Ur-Rehman, Ashraf J, Salam AHA (2016) Top-k miner: top-k identical frequent itemsets discovery without user support threshold. *Knowl Inf Syst* 48(3):741–762
23. Tzvetkov P, Yan X, Han J (2005) TSP: mining top-k closed sequential patterns. *Knowl Inf Syst* 7(4):438–457
24. Vo B, Le B (2009) Mining traditional association rules using frequent itemsets lattice. In: International conference on computers & industrial engineering. IEEE Press, pp 1401–1406
25. Vo B, Le B (2011) Interestingness measures for association rules: combination between lattice and hash tables. *Expert Syst Appl* 38(9):11630–11640
26. Vo B, Hong TP, Le B (2012) DBV-miner: a dynamic bit-vector approach for fast mining frequent closed itemsets. *Expert Syst Appl* 39(8):7196–7206
27. Vo B, Hong TP, Le B (2013) A lattice-based approach for mining most generalization association rules. *Knowl-Based Syst* 45:20–30
28. Webb GI, Zhang S (2005) K-optimal rule discovery. *Data Min Knowl Disc* 10(1):39–79
29. Webb GI (2011) Filtered top-k association discovery. *WIREs Data Min Knowl Discovery* 1(3):183–192
30. You Y, Zhang J, Yang Z, Liu G (2010) Mining top-k fault tolerant association rules by redundant pattern disambiguation in data streams. In: International conference intelligent computing and cognitive informatics. IEEE Press, pp 470–473
31. Zaki MJ (2004) Mining non-redundant association rules. *Data Min Knowl Disc* 9(3):223–248