Efficient Vertical Mining of Minimal Rare Itemsets

Laszlo Szathmary\textsuperscript{1}, Petko Valtchev\textsuperscript{2}, Amedeo Napoli\textsuperscript{3}, and Robert Godin\textsuperscript{2}

\textsuperscript{1} University of Debrecen, Faculty of Informatics, Department of IT, H-4010 Debrecen, Pf. 12, Hungary
szathmary.laszlo@inf.unideb.hu

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

\textsuperscript{3} LORIA (CNRS - Inria NGE - Université de Lorraine) BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
napoli@loria.fr

Abstract. Rare itemsets are important sort of patterns that have a wide range of practical applications, in particular, in analysis of biomedical data. 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 that traverses the search space bottom-up and proceeds in two steps: (1) moving across the frequent zone until the minimal rare itemsets are reached and (2) listing all rare itemsets. As the efficiency of the frequent zone traversal is crucial for the overall performance of the rare miner, we are looking for ways to speed it up. 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. The new method relies on a set of structural results that helps save a certain amount of computation and eventually ensures it outperforms the current levelwise procedure.

1 Introduction

Pattern mining is a basic data mining task whose aim is to uncover the hidden regularities in a set of data records, called \textit{transactions} \cite{1}. These regularities typically manifest themselves as repeating co-occurrences of properties, or \textit{items}, in the transactions, i.e., item patterns. As there is a potentially huge number of patterns, quality measures are applied to filter only promising patterns, i.e., patterns of potential interest to the analyst.

Designing a faithful interestingness metric in a domain independent fashion is not realistic \cite{2}. Indeed, without an access to the semantics of the items or another source of domain knowledge, it is impossible for the mining tool to assess the real value behind a pattern. 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 trans-
actions. This roughly translates the intuition that significant regularities should
occur often within a dataset.

Yet such a hypothesis fails to reflect the entire variety of situations in data
mining practice [3]. More precisely, it ignores some of the key factors for the suc-
cess of the mining task, namely, the expectations of the analyst and further to
that, her/his knowledge of the dataset and of the domain it stems from. Indeed,
while an analyst with little or no knowledge of the dataset will most probably
be happy with the most frequent patterns thereof, a better informed one may
find them of little surprise and hence barely useful. More generally speaking,
in some specific situations, frequency may be the exact opposite of pattern in-
terestingness. The reason behind is that in these cases, the most typical item
combinations from the data correspond to widely-known and well-understood
phenomena, hence there is no point in presenting them to the analyst. In con-
trast, less frequently occurring pattern may point to unknown or poorly studied
aspects of the underlying domain [3].

The above schematic description fits to a wide range of mining situations
where biomedical data are involved. For instance, in pharmacovigilance, one
is interested in associating the drugs taken by patients to the adverse effects
the latter may present as a result (safety signals or drug interactions in the
technical language of the field). To do that, a now popular way is to mine the
databases of pharmacovigilance reports, where each individual case is thoroughly
described, for such associations. However, as the data is accumulated throughout
the years, the most frequent associations tend to translate well-known signals and
interactions. The new and potentially interesting associations are less frequent
and hence "hidden" behind these most often occurring combinations.

The problem of unraveling them is a non-trivial one: In [4], a method based
on frequent pattern mining has been shown to only be able of dealing with a small
proportion of the existing pharmacovigilance datasets. The main difficulty is that
the data cannot be advantageously segmented as the new signals/interactions
may appear in any record. Alternatively, the problem cannot be approached
as outlier detection as a potential manifestation of a new signal need not have
any exceptional characteristics. Moreover, in order for an association to be val-
ified, it must occur in at least a given minimal number of patient records
(typically, five). Yet mining all patterns with only this weak constraint results in
an enormously-sized output whereby the overwhelming majority brings no new
insights.

The conclusion we drew out of that study was that the not-as-frequent, or
rare, patterns need to be addressed by specially designed algorithms rather than
by standard frequent miners fed with lower enough support. Similar observations
have been made in the pattern mining literature more than half a decade ago [3].
Since that time, a variety of methods that target non-frequent datasets have been
published, most of them adapting the classical levelwise mining schema exempli-
fied by the Apriori algorithm [1] to various relaxations of the frequent itemset
and frequent association notions [5,6,7] (see [8] for a recent survey thereof).
In our own approach, we focus on limiting the computational effort dedicated to the traversal of irrelevant areas of the search space. In fact, as indicated above, the rare itemsets represent a band of the underlying Boolean lattice of all itemsets that is located “above” the frequent part thereof and “below” the exceptional part (itemsets that occur in a tiny number, possibly none, of transactions). Thus, in a previous paper [9], we proposed a bottom-up, levelwise approach that traverses the frequent zone of the search space either exhaustively or in a more parsimonious manner by listing uniquely frequent generator itemsets. We also provided a levelwise method for generating all rare itemsets up to the minimal frequency required by the task (could be one in the worst case).

In this paper we are looking for a more efficient manner for traversing the frequent part of the Boolean lattice. In fact, the rapidity of pinpointing the minimal rare itemsets turned out to be a dominant factor for the overall performance of the rare pattern miner. It is therefore natural to investigate manners to speed it up. Further to that idea, and breaking with the dominant levelwise algorithmic schema, we study a depth-first method. Indeed, depth-first frequent pattern miners have been shown to outperform breadth-first ones on a number of datasets. We therefore decided to check the potential benefits of the approach in the rare pattern case. To that end, we have shown a set of structural results that allows for a sound substitution within the overall rare pattern mining architecture. Our experimental results show that the new method is most of the time much faster than the previous one.

The main contribution of this paper is a new algorithm called Walky-G for mining minimal rare itemsets. The algorithm limits the traversal of the frequent zone to frequent generators only. This traversal is achieved through a depth-first strategy.

The remainder of the paper is organized as follows. We first recall the basic concepts of frequent/rare pattern mining and then summarize the key aspects of our own approach. Next, we present a set of structural results about the search space and the supporting structure of the depth-first traversal of the pattern space. Then, the depth-first frequent zone-traversal algorithm is described and its modus operandi illustrated. A comparative study of its performance to those of the current breadth-first methods is also provided. Finally, lessons learned and future work are discussed.

2 Basic Concepts

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

Consider a set of objects or transactions \( O = \{o_1, o_2, \ldots, o_m\} \), a set of attributes or items \( A = \{a_1, a_2, \ldots, a_n\} \), and a relation \( R \subseteq O \times A \). 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 $\text{supp}(X)$, is the size of its image, i.e. $\text{supp}(X) = |t(X)|$.

A lattice can be separated into two segments or zones through a user-provided “minimum support” threshold, denoted by $\text{min}_\text{supp}$. Thus, given an itemset $X$, if $\text{supp}(X) \geq \text{min}_\text{supp}$, 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 $\text{supp}(X) = \text{supp}(Z)$ [10].

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

Generators are also known as free-sets [11] and have been targeted by dedicated miners [12].

**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 [13].

**Proposition 1.** An itemset $X$ is a generator iff $\text{supp}(X) \neq \text{min}_{i \in X}(\text{supp}(X \setminus \{i\}))$ [14].

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 [15]. 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 [15]. 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 [9].

\(^1\) The frequent zone contains the set of frequent itemsets.
Finding Minimal Rare Itemsets in a Levelwise Manner

As pointed out by Mannila and Toivonen [15], 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* [9] 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* [9] 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* [16] 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 × t(X)* pairs. *Eclat* traverses the IT-tree in a depth-first manner in a pre-order way, from left-to-right [16,17] (see Figure 2).
3.1 Talky-G

Talky-G [18] 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 [19] 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 [19] 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

In this subsection we present the algorithm Walky-G, which is the main contribution of this paper. 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.\footnote{In our implementation we used the java.util.HashMap class for $fgMap$.} 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
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) root.itemset \( \leftarrow \emptyset \) // root is an IT-node whose itemset is empty
5) // the empty set is included in every transaction:
6) root.tidset \( \leftarrow \{ \text{all transaction IDs} \} \)
7) \( fgMap.put(\emptyset, |O|) \) // the empty set is an FG with support 100%
8) loop over the vertical representation of the dataset (attr)
9) if (\( \text{min supp} \leq \text{attr.supp} < |O| \))
10) root.addChild(attr) // attr is frequent and generator
11) end if
12) end loop
13) saveMri(attr) // attr is a minimal rare itemset
14) loop over the children of root from right-to-left (child)
15) saveFg(child) // the direct children of root are FGs
16) extend(child) // discover the subtree below child
17) end loop

Efficient Vertical Mining of Minimal Rare Itemsets 275

IT-tree is initialized, which involves the creation of the root node, representing
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 \( \text{min supp} \)) 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 \textbf{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 \textbf{addChild} procedure inserts an IT-node under a node. The \textbf{saveFg} pro-
dedure stores a given FG with its support value in the hash structure \( fgMap \).

In the core processing, the \textbf{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 \textbf{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 \textbf{getNextGenerator} function (see Algorithm 3) takes two nodes and re-
turns 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 in-
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 ← getNextGenerator(curr, other)
3)    if (generator ≠ 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) }

tersection of their respective images. The input nodes are thus the candidate’s 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 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 Experimental Results

In our experiments, we compared Walky-G against Apriori-Rare [9] and MRG-Exp [9]. The algorithms were implemented in Java in the CORON platform [20].³ The experiments were carried out on a bi-processor Intel Quad Core Xeon 2.33 GHz machine running under Ubuntu GNU/Linux with 4 GB of RAM. All times reported are real, wall clock times as obtained from the Unix time command

³ http://coron.loria.fr
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) } else \// if ss is not present in fgMap
9) { \// case 2: cand has a non-FG subset ⇒ cand is not an FG either
10) return true
11) }
12) }
13) return false \// if we get here then cand is an FG

between input and output. For the experiments we have used the following datasets: T20I6D100K, C20D10K, C73D10K, and MUSHROOMS. The T20\(^4\) is a sparse dataset, constructed according to the properties of market basket data that are typical weakly correlated data. The C20 and C73 are census datasets from the PUMS sample file, while the MUSHROOMS\(^5\) describes mushrooms characteristics. The last three are highly correlated datasets.

The execution times of the three algorithms are illustrated in Table 1. The table also shows the number of frequent itemsets, the number of frequent generators, the proportion of the number of FGs to the number of FIs, and the number of minimal rare itemsets. The last column shows the number of mRIs whose support values exceed 0.

The T20 synthetic dataset mimics market basket data that are typical sparse, weakly correlated data. In this dataset, the number of FIs is small and nearly all FIs are generators. Thus, MRG-Exp works exactly like Apriori-Rare, i.e. it has to explore almost the same search space. Though Walky-G needs to explore a search space similar to Apriori-Rare’s, it can perform much better due to its depth-first traversal.

In datasets C20, C73, and MUSHROOMS, the number of FGs is much less than the total number of FIs. Hence, MRG-Exp and Walky-G can take advantage of exploring a much smaller search space than Apriori-Rare. Thus, MRG-Exp and Walky-G perform much better on dense, highly correlated data. For example, on MUSHROOMS at min_supp \(= 10\%\), Apriori-Rare needs to extract 600,817 FIs, while MRG-Exp and Walky-G extract 7,585 FGs only. This means that MRG-Exp and Walky-G reduce the search space of Apriori-Rare to 1.26\%! The

\(^4\) http://www.almaden.ibm.com/software/quest/Resources/

\(^5\) http://kdd.ics.uci.edu/
Table 1. Response times of Apriori-Rare, MRG-Exp, and Walky-G.

<table>
<thead>
<tr>
<th>min_supp</th>
<th>execution time (sec.)</th>
<th># FIs</th>
<th># FGs</th>
<th># mRIs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Apriori-Rare</td>
<td>MRG-Exp</td>
<td>Walky-G</td>
<td>Apriori-Rare</td>
</tr>
<tr>
<td>T20D100K</td>
<td>3.25</td>
<td>3.24</td>
<td>1.61</td>
<td>7</td>
</tr>
<tr>
<td>0.75%</td>
<td>30.22</td>
<td>30.92</td>
<td>11.80</td>
<td>4,710</td>
</tr>
<tr>
<td>0.5%</td>
<td>49.30</td>
<td>48.82</td>
<td>15.89</td>
<td>26,836</td>
</tr>
<tr>
<td>0.25%</td>
<td>115.35</td>
<td>117.11</td>
<td>33.47</td>
<td>155,163</td>
</tr>
<tr>
<td>C20D10K</td>
<td>21.92</td>
<td>5.49</td>
<td>0.57</td>
<td>5,319</td>
</tr>
<tr>
<td>30%</td>
<td>56.43</td>
<td>9.70</td>
<td>0.62</td>
<td>20,239</td>
</tr>
<tr>
<td>10%</td>
<td>157.09</td>
<td>18.27</td>
<td>0.77</td>
<td>89,883</td>
</tr>
<tr>
<td>5%</td>
<td>366.34</td>
<td>28.35</td>
<td>0.93</td>
<td>352,611</td>
</tr>
<tr>
<td>2%</td>
<td>878.93</td>
<td>40.77</td>
<td>1.47</td>
<td>1,741,883</td>
</tr>
<tr>
<td>C73D10K</td>
<td>35.97</td>
<td>6.97</td>
<td>0.84</td>
<td>1,007</td>
</tr>
<tr>
<td>95%</td>
<td>453.93</td>
<td>48.65</td>
<td>0.90</td>
<td>13,463</td>
</tr>
<tr>
<td>85%</td>
<td>1,668.19</td>
<td>117.62</td>
<td>0.95</td>
<td>46,575</td>
</tr>
<tr>
<td>Mushrooms</td>
<td>40%</td>
<td>3.24</td>
<td>1.77</td>
<td>0.50</td>
</tr>
<tr>
<td>30%</td>
<td>9.39</td>
<td>3.09</td>
<td>0.51</td>
<td>2,587</td>
</tr>
<tr>
<td>15%</td>
<td>160.88</td>
<td>8.32</td>
<td>0.66</td>
<td>99,079</td>
</tr>
<tr>
<td>10%</td>
<td>676.53</td>
<td>13.22</td>
<td>0.76</td>
<td>600,817</td>
</tr>
</tbody>
</table>

advantages of the depth-first approach of Walky-G is more spectacular on dense datasets: the execution times, with the exception of one case in Table 1, are always below 1 second.

5 Conclusion and Future Work

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. This traversal is achieved through a depth-first strategy. Experimental results prove the interest of our method not only on dense, highly correlated datasets, but on sparse ones too. Our approach breaks with the dominant levelwise algorithmic schema and shows that it outperforms its current levelwise competitors.

References