RPP algorithm: a method for discovering interesting rare itemsets

Sadeq Darrab, David Broneske, and Gunter Saake

University of Magdeburg, Germany
{firstname.lastname}@ovgu.de

Abstract. The importance of rare itemset mining stems from its ability to discover unseen knowledge from datasets in real-life domains, such as identifying network failures, or suspicious behavior. There are significant efforts proposed to extract rare itemsets. The RP-growth algorithm outperforms previous methods proposed for generating rare itemsets. However, the performance of the RP-growth degrades on sparse datasets, and it is costly in terms of time and memory consumption. Hence, in this paper, we propose the RPP algorithm to extract rare itemsets. The advantage of the RPP algorithm is that it avoid time for generating useless candidate itemsets by omitting conditional trees as RP-growth does. Furthermore, our RPP algorithm uses a novel data structure, RN-list, for creating rare itemsets. To evaluate the performance of the proposed method, we conduct extensive experiments on sparse and dense datasets. The results show that the RPP algorithm is around an order of magnitude better than the RP-growth algorithm.

Keywords: Rare Itemset · RN-list · RPP

1 Introduction

Since the emergence of data mining, there is a plethora of methods introduced to extract frequent itemsets [2]. However, rare itemset mining (RIM) discovers unusual events (known as rare itemsets). Discovering rare itemsets provides useful information in many domains since it extracts an uncommon valuable knowledge from datasets. The importance of RIM is that it is widely used in many applications, such as market basket analysis [2], predicting telecommunication equipment failures [8], identifying fraudulent credit card transactions [9], medical diagnosis [10, **?**], and adverse drug reactions [15]. For instance, in the health care domain, frequently discovering (known) complications are less interesting than rarely (unexpected) complications that may be more important to be discovered as early as possible. Therefore, discovering unusual events (rare itemsets) is more interesting, and it comes into the focus since it may help us to avoid adverse consequences. Traditional mining methods extract rare itemsets by setting a very low support threshold, which leads to an over-generation of itemsets. Thus, analyzing a large amount of itemsets is computationally expensive. To overcome this issue, a high support threshold is used, which leads to a loss of interesting rare itemsets. Thus, extracting rare itemsets is a challenge, and it is called a rare item problem [11]. To handle the rare item problem, there are various methods proposed [1, 3, 4, 6]. These methods can be classified based on their traversal of the search space into breadth-first search and depth-first search [20]. Breadth-first search methods use the most well-known algorithm called an apriori algorithm [2] to mine rare itemsets. These methods inherit apriori algorithm's drawbacks (overly huge candidate sets, redundant scans over the data). To overcome these shortcomings, depth-first search methods have been introduced to mine rare itemsets by utilizing an FP-tree structure [12]. A prominent FP-tree-based algorithm to mine rare itemsets is RP-growth [6]. The RP-growth algorithm solves the shortcomings of apriori-based algorithms in terms of time and memory consumption. However, it generates an unnecessary amount of conditional trees, which compromises search time and storage consumption in sparse datasets [14]. However, especially sparse datasets contain many uncommon patterns. Hence, how to design an efficient mining method for rare itemsets mining on sparse datasets is still critical research. In this paper, we propose the RPP algorithm for discovering rare itemsets. The RPP algorithm is inspired by a novel data structure, N-list [13], to generate interesting rare itemsets. The following steps summarize the contribution of this paper.

- An RPPC-tree is constructed to maintain all information needed to extract rare itemsets. It is constructed from transactions that contain at least one rare item.
- An RN-list of all interesting rare items is created.
- The RN-list of rare items are used in the mining process to generate the whole set of rare itemsets by intersecting these lists.

The rest of the paper is organized as follows. The concept of rare itemset mining is presented in Section 2 and relevant literature is introduced in Section 3. The details of the proposed algorithm, the RPP algorithm, is explained in Section 4. Sections 5 and 6 present the performance results and conclusion, respectively.

2 Preliminaries

To understand the basic concepts of RIM, let us consider the following motivating example.
Motivating example: given a transaction dataset DB in Table 1, let maximum support threshold ($maxSup$) and rare support threshold ($minSup$) be 0.80, 0.40, respectively. The task of rare itemset mining is to extract the set of all rare itemsets with support not less than $minSup$ but also not exceeding $maxSup$.
Definition 1. Rare itemsets: An itemset X is called rare itemset if its occurrence frequency in a given dataset is less than the user given minimum support threshold, $freqSup$, such that $Sup(X) < freqSup$.
Definition 2. Interesting rare itemsets: An itemset X whose support satisfies the following conditions is called the interesting rare itemset:
$Sup(X) < maxSup \wedge Sup(X) \geq minSup$.

For instance, the itemset {ce : 2} in the motivating example is called interesting rare itemset since $Sup(X) = 0.40$ which is less than $maxSup$ and it is greater or equals to the $minSup$.

Table 1: A simple dataset.

TID	Items	Ordered items
1	a, b, c, d	b, c, a, d
2	b, d	b, d
3	a, b, c, e	b, c, a, e
4	c, d, e, h	c, d, e
5	a, b, c, g	b, c, a

3 Related Work

Rare itemset methods can be classified based on the exploration of the search space into two categories. The first category includes level-wise exploration methods that depend on the downward closure property for pruning uninteresting itemsets [1, 2]. The second category uses a level-depth exploration [6, 22].

3.1 Level-wise Exploration Methods

Level-wise exploration methods work similar to an apriori algorithm [2]. These methods generate k-itemsets (itemsets of cardinality k) with using $(k-1)$-itemsets by utilizing the downward closure property. In [1], an apriori-inverse algorithm is proposed to extract rare itemsets that have a support below a maximum support threshold ($maxSup$). It works similar to the apriori algorithm except that in the first step the apriori-inverse algorithm generates 1-itemsets that do not satisfy $maxSup$. In [3], two algorithms are presented to detect rare itemsets. In the first step, an apriori-rare algorithm identifies the minimal rare itemsets (itemsets that do not have any subset which is rare). In the second step, the minimal rare itemsets are used by an MRG-Exp algorithm as a seed for generating the whole set of rare itemsets. A method in [4] stated that exploring the itemset space in a bottom-up fashion is costly in term of time and memory consumption since it is common that rare itemsets are found at the top of the search space (i.e., in the last iteration steps). Instead, they have proposed AfRIM using a top-down traversal approach. AfRIM starts with the largest n-itemset that contains all unique items found in a dataset. It discovers the candidate n-1-itemset subsets from rare n-itemsets and collects the interesting rare itemsets. Still, AfRIM is a time-consuming method since it considers zero-itemsets in the mining process as it begins from the largest itemset that contains all distinct items in the dataset. To avoid zero-itemset generation, the rarity algorithm [5] is proposed for retrieving rare itemset. It works similar to AfRIM except that it starts from items in the longest transaction in the dataset.
In [16], the first breadth-first search algorithm MSapriori, is proposed to extract both frequent and rare itemsets under different support thresholds (MIS). The MSapriori algorithm and its optimizations [17–19] extract frequent itemsets including rare ones by assigning lower support thresholds for rare (infrequent) itemsets than for most common (frequent) itemsets. Hence, they work similarly to the former apriori [2] with the following major difference: It declares an itemset (frequent and rare) as an interesting itemset if its support satisfies the lowest MIS of items within it.

3.2 Level-depth Exploration Methods

The above methods use the formal apriori algorithm [1], which is computationally expensive since 1) they employ candidate generation and test fashion, and 2) redundant scans over the dataset for each new generated candidate itemset. Furthermore, these methods spend a huge amount of time searching for the whole candidate itemsets (including useless itemsets whose support is zero) in order to identify the rare itemsets. To overcome these problems, several methods [24, 25, 22, 23] are presented to discover most common (frequent) itemsets including unusual (rare itemsets). CFP-growth [24] and its optimization CFP-growth++ [25] algorithms scan the dataset once to build a CFP-tree. Then, they reconstruct the tree by employing several punning and merging techniques. Reconstruction of the tree is computationally expensive in terms of memory and time consumption. To overcome this shortcoming, the MISFP-growth algorithm [22] is proposed to extract frequent itemsets including rare ones under MIS. The MISFP-growth efficiently builds the tree without a need for the reconstruction phase. Unlike the FP-growth based methods, mis-eclat [23] utilizes a vertical representation of data to extract both frequent and rare itemsets. Although these methods address the rare itemset problem, they suffer from overly huge itemsets since they extract both frequent and rare itemsets.
Focusing only on rare itemsets, the RP-Tree algorithm [6] is proposed to extract rare itemsets without the expensive itemset generation and pruning steps. It utilizes a tree data structure, the FP-Tree [12]. The RP-Tree is based on FP-growth, which uses a divide-and-conquer approach to generate rare itemsets. However, for each rare item, RP-growth generates a conditional tree in the mining process for generating rare itemsets. The RP-growth method becomes costly when datasets are sparse since building RP-trees, which bases on conditional pattern, recurrently makes RP-growth inefficient. It is common that rare itemsets mostly occur in sparse datasets. Hence, how to design efficient mining methods for mining rare itemsets is still a critical research problem.

4 Mining rare itemsets with Rare Pre Post (RPP) algorithm

Inspired by PrePost algorithm in [13], we propose the rare pre post algorithm (RPP) to extract meaningful rare itemsets. The RPP involves three sequential phases as follows.

1. At the first step, we build a Rare Pre-Post Code tree (RPPC-tree) to store all information needed to extract rare itemsets. The RPPC-tree is constructed by adding transactions that contain at least one rare item. Each node in the RPPC-tree consists of four fields: item name, count, pre-order rank and post-order rank. Item name represents the name of the item that this node represents. Count registers the number of transactions that reaches this node. Pre-order and post-order ranks stand for the number of traversed nodes from the root to this node when using pre or post order traversal.

2. At the second step, we traverse the RPPC-tree to generate the rare pre post code (RPP-code) for each node in the RPPC-tree. For a node X in the RPPC-tree, the tuple of {`(X-pre-order, X-post-order): count`} is called the RPP-code of X. For performance reasons, the RPP-codes are sorted in an ascending order according to their pre-order values. The RPP-codes of nodes that contain the same item X in the RPPC-tree is called RN-list of X, which is generated in Step 3.
3. For the mining process, the RPP algorithm constructs RN-lists by considering only the rare items. The RPP method iteratively constructs RN-lists of rare itemsets of length k by intersecting RN-lists of rare itemsets of length k-1. Finally, this process is terminated when there is no more RN-list that can be intersected with.

To show how the RPP method works, let us consider our motivating example to illustrate the three required phases of RPP algorithm in the following.

4.1 Construction of the RPPC-tree

To hold the necessary information from a dataset, the RPPC-tree is constructed. The tree is built by scanning the dataset twice. In the first scan, the support of 1-itemsets is counted. In the second scan, we add a transaction into the tree if the transaction has at least one rare item (item with support no greater than *maxSup* and no less than *minSup*). The RPPC-tree looks like the RP-tree and it can be defined as follows.

Definition 3. The RPPC-tree is a tree structure with the following properties: It consists of one root node, which is labeled as "null", and a set of nodes as the children of the root. Except the root, each node in the RPPC-tree consists of four fields: item-name, count, pre-order, and post-order. Item name holds the name of the item that this node represents. Count registers the number of transactions that reaches this node. A pre-post rank of a node registers the pre-order and post-order for each node within the tree, i.e., its position when traversing the tree in pre-order or post-order fashion, respectively.

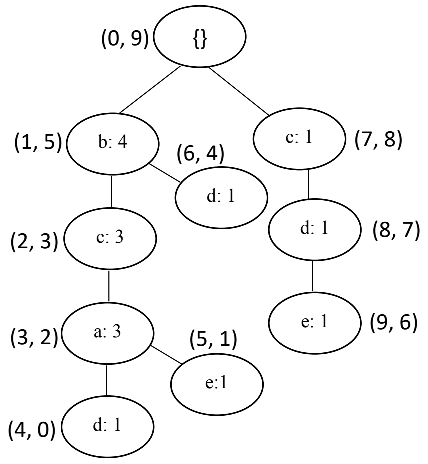


Fig. 1: RPPC-tree after adding all transactions in Table 1.

Following the motivating example, we show how the RPPC-tree is constructed. To build the RPPC-tree, the dataset is scanned twice. In the first scan, we calculate the support of the 1-items and eliminate useless items with support less than *minSup*. For instance, the items g, h are discarded since their support equals 1 which is less than *minSup* = 2. The interesting items are sorted in descending support order as in the right column in Table 1. In the second scan, the transactions in the right column in the Table 1 are used to construct the RPPC-tree. Figure 1 shows the final RPPC-tree after adding all transactions. As it can be seen from Figure 1, each node registers the item that this node represents, the number of transactions that reach this node, and the pre-post rank of the node. For instance, the node c, its item name is c, its count = 3 and (2, 3) is a pre-post rank of the node c.

4.2 Generating RN-list of Items

The main purpose of the RPPC-tree is to generate the RN-lists of items. The RN-lists hold all necessary information for discovering the whole set of rare itemsets. To construct the RN-lists of items, first, the pre-post codes are generated for each node in the RPPC-tree by traversing the tree in pre and post order. For a node X, its pre-post code is called RPP-code ((X.pre-order, X.post-order): count). For instance, the RPP-code of node a is ((3, 2): 3) indicating that a.pre-order is 3, a.post-order is 2, and its count is 3. The list of the RPP-codes of nodes that hold the same item X in the RPPC-tree is called the RN-list of X. The RPP-codes are sorted in ascending order of their pre-order values. For instance, a node (e: 2) in Figure 1 has two RPP-codes which are ((5, 1):1), and ((9, 6):1). Then, the RN-list of the itemset e is {((5, 1):1, ((9, 6):1)}. The advantage of the RN-list approach is its easy support calculation. For the RN-list of itemset X, which is denoted by (x1, y1): z1, (x2, y2): z2, . . . , (xm, ym) : zm, its support is z1 + z2 + · · · + zm. For instance, the RN-list of the itemset e are {((5, 1):1, ((9, 6):1)} and its support is 2. Following the motivating example, Table 2 shows the RN-list of all interesting 1-itemset.

Table 2: RN-lists of interesting rare items.

Item	RPP-codes	Support
b	{(1, 5): 4}	4
c	{(2, 3):3, (7, 8):1}	4
a	{(3, 2):3}	3
d	{(4, 0):1, (6, 4):1, (8,7):1}	3
e	{(5, 1):1, (9, 6):1}	2

4.3 Generation of Rare Itemsets

The rare itemsets can be generated by using the information that is contained in the RN-lists of itemsets. For RN-lists X, Y, the itemset XY can be generated if X is an ancestor of Y. We call a node X as an ancestor of node Y when:
X.pre-order < Y.pre-order and X.post-order > Y.post-order (i.e., we call it an ancestor-descendant relation). To generate the itemset XY, we traverse the RPP-codes of X and compare them with the RPP-codes of Y. Then, if the ancestor-descendant relation of X and Y holds (i.e., they are in the same path), we add the RPP-code of X to the RN-list of XY. For the support count, we add Y.count to the generated RN-list of XY since the items are sorted according to their descending support (i.e., the itemset X occurs together with itemset Y at most

Y.count). Following our motivating example, we illustrate this phase as follows. Table 2 contains the RN-list of all 1-itemsets. To shrink the search space, we compare the RN-list of itemsets with only the RN-list of rare items that satisfy both of $minSup = 2$ and $maxSup = 4$. Thus, the RN-lists of {a, d, e} are used to generate the whole set of rare itemsets. For instance, to generate rare itemset {ce}, we first check whether the support of item e satisfies Definition 3. We find out that $minSup = 2 \leq Sup(e) = 2 < maxSup = 4$. Then, we compare the RN-list of c with RN-list of e as follows.

1. The RN-list of the itemset c is {(2, 3): 3, (7, 8): 1}, and the RN-list of the itemset e is {(5, 1):1, (9, 6):1}. We compare each RPP-code of c with all RPP-codes of the itemset e to generate the itemset ce.
2. The RPP-code of c ((2, 3): 3) is compared with the first RPP-code of e ((5, 1):1). We notice that $2 < 5$ and $3 > 1$, which satisfies the ancestor-descendant relation. Then, we add the RPP-code {(2, 3): 1} to the RN-list of ce {(2, 3): 1}. Notice, we add the count of itemset e since it is descendant of c and cannot occur together more than e.count.
3. The RPP-code of c ((2, 3): 3) is compared with the next RPP-code of e ((9, 6):1). We find that $2 < 9$ and $3 < 6$, which does not satisfy the ancestor-descendant relation. Then, we would go for the next RPP-code of e. Since there is no further RPP-code of e, we traverse the next RPP-code of c.
4. The RPP-code of c ((7, 8): 1) is compared with the first RPP-code of e ((5, 1):1). We find that $7 > 5$ and $8 > 1$, which does not satisfy the ancestor-descendant relation. Hence, we do not add this RPP-code to the RN-list of itemset ce.
5. The RPP-code of c ((7, 8): 1) is compared with the next RPP-code of e ((9, 6):1). The ancestor-descendant relation $7 < 9$ and $8 > 1$, holds. Thus, we add ((7, 8): 1) to the RN-list of ce {(2, 3):1, (7, 8): 1}.
6. The resulted RN-list of the itemset ce is {(2, 3): 1, (7, 8): 1}.
7. The support of the itemset ce is 2. Thus, the itemset ce is interesting rare itemset since $minSup = 2 \leq sup(ce) = 2 < maxSup = 4$.

Similar to the above steps, the process is repeated for the remaining itemsets. The final rare itemsets for our example are {a: 3, e: 2, d: 2, ba:3, bd: 2, ca:3, cd: 2, ce: 2, bca: 3}. We generate rare itemsets only from rare items {a, d, e}. For instance, the RN-list of the itemset c will not be compared with the RN-list of b since the support of the itemset b is not less than the $maxSup = 4$ value.

5 Experimental results

To measure the performance of the RRP algorithm, we compare its performance with the state of art algorithm for mining rare itemset, RP-growth [6]. We carried out several experiments on four real-world datasets: Mushroom, Retail, Pumsb, and Kosarak. Both, sparse datasets (Kosarak, Retail), and dense datasets (Mushroom, Pumsb) are used for the evaluation process. The characteristics of the datasets are summarized in Table 3. For each dataset, the number of transactions, the number of distinct items, and an average transaction are denoted by # of Trans, # of items, and AvgTrans, respectively. The last column in Table 3 shows the density of the datasets. The datasets are downloaded from FIMI [7]. The experiments run on windows 10, 64 bit operating system, Intel Core i7-7700HQ CPU 2.80 GHz with 16 GB main memory, and 1 TB hard disk. The algorithms are implemented in Java to have a common implementation environment. The source code of RP-growth is downloaded from [21].

Table 3: The characteristics of the datasets.

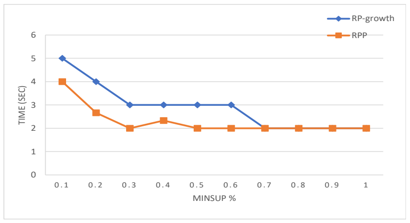
Dataset	Size (MB)	# Items	# Trans	AvgTrans	MaxSup	MinSup (%)
Mushroom	19.3	119	8124	23	0.01	{0.1, 0.2,. . . , 0.9}
Retail	4.2	16,470	88,126	10.3	0.1	{0.1, 0.2,. . . , 1}
Pumsb	16.3	2,113	49046	74	0.8	{52.5, 55, . . . , 70}
Kosarak	30.5	41,271	990,002	8.1	0.01	{0.1, 0.2,. . . , 0.9}

5.1 Execution Time

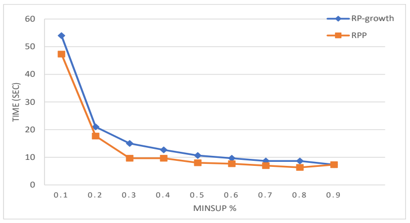
We compare our RPP algorithm with RP-growth to evaluate the execution time on all the datasets in Table 3. For all experiments, we use two support thresholds (*maxSup* and *minSup*) to extract rare itemsets. For each experiment, we fix the *maxSup* support threshold and vary the minimum rare support threshold (*minSup*), as shown in columns ($maxSup, minSup$) in Table 3. The interesting rare itemsets should be less than *maxSup* and greater or equal to *minSup*. In each graph, the X-axis represents the varied values of *minSup*, whereas the Y-axis stands for the execution time. Fig. 2(a)-(d) show the performance of the proposed algorithm, RPP, and RP-growth algorithm in terms of runtime. The graphs show that our RPP algorithm outperforms RP-growth for all datasets. The advantage of our RPP algorithm is that it utilizes only RN-lists of rare itemsets to generate rare itemsets. In contrast, the RP-growth is costly since it builds conditional trees for each rare item during the mining process. It can be noticed from the graphs that the RPP algorithm is orders of magnitude faster than RP-growth at low *minSup* values. This is a significant improvement since most of the interesting rare itemsets can be generated with low *minSup* value. It can be observed from the graphs that the performance is approximately the same when increasing the *minSup* value. They gained the same performance when the *minSup* value increased since the difference between *maxSup* and *minSup* is decreased, and a small number of rare itemsets will be generated.

5.2 Memory Consumption

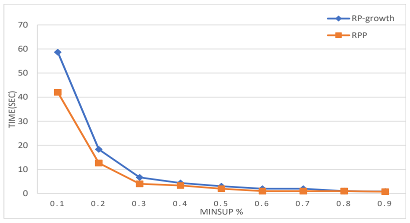
To evaluate the memory cost, we use the same factors that are shown in columns ($maxSup, minSup$) in Table 3. Fig. 3(a)-(d) show the memory cost of RPP and



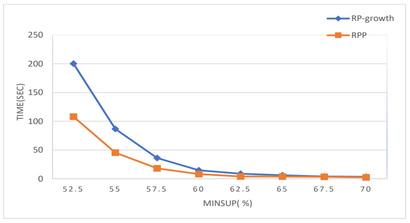
(a) Execution time for Retail dataset



(b) Execution time for Kosarak dataset



(c) Execution time for Mushroom dataset



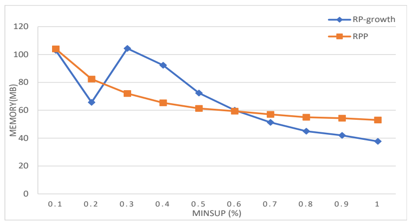
(d) Execution time for Pumsb dataset

Fig. 2: Execution time of RP-growth and our RPP algorithm for different datasets.

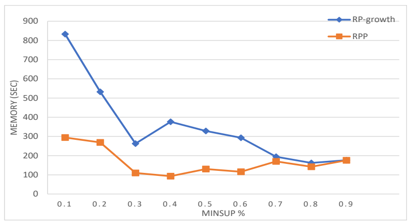
RP-growth algorithms on all datasets at different *minSup* values. Similar to the execution time figures, the *maxSup* value is fixed, while the *minSup* value is changed. In all figures, *minSup* values are located at X-axes, whereas the Y-axes represent the memory consumption by both algorithms. Fig. 3(a)-(b) show the memory cost of RPP algorithm and RP-growth algorithm on the retail and Kosarak datasets. The graphs show that our RPP algorithm consumes less memory than the RP-growth algorithm for the sparse datasets. For the retail dataset, the RPP algorithm consumes less memory than the RP-growth when the *minSup* value exceeds 0.3 %, and it consumes a little more memory than the RP-growth when the *minSup* value exceeds 0.6 %. Fig. 3(b) shows the memory consumption of RPP and RP growth algorithms on the very sparse datasets, Kosarak. Notably, the RPP algorithm consumes a little less memory than RP-growth for all *minSup* values. For dense datasets (Mushroom, Pumsb), Fig. 3(c) and Fig. 3(d) show the memory consumption of RPP and RP-growth algorithms. The RPP algorithm consumes more memory than the RP-growth algorithm on Mushroom and Pumsb datasets.

5.3 Scalability

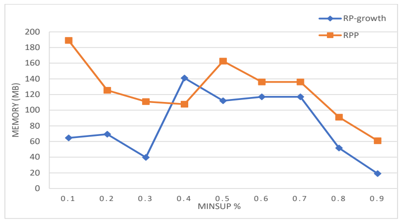
To evaluate the scalability of RPP algorithm and the RP-growth, we choose the largest dataset, Kosarak. It contains about 1 million transactions. The dataset is equally divided into ten parts. For each experiment, we add 10 % to the previous



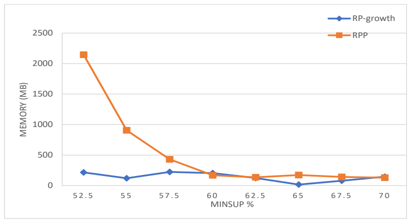
(a) Memory consumption for Retail dataset



(b) Memory consumption for Kosarak dataset



(c) Memory consumption for Mushroom dataset



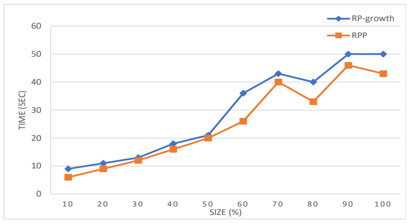
(d) Memory consumption for Pumsb dataset

Fig. 3: Memory consumption of RP-growth and our RPP algorithm for different datasets.

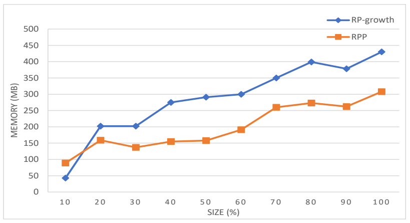
accumulative parts. Fig. 4(a)-(b) demonstrates the experimental results of the RPP and RP-growth algorithms in terms of time and memory consumption. The graphs illustrate that the proposed RPP algorithm scales better than RP-growth when increasing the size of the dataset. The RPP algorithm requires less time and memory since it depends on RN-lists of rare itemsets during the mining process. For RP-growth, it needs to traverse a big search space and generate a large number of conditional trees to generate rare itemsets. To sum up, our RPP algorithm is faster than RP-growth on all datasets that are given in Table 3. For memory consumption, our RPP algorithm requires less memory on sparse datasets, and it consumes more memory on dense datasets. Finally, when the size of the dataset increases, our RPP algorithm scales better than RP-growth.

6 Conclusion

In this paper, we proposed the RPP algorithm to discover the whole set of rare itemsets. The RPP algorithm utilizes RN-lists of items, which contains the whole information needed to generate rare itemsets. The RPP algorithm shrinks the search since 1) it avoids redundant scans of the dataset, 2) it extracts the whole set of interesting rare itemsets without generating conditional trees, and 3) the support of the generated itemsets is calculated by intersection operations. To test the performance of the RPP algorithm, we compared its performance with the well-known algorithm for rare itemsets mining on dense and sparse datasets, the



(a) Scalability of execution time.



(b) Scalability of memory consumption.

Fig. 4: Scalability of our RPP algorithm compared to RP-growth for the Kosarak dataset.

RP-growth algorithm. The experimental results showed that the RPP algorithm is significantly better than the RP-growth algorithm in terms of execution time, memory cost, and scalability.

References

1. Koh Y. S., and Rountree N., Finding sporadic rules using apriori-inverse, In Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Springer, pp. 97-106, 2005.
2. Agrawal R.,and Srikant R., Fast algorithms for mining association rules in large databases, In Proceedings of 20th International Conference on Very Large Data Bases(VLDB),VLDB, pp. 487–499, 1994.
3. Szathmary L., Napoli A., and Petko V.,Towards rare itemset mining, In 19th International Conference on Tools with Artificial Intelligence (ICTAI),IEEE , pp. 305-312, 2007.
4. Adda, M., Wu L., and Feng, Y, Rare itemset mining, In Sixth International Conference on Machine Learning and Applications (ICMLA),IEEE, pp. 73-80, 2007.
5. Troiano L., Scibelli G., and Birtolo C, A fast algorithm for mining rare itemsets, In Proceedings of 9th International Conference on Intelligent Systems Design and Applications,IEEE Computer Society Press, pp. 1149–1155, 2009.
6. Tsang S., Koh Y.S.,and Dobbie G., RP-Tree: Rare Pattern Tree Mining, In Data Warehousing and Knowledge Discovery (DaWaK) 2 Lecture Notes in Computer Science, Springer, pp. 277-288, 2011.
7. Frequent Itemset Mining Dataset Repository, http://fimi.uantwerpen.be/data/.
8. Bhatt U.Y., and Patel P.A. , An effective approach to mine rare items using Maximum Constraint, In Intelligent Systems and Control (ISCO), IEEE, pp.1-6, 2015.
9. Weiss, G. M., Mining with rarity: a unifying framework, ACM SIGKDD Explorations Newsletter, pp. 7-19, 2004.
10. Szathmary L., Valtchev P., Napoli, A.,and Godin R., Efficient Vertical Mining of Minimal Rare Itemsets. In Proceedings of 9th International Conference Concept Lattices and Their Applications(CLA), pp.269-280, 2012.
11. Lui, Chung-Leung, and Fu-Lai Chung. ”Discovery of generalized association rules with multiple minimum supports.” In European Conference on Principles of Data Mining and Knowledge Discovery, Springer, Berlin, Heidelberg, pp. 510-515,2000.

12. Han, J., Pei, J., and Yin Y. , Mining frequent patterns without candidate generation, In Proceedings of the International Conference on Management of Data(ACM SIGMOD), PP.1-12,2000.
13. Deng, Z., Wang, Z., and Jiang J., A new algorithm for fast mining frequent itemsets using N-lists, Science China Information Sciences, pp.2008-2030, 2012.
14. Borah A., and Nath B., Rare pattern mining: challenges and future perspectives, Complex and Intelligent Systems, pp.1-23, 2019.
15. Ji Y., Ying H., Tran J., Dews P., Mansour A., and Massanari R. M., A method for mining infrequent causal associations and its application in finding adverse drug reaction signal pairs, IEEE transactions on Knowledge and Data Engineering, pp.721-733, 2012.
16. Lui, C. L., and Chung F. L., Discovery of generalized association rules with multiple minimum supports, In European Conference on Principles of Data Mining and Knowledge Discovery (ECML PKDD), Springer,pp.510-515, 2000.
17. Xu T., and Dong, X., Mining frequent patterns with multiple minimum supports using basic Apriori, In Natural Computation (ICNC),IEEE, pp. 957-961,2013.
18. Kiran R. U., and Re P. K., An improved multiple minimum support based approach to mine rare association rules, In Computational Intelligence and Data Mining (CIDM), IEEE, pp. 340-347,2009.
19. Lee Y. C., Hong T. P., and Lin W. Y., Mining association rules with multiple minimum supports using maximum constraints, International Journal of Approximate Reasoning , pp. 44-54, 2005.
20. Darrab S,, David B., and Gunter S., Modern Application and Challenges for Rare Itemset Mining, in 8th International Conference on Knowledge Discovery, 2019.
21. Fournier-Viger P., Gomariz, A., Gueniche, T., Soltani A. Wu. C., and Tseng V. S.: SPMF: a Java Open-Source Pattern Mining Library. Journal of Machine Learning Research (JMLR), 15, pp. 3389-3393, 2014.
22. Darrab S., and Ergenç B., Frequent pattern mining under multiple support thresholds, the International Conference on Applied Computer Science (ACS), Wseas Transactions on Computer Research, pp. 1–10, 2016.
23. Darrab S., and Ergenç B., Vertical pattern mining algorithm for multiple support thresholds,International Conference on Knowledge Based and Intelligent Information and Engineering (KES), Procedia Computer Science 112, pp. 417-426, 2017.
24. Hu, Y. H., and Chen, Y. L., Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism, Decision Support Systems , pp. 1-24,2006.
25. Kiran, R. U., and Reddy P. K., Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms, In Proceedings of the international conference on extending database technology(EDBT), pp. 11-20, 2011.