




Discovery of Rare Itemsets Using Hyper-Linked Data Structure: A Parallel Approach

Goutham Yadavalli and Shwetha Rai^(✉) 

Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal
Academy of Higher Education, Manipal, Karnataka 576104, India
shwetha.raimanipal.edu

Abstract. Pattern mining has been more important in the solution of various data mining jobs over the years. The extraction of common patterns was the primary focus of pattern mining research for a long period of time, with the mining of rare patterns being neglected. Rare pattern mining is becoming more popular as researchers recognize the importance of rare patterns. The hyper-linked data structure is suitable to store sparse data set in the main memory and enables dynamic adjustment of links during the mining process using recursion. However, a sequential approach to discovering rare patterns from a large dataset is inefficient. Hence a CUDA-based parallel algorithm has been implemented to discover rare itemsets. The algorithm is tested using dense and sparse datasets on a GPU. The GPU initialization time affects the time taken to discover rare itemsets. The time taken to transfer data between CPU and GPU is significantly large and the parallel implementation of an algorithm with a recursive approach is unsuitable.

Keywords: CUDA · Data mining · Parallel programming · Rare itemset · Tree data structure

1 Introduction

Data mining has gained popularity since the beginning of the digital revolution due to its application in various fields. Alan Turing introduced the idea of a universal machine in 1936, which was one of the first instances of data mining since it could do computations like those performed by modern computers [1]. Data mining is increasingly being used by businesses to enhance anything from sales operations to financial analysis for investment reasons. It involves analyzing huge quantities of data to discover business insights that may help companies solve problems, reduce risks, and take advantage of new opportunities [2]. It requires sifting through enormous amounts of data to discover hidden value from the dataset. To get the best results, data mining requires a range of tools and techniques such as artificial intelligence, association rule mining (ARM), clustering, classification, machine learning and regression [2].

Pattern mining, a technique in ARM, has played an important part in various data mining activities throughout the years. For a long time, pattern mining research focused only on the extraction of common patterns, ignoring the mining of rare patterns. Rare patterns have been shown to be useful in a variety of applications, including network anomaly detection, equipment failure, medicine, and fraud detection [3]. Given the importance of rare patterns, rare pattern mining research is gaining traction, and a significant amount of effort has already been done to extract these significant patterns [4].

The term “frequent itemset mining” focuses on locating itemsets that appear in groups frequently [5]. Rare itemsets mining uncovers previously unknown relationship among the less frequent itemsets. All the algorithms in place to extract rare itemsets are serial algorithms which will be time inefficient with the increasing volume of data. Due to the nature of the task of finding rare itemsets, a certain part of the process, which is time and resource-consuming, can be implemented in parallel hence reducing the time taken to discover the rare itemsets. The research aims to understand the nature of the hyperlink data structure in a parallel environment by implementing parallel mining with the assistance of the CUDA framework.

2 Literature Review

Different algorithms that have been developed in a similar field are discussed with its pros and cons. Liu et al. [7] implemented rare pattern mining and assigned minimum support threshold to each of the items separately, utilizing the Apriori-based method. The rare itemset mining algorithms, ARIMA [6] and the AfRIM [8], on the other hand, utilized a single minimum support value for all components. The use of Apriori-based algorithms has several limitations which led to the development of tree-based methods, such as the Compressed FP-Growth (CFP-Growth) which uses a multiple minimum support framework to extract rare patterns. For mining rare patterns, the Rare Pattern Tree (RP-Tree) Mining method described by Tsang et al. [9] is the most efficient pattern growth technique. Only transactions with at least one rare item are considered by the system, which employs two support criteria. Bhatt et al. implemented MCRP-Tree algorithm [10] that improves it even further by utilizing several support levels for improved performance. It dynamically assigns suitable minimum support to each item, allowing for more efficient extraction of frequent itemsets including rare items than current methods. Rare pattern mining methods as discussed by Borah et al. [4] collect patterns from databases in a horizontal style, with one column representing the transaction id. The other, on the other hand, indicates the total number of objects involved in the transaction with that specific id. The Apriori algorithm is the most often used algorithm in pattern mining.

Most rare pattern mining methods are based on frequent itemset mining techniques, resulting in time-consuming candidate creation or traversal of all frequent patterns. To address the aforementioned difficulties, the author offers a novel algorithm that is based on top-down and depth-first approaches. For each transaction, a negative itemset is created, which includes all the items that are not in the itemset. The method presented by Lu et al. [11], the Negative Infrequent Itemset tree miner (NIIMiner), was implemented in Java. The negative itemset in NIIMiner gets excessively big for sparse datasets, when there is only one item in a transaction, resulting in extremely lengthy calculations.

The RP-growth method outperforms all prior algorithms in terms of finding rare itemsets, but its performance degrades for sparse data, and memory consumption rises. Rare Pre-Post is a novel algorithm proposed by Darrab et al. [12] (RPP). By eliminating conditional trees at each stage of the RP growth, it saves spending time and resources on worthless candidate itemsets. The memory usage of RPP was lower than that of the RP-growth algorithm for certain minSup criteria. The threshold values chosen for both minSup and maxSup have an impact on the outcomes of rare itemset mining by Kanimozhi et al. [13]. However, when these numbers are changed, the results vary dramatically, allowing the discovery of intriguing patterns. The itemsets are divided into three groups in this method, and each group's threshold is generated independently. The itemsets over the suggested threshold in each group are then grouped back together.

By using the principles of fuzzy theory, the author introduces us to FRI-Miner by Cui et al. [14], a fuzzy-based mining method that helps us discover intriguing, rare itemsets from a quantitative database. FRI-Miner is described in detail in this article. It is necessary to perform certain pruning procedures in each step-in order to improve the performance of the algorithm and to make better use of the available resources. This algorithm is more concerned with discovering fewer rare itemsets that are qualitative in nature rather than finding numerous rare itemsets, which is what the Automated Apriori algorithm is concerned with doing. The method has been thoroughly tested by changing various factors such as the minSup value and the density of the dataset, and the results have been documented for the varied runtimes for the various values. The most significant disadvantage of this method is the difficulty in determining the right minSup value since it must be set manually.

Recent advancements in Graphics Processing Units (GPUs) have made it possible to achieve low-cost high-performance computing for a wide range of general-purpose applications. A programming paradigm based on Compute Unified Device Architecture (CUDA) offers programmers with sufficient C language-like APIs that allow them to better utilize the parallel capabilities of the graphics processing unit (GPU). Data mining is extensively utilized and has substantial applications across a broad range of industries and disciplines. Current data mining toolkits, on the other hand, are unable to satisfy the performance requirements of applications that use large-scale databases in terms of speed. According to this paper [15], the author proposes three techniques for speeding up fundamental problems in data mining algorithms on the CUDA platform. These techniques are scalable thread scheduling scheme for irregular pattern, concurrent distributed top-k scheme, and concurrent large-scale high dimension reduction scheme. Adil et al. [16] propose a new CUDA based approach for association rule mining what has a 18x speedup over the serial algorithm for association rule mining.

3 Methodology

Various terminologies that are used to describe the algorithm are discussed. Furthermore, the algorithm, and its implementation details are discussed in this section.

3.1 Basic Terminologies

- **Itemset:** An itemset is a collection of items in the database T_d and it is represented as I . A set of n items in an itemset I is represented as $I = i_1, i_2, \dots, i_n$.
- **Database:** The database is a collection of m transactions represented as $T_d = t_1, t_2, \dots, t_m$ where t_1, t_2, \dots, t_m are transaction identifiers representing each transaction.
- **Support:** A support is a user-defined threshold to discover frequent and/or infrequent itemsets. The support of any itemset in a database with T_d transactions is calculated as the total number of times it has appeared in the transaction database.
- **Frequent Itemset:** An itemset is said to be frequent if its occurrence in the database is greater than or equal to the user-defined threshold minfreq .
- **Infrequent (or Rare) Itemset:** An itemset is said to be infrequent if its occurrence in the database is between two thresholds minrare and minfreq . If the occurrence of the itemset is less than minrare then it is considered noise.

3.2 Efficient Hyper-Linked Rare Pattern Structure Algorithm

Algorithm 1 computes the header table which consists of the different items present in the database transactions and their support counts, this table can be further used to find our rare and frequent itemsets. Algorithm 2 is used to generate the projected databases after discarding the frequent itemsets at each step to discover rare itemsets.

 Algorithm 1 : Efficient Hyper-Linked Rare Pattern Structure (EHLRPS)

Input: Complete original transaction database (DB), minrareSup, minfreqSup

Output: Complete set of rare itemsets

- 1 Create header table (HT) containing the following fields: itemid, supcount and hyperlink
 - 2 for each item $IT \in DB$ do
 - 3 if item IT is in the HT then
 - 4 support count(IT) \leftarrow support count(IT)+1
 - 5 Sort HT in ascending order of Sup_Count
 - 6 else if item IT not in the HT then
 - 7 create entry for item IT in HT
 - 8 support count(IT) \leftarrow 1
 - 9 hyperlink \leftarrow NULL
 - 10 end if
 - 11 end for each
 - 12 for each item IT in HT do
 - 13: if minrareSup < support count(IT) < minfreqSup
 - 14 Rare \leftarrow Rare \cup IT
 - 15 for each transaction $TR \in DB$ do
 - 16 if $\exists r | (\text{minrareSup} < r < \text{minfreqSup}) \wedge r \in TR$ then
 - 17 RareItemTransaction \leftarrow TR
 - 18 end if
 - 19 end for each
 - 20 Create different queues, Q_j with following fields: itemid and hyper-link that stores the items of j th RareItemTransaction. Use a hyperlink to link all transactions with the same first item.
 - 21 for each item x in HT
 - 22 create rare item projections from x -projected database()
 - 23 end for each
-

Algorithm 2: Create a x-projected database to mine rare itemsets:

```

Input: Main Header table HT, str
Output: Rare itemsets
1  for each item x in HT do
2    Traverse the queues attached through hyperlink
3    Create a sub-header HTsub with the items in the transaction queues and its
    corresponding support count.
4    Create a string 'str' to store the itemsets and its corresponding support
5    for each item in HTsub do
6      if minrareSup < HTsub.item_id.support < minreqSup
7        concatenate (x, HTsub.item_id)
8        concatenate (x, ": HTsub.item_id.support")
9      end if
10   end for
11   if HTsub.item.hyperlink is not NULL and if the item y in HTsub is Rare then
12     xy-projected database ()
13   end if
14 end for

```

The flowchart shown in Fig. 1 (a) describes the basic flow of the sequential algorithm. Initially, the data is read from the file and the transactions are copied into the main memory in a hyperlink data structure. In the next step, a support count map is created for the items in the transaction. This step is taken when the database is read once, to reduce multiple reads of the database. Next, the support count map is sorted based on the support count in ascending order. This sorted map is used to create the data structure HeaderTable, HT which contains the item, its support count, and a hyperlink which links to the transactions containing that item. The items are added to the results that have a support count which falls in the range of minSup and maxSup. In the next step, a sub HeaderTable is created for the items that have the support count in the required frequency. Further, a loop is run over these sub HeaderTables and this process is repeated for all the elements in the sub HeaderTable using a recursive function, and the rare itemsets are added to the results.

In the second algorithm, it can be noticed that the rare items are extracted for each entry of the HeaderTable HT in a loop, thus indicating that each entry in the HeaderTable can be processed separately. This is the main idea behind the design of the parallel algorithm. The initial preprocessing part which includes the algorithm -1 can be implemented in serial. We can process each entry of the HeaderTable in a different thread parallelly in CUDA. This process is done in the kernel code, and the further recursive calls made by each separate entry of the HeaderTable are executed on the same thread. All the results are collected in an array and displayed after the parallel execution is completed. The CUDA program can be divided into two parts, sequential instructions, and parallel instructions. In the sequential part, the input file is read and the initial HeaderTable is constructed. Then the contents of the HeaderTable are copied onto the GPU since the parallel part of the code cannot refer to the variables declared and initialized in the serial code. The parallel execution of the code begins from when the `__global__` function is

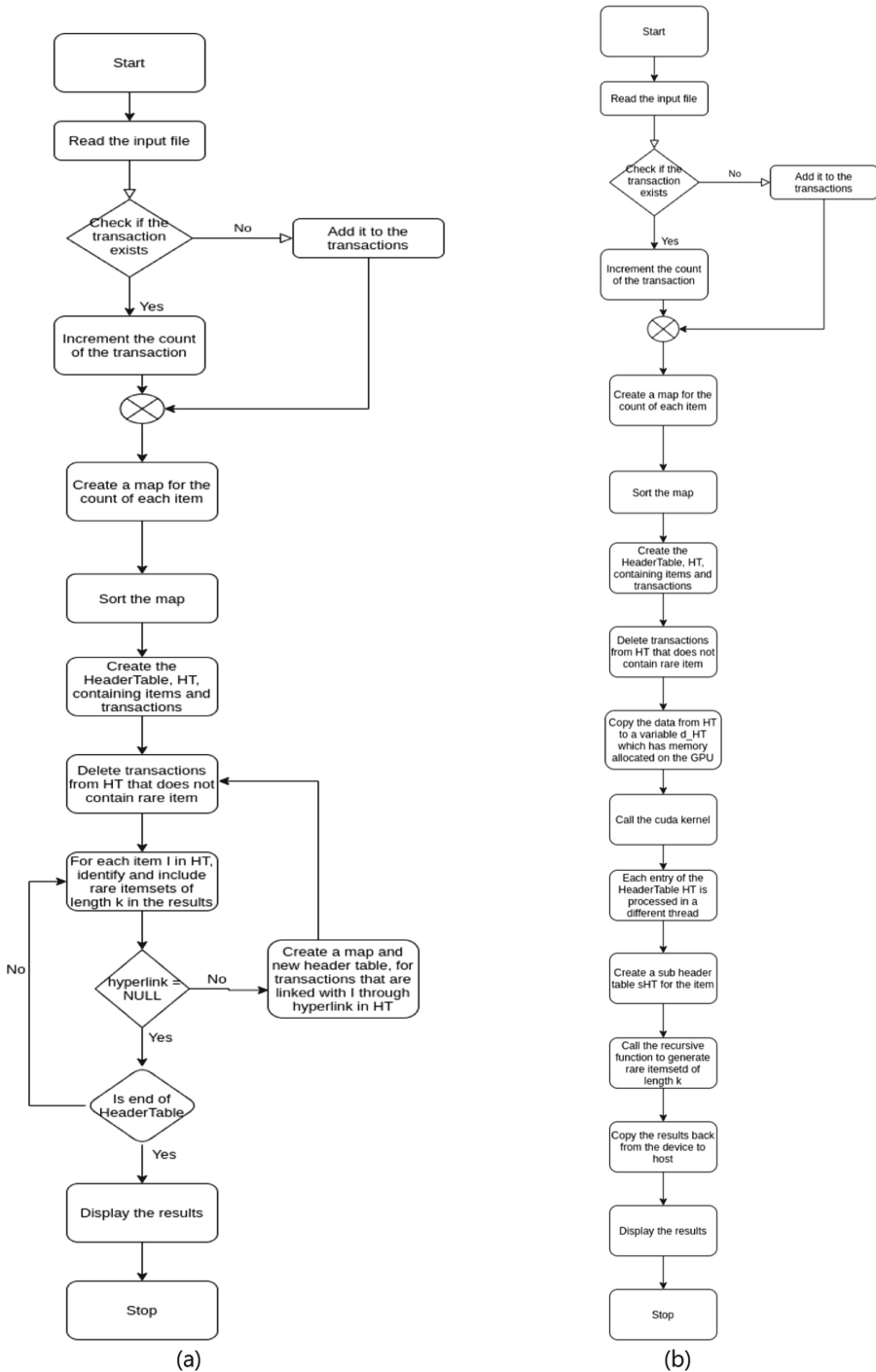


Fig. 1. Flowchart of (a) serial algorithm and (b) parallel algorithm

called. This function can only be called from the host code. Each of the items in the HeaderTable is used on a different thread to construct the sub header table and then process it recursively. This recursive process is done in a `__device__` function which can only be called from the `__global__` function. All the results are collected in an array and this array is copied back from the device to the host. The flowchart of the parallel approach is shown in Fig. 1 (b).

4 Result Analysis

The results are recorded by testing the algorithms against standard transaction databases used to test the performance of data mining algorithms. All the tests have been conducted on Chess and Connect datasets obtained from [17]. The number of transactions in chess data is 3,196 and tests have been conducted by varying the number of the transactions and the number of items in each transaction between 6 and 7. There are 67,557 transactions in the Connect dataset.

4.1 Performance of Sequential Approach

The serial algorithm is implemented in C++ and all the tests are conducted on a Linux machine running Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz processor. The compiler used is g++, which is a GNU-based C++ compiler.

The experiments are conducted by varying the number of rows in the database and the size of the transactions in the database. In Tables 1 and 2, we summarize the results that are recorded after running the code 5 times and the average execution time is noted down.

The following observations can be made from the results obtained as shown in Tables 1 and 2:

1. The execution time increases as the length of the transaction increases, this is because it increases the depth of the recursion as we generate more sub header tables.
2. The execution time also increases as we increase the number of transactions, but this observation has a lot of irregularities in it. This is because, as we store only the unique transactions, an increase in the size of the database might not always lead to growth in the size of our data structure as the new transactions might already exist in it. Also, it depends on the items falling in the range of `minSup` and `maxSup`. If the items that are rare have only a few transactions linked to it through the hyperlink, the depth of the recursion decreases, and the runtime decreases substantially because of this.

4.2 Performance of Parallel Approach Using CUDA

The parallel code has been developed in C++ using the CUDA library. The experiments have been conducted on the Google Colab environment which provides a GPU for

Table 1. Execution time(milliseconds) for Chess dataset using sequential and parallel approach

# Transactions	Chess 6		Chess 7	
	Sequential	Parallel	Sequential	Parallel
500	4366	8501	5467	8917
1000	5677	8648	6668	8927
1500	4494	8764	6286	8841
2000	4855	8601	6738	8903
2500	4686	8662	7074	9018
3000	6435	8623	6020	8961

running our code. It provides a single 12GB NVIDIA Tesla K80 GPU. We have used the nvcc compiler to compile the CUDA code.

The results in Table 1 show initial observation from executing the parallel code. The execution time is much higher than the execution time of the serial code for the same data. On further investigation it is found out that this is due to the time taken to initialize the GPU as the GPU switches off whenever it is not in use. Most of the algorithms that are developed in CUDA are tested after running a warmup call to the kernel code. This warmup call initializes the GPU and all the setup that is required. Interesting results were found after timing the second call to the kernel function for the Chess dataset.

Table 2. Execution time(milliseconds) for Connect dataset using sequential and parallel approach

# Transactions	Connect 6		Connect 7	
	Sequential	Parallel	Sequential	Parallel
10000	2218	8440	3211	8956
20000	2713	8578	3072	9148
30000	2392	8535	3431	9278
40000	2885	8702	3050	8988
50000	2738	8561	3718	8863
60000	2985	8543	3334	8958

The execution times for the second run are much faster than any of the results. It has a 400 times speedup on the initial run of the parallel code and approximately 250 times speedup against the serial code. To make sure that the results are accurate, we have flushed the Cuda memory using cudaFree() API and copied the data back to the device again to test the algorithm before the second run. The execution time includes the cudamemcpy API as well. To make sure the results are accurate we have noted down the average execution times of multiple runs.

Table 3. Execution times for parallel code in after initiating the GPU for Chess data

#Transactions	500	1000	1500	2000	2500	3000
Execution time (ms)	16	21	20	32	21	20

Both the Chess and Connect datasets are dense datasets, experiments are also conducted against sparse data that are generated using the SPMF generator from [16]. This data set contains 1000 transactions with 5000 unique items having a maximum transaction length of 7. A real-life example may be any rare disease with multiple mimicking symptoms such as sarcoidosis. The sparse data utilizes more memory than the dense data because the number of items in the sparse data is much higher than that of the dense data, hence it was running out of space and time on the CUDA kernel. A parallel approach was implemented to analyze the performance to discover rare itemsets from dense and sparse dataset. Theoretically, one can assume that the parallel algorithm is faster than the serial code because of the nature of the code. But the parallel code takes longer time to execute, this is because of the time taken to initialize the GPU which was not accounted for beforehand. The same can be observed in Table 3 which contains the execution time after initiating the GPU for Chess dataset.

5 Conclusion and Future Scope

This is one of the first algorithms in the field of rare itemset mining that has been developed to execute in parallel. The execution time increases as the length of the transaction increases, this is because it increases the depth of the recursion as we generate more sub header tables. The execution time also increases as we increase the number of transactions, but this observation has a lot of irregularities in it. This is because, as we store only the unique transactions, an increase in the size of the database might not always lead to growth in the size of our data structure as the new transactions might already exist in it. Also, it depends on the items falling in the range of minSup and maxSup. If the items that are rare have only a few transactions linked to it through the hyperlink, the depth of the recursion decreases, and the runtime decreases substantially because of this. One of the limitations of the parallel and serial codes that have been implemented is memory consumption. Since it is a recursive process and due to the large sizes of the datasets the main stack memory might be full if we increase the size of the transactions or the size of the database. The memory that can be stored in the CUDA functions is even lesser. We can run the code in the latest NVIDIA GPUs which have a higher compute capability and thus allow us to store more data. Further optimizations can be made in the algorithms too to reduce memory usage. This limitation can be observed while running the algorithms on sparse datasets which consume more memory than the dense datasets like chess and connect data.

In the parallel code, only one level of parallelism is achieved i.e., the load on the main thread is reduced by only one-fold. After the initial kernel call, all the processing happens on the same thread for each item of the HeaderTable HT. Further developments can be made to increase the parallelism of the algorithm by making the algorithm run in

parallel in the next level too. This can be done using dynamic parallelism in CUDA. A recursive method has been used in the kernel function to generate the itemsets of size k , where $k = 1, 2, 3, \dots, n$ where n is the size of the transaction. This algorithm is converted into an iterative algorithm by using a data structure to store all the necessary parameters for the next iteration since CUDA does allow recursion only up to one level. Further optimizations can be made in the algorithm by using tree-based approaches. Another optimization that can be done to the code is the usage of variable support values [13]. Only static minSup and maxSup values are used in the code. Using variable values for each itemset might lead to more accurate results. The memory that can be utilized on the CUDA device is little. Since the sparse data uses more memory than the dense datasets, it was not possible to execute the generated sparse dataset. This is a limitation of the parallel algorithm.

References

1. “Universal Turing Machine”. https://en.wikipedia.org/wiki/Universal_Turing_machine
2. Padhy, N., Mishra, P., Panigrahi, R. The survey of data mining applications and feature scope. ArXiv, abs/1211.5723 (2012)
3. Han, J., Cheng, H., Xin, D., et al.: Frequent pattern mining: current status and future directions. *Data Min. Knowl. Disc.* **15**, 55–86 (2007)
4. Borah, A., Nath, B.: Rare pattern mining: challenges and future perspectives. *Complex Intel. Syst.* **5**(1), 1–23 (2018). <https://doi.org/10.1007/s40747-018-0085-9>
5. Borgelt, C.: Frequent item set mining. *WIREs Data Mining Knowl. Discov.* **2**, 437–456 (2012). <https://doi.org/10.1002/widm.1074>
6. Szathmary, L., Napoli, A., Valtchev, P.: Towards rare itemset mining. In: 2007 19th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2007, vol. 1, pp. 305–312. IEEE (2007)
7. Liu, B., Hsu, W., Ma, Y.: Mining association rules with multiple minimum supports. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 337–341. ACM (1999)
8. Adda, M., Wu, L., Feng, Y.: Rare itemset mining. In: 2007 Sixth International Conference on Machine Learning and Applications, ICMLA 2007, pp. 73–80. IEEE (2007)
9. Cuzzocrea, A., Dayal, U. (eds.): LNCS, vol. 6862. Springer, Heidelberg (2011). <https://doi.org/10.1007/978-3-642-23544-3>
10. Bhatt, U., Patel, P.: A novel approach for finding rare items based on multiple minimum support framework. *Proc. Comput. Sci.* **57**, 1088–1095 (2015)
11. Lu, Y., Richter, F., Seidl, T.: Efficient infrequent pattern mining using negative itemset tree. In: Appice, A., Ceci, M., Loglisci, C., Manco, G., Masciari, E., Ras, Z.W. (eds.) *Complex Pattern Mining*. SCI, vol. 880, pp. 1–16. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-36617-9_1
12. Darrab, S., Broneske, D., Saake, G.: RPP algorithm: a method for discovering interesting rare itemsets. In: Tan, Y., Shi, Y., Tuba, M. (eds.) *Data Mining and Big Data: 5th International Conference, DMBD 2020, Belgrade, Serbia, July 14–20, 2020, Proceedings*, pp. 14–25. Springer Singapore, Singapore (2020). https://doi.org/10.1007/978-981-15-7205-0_2
13. Kanimozhi Selvi, C.S., Tamilarasi, A.: Mining rare itemset with automated support thresholds. *J. Comput. Sci.* **7**(3), 394–399 (2011). <https://doi.org/10.3844/jcssp.2011.394.399>
14. Cui, Y., Gan, W., Lin, H., Zheng, W.: FRI-miner: fuzzy rare itemset mining. *Appl. Intell.* **52**, 3387–3402 (2021). <https://doi.org/10.1007/s10489-021-02574-1>

15. Jian, L., Wang, C., Liu, Y., et al.: Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). *J. Supercomput.* **64**, 942–967 (2013)
16. Adil, S.H., Qamar, S.: Implementation of association rule mining using CUDA. *Int. Conf. Emerging Technol.* **2009**, 332–336 (2009). <https://doi.org/10.1109/ICET.2009.5353149>
17. “SPMF Datasets”. <http://www.philippe-fourmier-viger.com/spmf/index.php?link=datasets.php>