Finding minimal rare itemsets in a depth-first manner

Laszlo Szathmary[1], Petko Valtchev[2], Amedeo Napoli[3], and Robert Godin[2]

[1] University of Debrecen, Faculty of Informatics, Department of IT, H-4010 Debrecen, Pf. 12, Hungary
`Szathmary.L@gmail.com`

[2] Dépt. d'Informatique UQAM, C.P. 8888, Succ. Centre-Ville, Montréal H3C 3P8, Canada
`{valtchev.petko, godin.robert}@uqam.ca`

[3] LORIA UMR 7503, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
`napoli@loria.fr`

Abstract. Rare itemsets are an important sort of patterns that have a wide range of practical applications. Although mining rare patterns poses specific algorithmic problems, it is yet insufficiently studied. In a previous work, we proposed a levelwise approach for rare itemset mining. Here, we examine the benefits of depth-first methods for that task as such methods are known to outperform the levelwise ones in many practical cases.

1 Introduction

Pattern mining is a basic data mining task whose aim is to uncover the hidden regularities in a set of data [1]. As a simplifying hypothesis, the overwhelming majority of pattern miners chose to look exclusively on item combinations that are sufficiently frequent, i.e., observed in a large enough proportion of the transactions. Yet such a hypothesis fails to reflect the entire variety of situations in data mining practice [2]. In some specific situations, frequency may be the exact opposite of pattern interestingness. The reason behind is that in these cases, the most typical item combinations from the data correspond to widely-known and well-understood phenomena. In contrast, less frequently occurring patterns may point to unknown or poorly studied aspects of the underlying domain [2].

In a previous paper [3], we proposed a bottom-up, levelwise approach that traverses the frequent zone of the search space. In this paper we are looking for a more efficient manner for traversing the frequent part of the Boolean lattice, using a depth-first method. Indeed, depth-first frequent pattern miners have been shown to outperform breadth-first ones on a number of datasets.

2 Basic Concepts

Consider the following 6×5 sample dataset: $\boldsymbol{\mathcal{D}} = \{(1,\ ABCDE), (2,\ BCD), (3,\ ABC), (4,\ ABE), (5,\ AE), (6,\ DE)\}$. Throughout the paper, we will refer to this example as **"dataset $\boldsymbol{\mathcal{D}}$"**.

Consider a set of *objects* or *transactions* $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, a set of *attributes* or *items* $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, and a relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$. A set of items is called an *itemset*. Each transaction has a unique identifier (*tid*), and a set of transactions is called a *tidset*. The tidset of all transactions sharing a given itemset X is its *image*, denoted by $t(X)$. The *length* of an itemset X is $|X|$, whereas an itemset of length i is called an i-itemset. The (absolute) *support* of an itemset X, denoted by $supp(X)$, is the size of its image, i.e. $supp(X) = |t(X)|$.

The lattice is separated into two segments or zones through a user-provided "minimum support" threshold, denoted by *min_supp*. Thus, given an itemset X, if $supp(X) \geq min\_supp$, then it is called *frequent*, otherwise it is called *rare* (or *infrequent*). In the lattice in Figure 1, the two zones corresponding to a support threshold of 2 are separated by a solid line. The rare itemset family and the corresponding lattice zone is the target structure of our study.

Definition 1. X subsumes Z*, iff* $X \supset Z$ *and* $supp(X) = supp(Z)$ [4].

Definition 2. *An itemset Z is* generator *if it has no proper subset with the same support.*

Property 1. Given $X \subseteq \mathcal{A}$, if X is a generator, then $\forall Y \subseteq X$, Y is a generator, whereas if X is not a generator, $\forall Z \supseteq X$, Z is not a generator [5].

Proposition 1. *An itemset X is a generator iff* $supp(X) \neq min_{i \in X}(supp(X \setminus \{i\}))$ [6].

Each of the frequent and rare zones is delimited by two subsets, the maximal elements and the minimal ones, respectively. The above intuitive ideas are formalized in the notion of a border introduced by Mannila and Toivonen in [7]. According to their definition, the maximal frequent itemsets constitute the *positive border* of the frequent zone[1] whereas the minimal rare itemsets form the *negative border* of the same zone.

Definition 3. *An itemset is a* maximal frequent itemset *(MFI) if it is frequent but all its proper supersets are rare.*

Definition 4. *An itemset is a* minimal rare itemset *(mRI) if it is rare but all its proper subsets are frequent.*

The levelwise search yields as a by-product all mRIs [7]. Hence we prefer a different optimization strategy that still yields mRIs while traversing only a subset of the frequent zone of the Boolean lattice. It exploits the minimal generator status of the mRIs. By Property 1, frequent generators (FGs) can be traversed in a levelwise manner while yielding their negative border as a by-product. It is enough to observe that mRIs are in fact generators:

Proposition 2. *All minimal rare itemsets are generators* [3].

[1] The frequent zone contains the set of frequent itemsets.

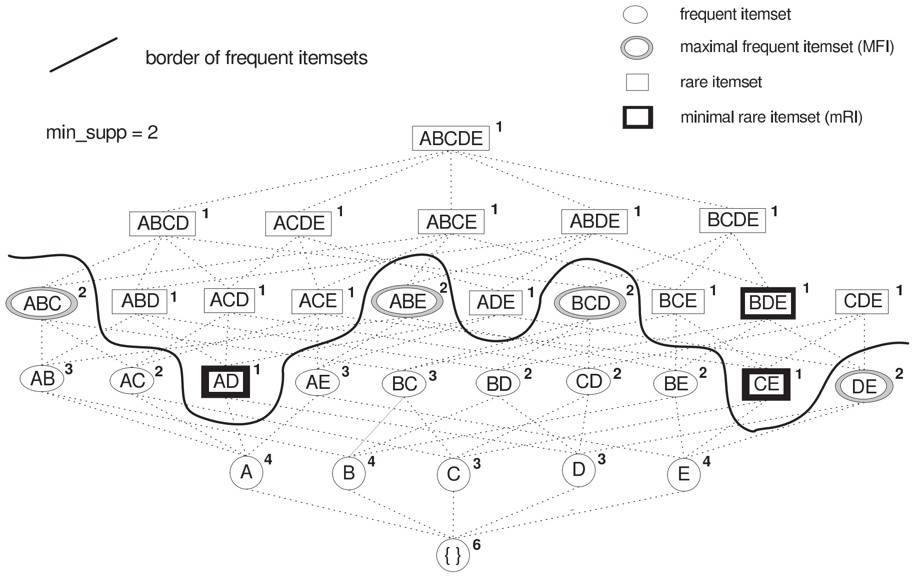

Fig. 1. The powerset lattice of dataset $\boldsymbol{\mathcal{D}}$.

Finding Minimal Rare Itemsets in a Levelwise Manner

As pointed out by Mannila and Toivonen [7], the easiest way to reach the negative border of the frequent itemset zone, i.e., the mRIs, is to use a levelwise algorithm such as *Apriori*. Indeed, albeit a frequent itemset miner, *Apriori* yields the mRIs as a by-product.

Apriori-Rare [3] is a slightly modified version of *Apriori* that retains the mRIs. Thus, whenever an i-long candidate survives the frequent $i-1$ subset test, but proves to be rare, it is kept as an mRI.

MRG-Exp [3] produces the same output as *Apriori-Rare* but in a more efficient way. Following Proposition 2, *MRG-Exp* avoids exploring all frequent itemsets: instead, it looks after frequent generators only. In this case mRIs, which are rare generators as well, can be filtered among the negative border of the frequent generators. The output of *MRG-Exp* is identical to the output of *Apriori-Rare*, i.e. both algorithms find the set of mRIs.

3 Finding Minimal Rare Itemsets in a Depth-First Manner

Eclat [8] was the first FI-miner to combine the vertical encoding with a depth-first traversal of a tree structure, called IT-tree, whose nodes are $X \times t(X)$ pairs. *Eclat* traverses the IT-tree in a depth-first manner in a pre-order way, from left-to-right [8] (see Figure 2).

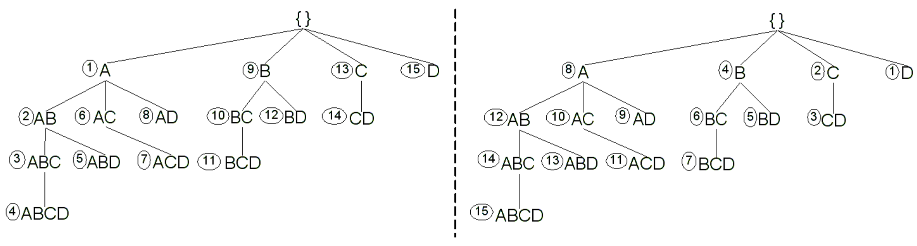

Fig. 2. Left: pre-order traversal with *Eclat*; **Right:** reverse pre-order traversal with *Eclat*. The direction of traversal is indicated in circles

3.1 Talky-G

Talky-G [9] is a vertical FG-miner following a depth-first traversal of the IT-tree and a right-to-left order on sibling nodes. *Talky-G* applies an inclusion-compatible traversal: it goes down the IT-tree while listing sibling nodes from right-to-left and not the other way round as in *Eclat*.

The authors of [10] explored that order for mining calling it *reverse pre-order*. They observed that for any itemset X its subsets appear in the IT-tree in nodes that lay either higher on the same branch as $(X, t(X))$ or on branches to the right of it. Hence, depth-first processing of the branches from right-to-left would perfectly match set inclusion, i.e., all subsets of X are met before X itself. While the algorithm in [10] extracts the so-called non-derivable itemsets, *Talky-G* uses this traversal to find the set of frequent generators. See Figure 2 for a comparison of *Eclat* and its “reversed” version.

3.2 Walky-G

Since *Walky-G* is an extension of *Talky-G*, we also present the latter algorithm at the same time. *Walky-G*, in addition to *Talky-G*, retains rare itemsets and checks them for minimality.

Hash structure. In *Walky-G* a hash structure is used for storing the already found frequent generators. This hash, called $fgMap$, is a simple dictionary with key/value pairs, where the key is an itemset (a frequent generator) and the value is the itemset's support.[2] The usefulness of this hash is twofold. First, it allows a quick look-up of the proper subsets of an itemset with the same support, thus the generator status of a frequent itemset can be tested easily (see Proposition 1). Second, this hash is also used to look-up the proper subsets of a minimal rare candidate. This way rare but non-minimal itemsets can be detected efficiently.

Pseudo code. Algorithm 1 provides the main block of *Walky-G*. First, the IT-tree is initialized, which involves the creation of the root node, representing

[2] In our implementation we used the `java.util.HashMap` class for $fgMap$.

Algorithm 1 (main block of Walky-G):

```
 1) // for quick look-up of (1) proper subsets with the same support
 2) // and (2) one-size smaller subsets:
 3) $fgMap \leftarrow \emptyset$   // key: itemset (frequent generator); value: support
 4)
 5) root.itemset $\leftarrow \emptyset$   // root is an IT-node whose itemset is empty
 6) // the empty set is included in every transaction:
 7) root.tidset $\leftarrow$ {all transaction IDs}
 8) $fgMap$.put($\emptyset, |\mathcal{O}|$)   // the empty set is an FG with support 100%
 9) loop over the vertical representation of the dataset ($attr$) {
10)       if ($min\_supp \leq attr$.supp $< |\mathcal{O}|$) {
11)             // $|\mathcal{O}|$ is the total number of objects in the dataset
12)             root.addChild($attr$)   // attr is frequent and generator
13)       }
14)       if ($0 < attr$.supp $< min\_supp$) {
15)             saveMri($attr$)   // attr is a minimal rare itemset
16)       }
17) }
18) loop over the children of root from right-to-left ($child$) {
19)       saveFg($child$)   // the direct children of root are FGs
20)       extend($child$)   // discover the subtree below child
21) }
```

the empty set (of 100% support, by construction). *Walky-G* then transforms the layout of the dataset in vertical format, and inserts below the root node all 1-long frequent itemsets. Such a set is an FG whenever its support is less than 100%. Rare attributes (whose support is less than min_supp) are minimal rare itemsets since all their subsets (in this case, the empty set) are frequent. Rare attributes with support 0 are not considered.

The `saveMri` procedure processes the given minimal rare itemset by storing it in a database, by printing it to the standard output, etc. At this point, the dataset is no more needed since larger itemsets can be obtained as unions of smaller ones while for the images intersection must be used.

The `addChild` procedure inserts an IT-node under a node. The `saveFg` procedure stores a given FG with its support value in the hash structure $fgMap$.

In the core processing, the `extend` procedure (see Algorithm 2) is called recursively for each child of the root in a right-to-left order. At the end, the IT-tree contains all FGs. Rare itemsets are verified during the construction of the IT-tree and minimal rare itemsets are retained. The `extend` procedure discovers all FGs in the subtree of a node. First, new FGs are tentatively generated from the right siblings of the current node. Then, certified FGs are added below the current node and later on extended recursively in a right-to-left order.

The `getNextGenerator` function (see Algorithm 3) takes two nodes and returns a new FG, or "null" if no FG can be produced from the input nodes. In addition, this method tests rare itemsets and retains the minimal ones. First, a candidate node is created by taking the union of both itemsets and the intersection of their respective images. The input nodes are thus the candidate's

Algorithm 2 ("extend" procedure):

```
Method: extend an IT-node recursively (discover FGs in its subtree)
Input:    an IT-node ($curr$)

1) loop over the right siblings of $curr$ from left-to-right ($other$) {
2)     $generator \leftarrow$ getNextGenerator($curr, other$)
3)     if ($generator \neq$ null) then $curr$.addChild($generator$)
4) }
5) loop over the children of $curr$ from right-to-left ($child$) {
6)     saveFg($child$)   // child is a frequent generator
7)     extend($child$)   // discover the subtree below child
8) }
```

parents. Then, the candidate undergoes a frequency test (test 1). If the test fails then the candidate is rare. In this case, the minimality of the rare itemset *cand* is tested. If all its one-size smaller subsets are present in $fgMap$ then *cand* is a minimal rare generator since all its subsets are FGs (see Property 1). From Proposition 2 it follows that an mRG is an mRI too, thus *cand* is processed by the `saveMri` procedure. If the frequency test was successful, the candidate is compared to its parents (test 2): if its tidset is equivalent to a parent tidset, then the candidate cannot be a generator. Even with both outcomes positive, an itemset may still not be a generator as a subsumed subset may lay elsewhere in the IT-tree. Due to the traversal strategy in *Walky-G*, all generator subsets of the current candidate are already detected and the algorithm has stored them in $fgMap$ (see the `saveFg` procedure). Thus, the ultimate test (test 3) checks whether the candidate has a proper subset with the same support in $fgMap$. A positive outcome disqualifies the candidate.

This last test (test 3) is done in Algorithm 4. First, one-size smaller subsets of *cand* are collected in a list. The two parents of *cand* can be excluded since *cand* was already compared to them in test 2 in Algorithm 3. If the support value of one of these subsets is equal to the support of *cand*, then *cand* cannot be a generator. Note that when the one-size smaller subsets are looked up in $fgMap$, it can be possible that a subset is missing from the hash. It means that the missing subset was tested before and turned out to subsume an FG, thus the subset was not added to $fgMap$. In this case *cand* has a non-FG subset, thus *cand* cannot be a generator either (by Property 1).

Candidates surviving the final test in Algorithm 3 are declared FG and added to the IT-tree. An unsuccessful candidate X is discarded which ultimately prevents any itemset Y having X as a prefix to be generated as candidate and hence substantially reduces the overall search space. When the algorithm stops, all frequent generators (and *only* frequent generators) are inserted in the IT-tree *and* in the $fgMap$ structure. Furthermore, upon the termination of the algorithm, all minimal rare itemsets have been found. For a running example, see Figure 3.

Algorithm 3 ("getNextGenerator" function):

```
Method: create a new frequent generator *and* filter minimal rare itemsets
Input:  two IT-nodes ($curr$ and $other$)
Output: a frequent generator or null

 1) $cand$.itemset ← $curr$.itemset ∪ $other$.itemset
 2) $cand$.tidset ← $curr$.tidset ∩ $other$.tidset
 3) if (cardinality($cand$.tidset) < $min\_supp$)   // test 1: frequent?
 4) {    // now cand is an mRI candidate; let us test its minimality:
 5)       if (all one-size smaller subsets of $cand$ are in $fgMap$) {
 6)             saveMri($cand$)   // cand is an mRI, save it
 7)       }
 8)       return null   // not frequent
 9) }
10) // else, if it is frequent; test 2:
11) if (($cand$.tidset = $curr$.tidset) or ($cand$.tidset = $other$.tidset)) {
12)       return null   // not a generator
13) }
14) // else, if it is a potential frequent generator; test 3:
15) if (candSubsumesAnFg($cand$)) {
16)       return null   // not a generator
17) }
18) // if cand passed all the tests then cand is a frequent generator
19) return $cand$
```

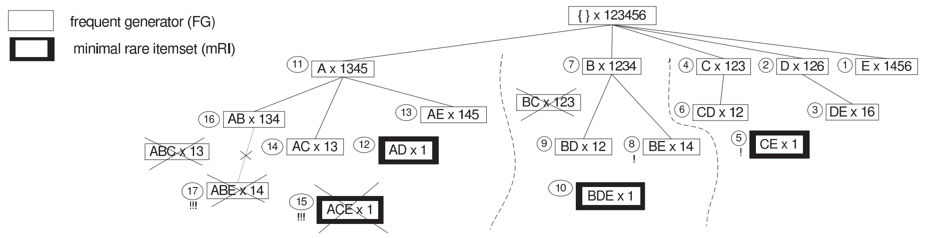

Fig. 3. The IT-tree built during the execution of *Walky-G* on dataset $\mathcal{D}$ with $min\_supp = 2$ (33%). Notice the two special cases: ACE is not an mRI because of CE; ABE is not an FG because of BE.

4 Conclusion

We presented an approach for rare itemset mining from a dataset that splits the problem into two tasks. Our new algorithm, *Walky-G*, limits the traversal of the frequent zone to frequent generators *only*. Our approach breaks with the dominant levelwise algorithmic schema since the traversal is achieved through a depth-first strategy.

Algorithm 4 ("candSubsumesAnFg" function):

Method: verify if *cand* subsumes an already found FG
Input: an IT-node (*cand*)

```
 1) subsets ← {one-size smaller subsets of cand minus the two parents}
 2) loop over the elements of subsets (ss) {
 3)     if (ss is stored in fgMap) {
 4)         stored_support ← fgMap.get(ss)   // get the support of ss
 5)         if (stored_support = cand.support) {
 6)             return true   // case 1: cand subsumes an FG
 7)         }
 8)     }
 9)     else   // if ss is not present in fgMap
10)     {   // case 2: cand has a non-FG subset ⇒ cand is not an FG either
11)         return true
12)     }
13) }
14) return false   // if we get here then cand is an FG
```

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in knowledge discovery and data mining. American Association for Artificial Intelligence (1996) 307–328
2. Weiss, G.: Mining with rarity: a unifying framework. SIGKDD Explor. Newsl. **6**(1) (2004) 7–19
3. Szathmary, L., Napoli, A., Valtchev, P.: Towards Rare Itemset Mining. In: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '07). Volume 1., Patras, Greece (Oct 2007) 305–312
4. Zaki, M.J., Hsiao, C.J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: SIAM International Conference on Data Mining (SDM' 02). (Apr 2002) 33–43
5. Kryszkiewicz, M.: Concise Representations of Association Rules. In: Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery. (2002) 92–109
6. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining Frequent Patterns with Counting Inference. SIGKDD Explor. Newsl. **2**(2) (2000) 66–75
7. Mannila, H., Toivonen, H.: Levelwise Search and Borders of Theories in Knowledge Discovery. Data Mining and Knowledge Discovery **1**(3) (1997) 241–258
8. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. In: Proceedings of the 3rd International Conference on Knowledge Discovery in Databases. (August 1997) 283–286
9. Szathmary, L., Valtchev, P., Napoli, A., Godin, R.: Efficient Vertical Mining of Frequent Closures and Generators. In: Proc. of the 8th Intl. Symposium on Intelligent Data Analysis (IDA '09). Volume 5772 of LNCS., Lyon, France, Springer (2009) 393–404
10. Calders, T., Goethals, B.: Depth-first non-derivable itemset mining. In: Proceedings of the SIAM International Conference on Data Mining (SDM '05), Newport Beach, USA. (Apr 2005)