DATA SCIENCE JOURNAL

Gawwad, M A et al 2017 Frequent Itemset Mining for Big Data Using Greatest Common Divisor Technique. *Data Science Journal,* 16: 25, pp. 1–10, DOI: https://doi.org/10.5334/dsj-2017-025

RESEARCH PAPER

Frequent Itemset Mining for Big Data Using Greatest Common Divisor Technique

Mohamed A. Gawwad, Mona F. Ahmed and Magda B. Fayek

Cairo University, Faculty of Engineering, Computer Engineering Department, EG

Corresponding Author: Mohamed A. Gawwad, Ph.D. researcher (mohamedabgo@yahoo.com)

The discovery of frequent itemsets is one of the very important topics in data mining. Frequent itemset discovery techniques help in generating qualitative knowledge which gives business insight and helps the decision makers. In the Big Data era the need for a customizable algorithm to work with big data sets in a reasonable time becomes a necessity. In this paper we propose a new algorithm for frequent itemset discovery that could work in distributed manner with big datasets. Our approach is based on the original Buddy Prima algorithm and the Greatest Common Divisor (GCD) calculation between itemsets which exist in the transaction database. The proposed algorithm introduces a new method to parallelize the frequent itemset mining without the need to generate candidate itemsets and also it avoids any communication overhead between the participated nodes. It explores the parallelism abilities in the hardware in case of single node operation. The proposed approach could be implemented using map-reduce technique or Spark. It was successfully applied on different size transactions DBs and compared with two well-known algorithms: FP-Growth and Parallel Apriori with different support levels. The experiments showed that the proposed algorithm achieves major time improvement over both algorithms especially with datasets having huge number of items.

Keywords: data mining; frequent itemset mining; greatest common divisor; big data

1 Introduction

Frequent itemsets discovery "is one of the most important techniques in data mining" (Zhengui Li, 2012). It can find out the association relationships among events or data objects that are hidden in the data, even if the associated events or objects seems not related at all. Digging into these relationships will help to analyze events and "may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosis and many other applications" (Pramod S., 2015).

The Frequent itemset mining (FIM) methodologies are most popular and very well used by various researchers for finding frequent patterns among variables in large datasets (J. Manimaran1, 2013). Literature contains many approaches that tackle the FIM problem like Apriori, FP-Growth, multi-level frequent itemsets, DHP (Direct Hashing and Pruning), maximal association rule mining, primitive association rules, soft-matching rules and Buddy Prima.

There are many data mining techniques like association rules mining, correlation, sequential pattern analysis, classification and clustering that try to find interesting patterns in datasets. "The problem of mining frequent itemsets arose first as a sub-problem of mining association rules" (Yihua Zhong, 2010). The original motivation for finding frequent itemsets came from the need to analyze the "so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products" (Pramod S., 2015). FIM describes how often items are purchased together. For example, an association rules cheese, chips (80%) states that four out of five customers that bought cheese also bought chips. Such rules can be useful for decisions concerning products pricing, promotions, store layout and many others.

This paper is organized as follows. Section 2 formulates the problem statement of the FIM problem. Section 3 briefly introduces a literature overview and related work. Section 4 presents our proposed

approach. Section 5 discusses our experimental results. Finally, section 6 summarizes our conclusions and future work directions.

2 Problem Statement

"Studies of Frequent Itemset (or pattern) Mining is acknowledged in the data mining field because of its broad applications in Market basket analysis, Medical diagnosis, Protein sequences, Census data, CRM of credit card business" (Akash Rajak, 2012), graph pattern matching, sequential pattern analysis, and many other data mining tasks (Pramod S., 2015). Finding an efficient way for mining frequent itemsets are very important for mining association rules as well as for many other data mining tasks. One of the major challenges found in frequent itemset mining nowadays is the large amount of data and how to use the multicore feature in the new hardware as well as the distributed architecture with large datasets. As a result the need for new frequent itemset mining algorithms that could tackle the new trends has emerged. In this paper we propose a parallelizable algorithm for FIM that could deal with big data sets exploiting the multicore feature of the hardware.

3 Literature Review

In this section a brief overview about the well-known FIM algorithms is presented. It includes: the AIS algorithm, Apriori, FP-Growth and algorithms that depend on prime numbers representation.

"The AIS algorithm" (Qiankun Zhao, 2003) "was the first algorithm proposed by Agrawal, Imielinski, and Swami for mining association rule" (Kumbhare et al, 2014). AIS algorithm depends on scanning the databases many times to get the frequent itemsets (Kumbhare et al, 2014). "The support count of each individual item was accumulated during the first pass over the database. Based on the threshold of support count those items whose count is less than its minimum value are eliminated from the list of items. Candidate 2-itemsets are generated by extending frequent 1-itemsets with other items in the transactions" (Kumbhare et al, 2014). "During the second pass over the database, the support count of those candidate 2-itemsets are accumulated and checked against the support threshold" (Kumbhare et al, 2014). "Similarly those candidate (k + 1)-itemsets are generated by extending frequent k-itemsets with items in the same transaction. The candidate itemsets generation and frequent itemsets generation process iterates until any one of them becomes empty" (Kumbhare et al, 2014). "AIS Algorithm has efficiency problems so some modifications have been introduced to give an estimation for candidate itemsets that have no hope to be large, consequently the unnecessary effort of counting those itemsets can be avoided" (Kumbhare et al, 2014). "Also since all the candidate itemsets and frequent itemsets are assumed to be stored in the main memory, memory management is also proposed for AIS when memory is not enough" (Kumbhare et al, 2014).

Original Apriori Algorithm is one of the well-known algorithms for mining frequent itemsets. It was introduced in (R. Agrawal, 1993). "The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemsets properties" (G.S Almasi, 1989). "Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k + 1)-itemsets. First, the frequent 1-itemset is found, this is denoted by L_1, which is used to find the frequent 2-itemset L_2 and so on" (Pramod S., 2015). "To improve the efficiency of the level-wise generation of frequent itemsets, a property called Apriori property is used to reduce the search space" (R. Agrawal, 1993). "This property states that all nonempty subsets of a frequent itemset must also be frequent" (Pramod S., 2015). A two-step process is used to find L_k from L_{k-1} 1)

The join step: To find L_k, a set of k-itemsets is generated by joining L_{k-1} with itself (G.S Almasi, 1989). Suppose L_{k-1} contains items (A, B, C) then L_k set will contain (AB, AC, BC). This set of candidate itemsets is denoted C_k. 2)

The prune step: C_k is a superset of L_k that contains all frequent and non-frequent k-itemsets.

A scan of the database is done to determine the frequency of each candidate which exists in Ck, those who satisfy the minimum support is added to L_k for the next iteration and the other items are pruned.

AprioriTID algorithm (Manisha Girotra et al, 2013) tries to improve the performance of Apriori by avoiding multiple DB hits in the reading process. "AprioriTID doesn't use the database for counting the support of candidate itemsets after the first pass" (Kumbhare et al, 2014). "The process of candidate itemset generation is the same like the original Apriori algorithm. Another set C′ is generated of which each member has the Transaction ID (TID) of each transaction and the frequent itemsets (present in this transaction where C_k' are in the form <TID, $\{X_k\}$> and each X_k is a potentially frequent k-itemset present in the transaction with identifier TID" (Kumbhare et al, 2014). "This set is used to count the support of each candidate itemset" (Kumbhare et al, 2014). Suppose that the original transaction database contains the following transactions associated with its transactions IDs

<TID, items > {(100, 1 3 4), (200, 2 3 5), (300, 1 2 3 5), (400, 2 5)}.
Then C_1 will be in form of ($X_{k=1}$, Support Count).
C_1 = {({1}, 2), ({2}, 3),({3}, 3), ({5}, 3)}.
C_1 frequent itemsets are used to generate itemsets for C_2 and the support count for each itemset generated in C_2 is calculated using another subset called C_2'.
C_2 (itemsets) = {{1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}}.
C_2' is derived from the original database taken into consideration only items equal or more than support threshold.
C_2' = {(100, {1 3}), (200, {2 3}{2 5}{3 5}), (300, {1 2}{1 3}{1 5} {2 3} {2 5} {3 5}), (400, {2 5}),.....}.
To get the frequency of C_2 elements the search is done in C_2'.
C_2 = {({1,2}, 1), ({1,3}, 2), ({1,5}, 1), ({2,3}, 2), ({2,5}, 3), ({3,5}, 2)}.
This method may lead to less number of scans.

"**FP-Growth Algorithm** is the most popular frequent itemset mining algorithm that was introduced in" (J. Han, 2000). "The main aim of this algorithm was to remove the bottlenecks of the Apriori algorithm in generating and testing candidate sets" (Pramod S., 2015). "The problem of the Apriori algorithm was dealt with by introducing a novel, compact data structure called frequent pattern tree or FP-tree" (Pramod S., 2015). "Then based on this structure an FP-tree based pattern fragment growth method was developed" (Pramod S., 2015). "FP-Growth uses a combination of the vertical and horizontal database layout to store the database in main memory" (Anjana Gosain, 2013). "Instead of storing the ID for every transaction in the database, it stores the actual transactions from the database in a tree structure and every item has a linked list going through all transactions that contain that item" (Anjana Gosain, 2013). "This new data structure is denoted by FP-tree (Frequent Pattern tree)" (J. Han, 2003).

CARMA Algorithm (C. Hidber et al, 1999) Continuous Association Rule Mining Algorithm. The algorithm mainly aims to process large online datasets (C. Hidber et al, 1999). CARMA algorithm comprises two phases during its operation: in the first phase CARMA builds "lattice of all potential large itemsets with respect to the scanned part of data" (C. Hidber et al, 1999). "For each set on the lattice CARMA determines the lower and upper limits of its support" (V. Pudi, 2001). The deduced association rules are displayed to the user after processing transactions and the user is free to adjust the support count (C. Hidber et al, 1999). In the second phase after getting the user's feedback CARMA fully scans all the dataset to determine exactly the occurrences of each itemset and removes all the itemsets below the user's specified threshold (V. Pudi, 2001).

3.1 Prime Numbers Representation Algorithms

In this section brief overview about prime numbers representation algorithms will be introduced:

Parallel Buddy Prima (S.N. Sivanandam, 2004) depends on the concept of prime numbers which are used for prima representation operation. "By using prime numbers to represent items in a transaction where each item is assigned a unique prime number, each transaction is represented by the product of the corresponding prime numbers of individual items in the transaction" (S.N. Sivanandam, 2004). Since the product of different prime numbers is unique it is easy to check for the presence of a specific candidate in a transaction by modulo division of a transaction's prime product by the prime product of an itemset. "In addition, this representation stores only one number for each transaction so it uses less memory. Parallel Buddy Prima algorithm implements the original Buddy Prima in a master-slave architecture using candidate distribution technique" (S.N. Sivanandam, 2004). "The technique assigns disjoint sets of transactions from the transactions database to different processors" (S.N. Sivanandam, 2004). "The Master node removes infrequent itemsets and stores the Prime multiplication representation for each transaction in shared memory" (S.N. Sivanandam, 2004). "Master node also finds the Maximal length transaction size and puts it in shared memory then it divides the transactions equally among nodes for candidate itemsets generation" (S.N. Sivanandam, 2004). If there are k slaves and n transactions, then n/k transactions are assigned to each slave. Master connects to each slave node and initiates the process of finding the frequent itemsets. The slave nodes generate the candidate itemsets with different size starting with size 2 until the maximal length specified by the master node. For each candidate itemset generated, it finds its support count by accumulating the count of the itemset existence through the whole dataset assigned to it. Each slave node sends a list of the local frequent itemsets

with respect to the support count threshold to the master node. "Finally, the master node shows the global frequent itemsets after gathering the local frequent itemsets" (S.N. Sivanandam, 2004).

Cluster Based Partition Approach "uses prime numbers to represent the itemset in the transaction. This algorithm integrates both the bottom-up search as well as the top-down search" (Akhilesh Tiwari, 2009) and is suitable for itemsets of different sizes (Akhilesh Tiwari, 2009). "The Algorithm uses the top-down approach to find the frequent subsets of itemsets" (Akhilesh Tiwari, 2009). "The bottom up approach is used to find the supersets of the frequent itemsets" (Akhilesh Tiwari, 2009). "Most algorithms for mining frequent itemsets are based on bottom-up search approach" (Akhilesh Tiwari, 2009). "In this approach, the search starts from itemsets of size 1 and extends one level in each pass until all maximal frequent itemsets are found" (Akhilesh Tiwari, 2009). "This approach performs well if the length of the maximal itemset is short" (Akhilesh Tiwari, 2009). "If the maximal itemset is longer, top-down search is suitable" (Akhilesh Tiwari, 2009). "For a transaction with a medium sized maximal frequent set, a combination of both these approaches performs well" (Akhilesh Tiwari, 2009). "This algorithm adopts Candidate distribution method to distribute the candidates among all nodes" (Akhilesh Tiwari, 2009). "The support count of the supersets and the subsets are found effectively from the prime number representation method" (Akhilesh Tiwari, 2009).

4 The Proposed Algorithm (POBPA)

In this paper, we applied a new approach to parallelize Buddy Prima algorithm and optimize it to work more efficiently with Big Data and exploit the hardware parallelism capabilities. The new algorithm, called **"Parallelizable Optimized Buddy Prima Algorithm" POBPA** for short depends on greatest common divisor calculation between different transactions in the transaction database. Generally speaking, POBPA consists of two main steps:

1. Data preparation:
   Items and accordingly transactions are represented in the same manner as done by the Buddy Prima algorithm illustrated in the previous section. POBPA analyzes the transaction dataset and extracts single items with frequency equal to or higher than the support count specified by the user. Also it prepares ignore list for infrequent items and their combinatorials. After removing these infrequent items from the original dataset it calculates prime multiplications and string representation.
2. Frequent Itemsets Deduction (Greatest Common Divisor calculation (GCD)):
   By calculating the GCD between two transactions we can get the common items between these two transactions and by counting the highest repeated GCDs we get the frequent itemsets in the transaction dataset.

POBPA is not an architecture dependent algorithm it could be applied using different architectures like distributed environments, multiple core processors or GPUs. In this study POBPA exploits the multithreading feature which exists in JAVA to take the advantage of multithreading and multicore processors. POBPA works as multiple data single instruction (MDSI) with all nodes participating equally in the mining process to avoid any communication overhead between different nodes. POBA creates lightweight processes as the overhead of switching between threads is less. The Java Virtual Machine (JVM) spawns threads when the algorithm is run. Each thread works with part of the data and JVM takes the responsibility of orchestrating between them and consolidating the results.

4.1 Data Preparation

POBPA uses horizontal representation for transaction datasets (Gupta, 2011) where each transaction is represented by a row in the database which has a transaction identifier (TID) followed by a set of items as shown in **Table 1**.

The first step during the data preparation phase is counting the frequency of each individual item in the dataset. Once items frequencies are determined, the algorithm will prepare the ignore list which includes a set of items not to be considered in our further analysis. Ignoring items depends on the support count specified by the algorithm user. Support count is the minimum frequency for items according to user mining requirements (Anjana Gosain, 2013). The ignore list contains all the items with frequency less than the support count and also all these items' combinatorials. **Table 2** illustrates the ignore list for support count 3.

Table 1: Dataset Horizontal Representation.

TID	ITEMS
T1	I1, I2, I3, I4, I5, I6
T2	I1, I2, I4, I7
T3	I1, I2, I4, I5, I6
T4	I1, I2
T5	I3
T6	I2
T7	I7

Table 2: Ignore List for Support Count 3.

Ignore List
I3
I5
I6
I7
I3 I5
I3 I6
I3 I7
I5 I6
I5 I7
I6 I7
I3 I5 I6
I3 I5 I7
I3 I6 I7
I5 I6 I7

Our algorithm assigns a prime number to each item in the dataset except the ones in the ignore list. Prime number assigning is done with respect to item's frequency. The highest frequency item will be represented by the minimum prime number which is 2 and so on. The inverse proportional relationship between the item frequency and the assigned prime number will result in less value for prime multiplication in the subsequent steps and hence less storage and less computational complexity.

After this, each transaction matching one of the combinatorials existing in the ignore list will be removed from the transaction database. The remaining transactions will be represented in two representations, string ordered representation with all prime numbers replacing items in ascending order in one string and prime multiplication representation. **Table 3** shows these representations. Repetition of item I is denoted as R(I). The prime number assigned to item I is denoted as P (I).

The next step is partitioning the transaction dataset depending on the string representation, making a subset that includes transactions starting with 2 then a subset which starts with 3 and so on. This partitioning provides the ability to parallelize the subsequent operations and also helps to reduce the computations. The following example illustrates the partitioning process. Suppose that after the row transactions were processed as per the previous steps the string representation was as illustrated in **Table 4**.

As illustrated in **Table 4**, the transactions are partitioned into groups depending on their string representation. This technique gives the ability to use a modern distributed processing model like distributed file system in Hadoop or even utilize the parallel processing abilities in GPUs.

Table 3: Transactions Representations.

Transaction ID	Items	Items Repetition and Prime Assignment	Repetition	String ordered representation	Prime Multiplication
T1	I1, I2, ~~I3~~, I4, ~~I5, I6~~	R(I1) = 4, R(I2) = 5 and R(I4) = 3 Then P(I1) = 3, P(I2) = 2 and P(I4) = 5	3	2 3 5	30
T2	I1, I2, I4, ~~I7~~				
T3	I1, I2, I4, ~~I5, I6~~				
T4	I1, I2	P(I1) = 3 P(I2) = 2	1	2 3	6
T5	~~I3~~				
T6	I2	P(I2) = 2	1	2	2
T7	~~I7~~				

Table 4: Partitioning Process.

Transaction ID	String Representation	Partition Number
T1	2 3 7	**Par 1** for all transactions with string representation starting by "2"
T2	3 7 11	**Par 2** for all transactions with string representation starting by "3"
T3	11 19 23	**Par 3** for all Transaction with string representation starting by "11"
T4	2 23 29	**Par 1** for all transactions with string representation starting by "2"
T5	3 31 37	**Par 2** for all transactions with string representation starting by "3"
T6	11 53 59	**Par 3** for all transactions with string representation starting by "11"
T7	29 31 37	**Par 4** for all transactions with string representation starting by "29"

4.2 Frequent Itemsets Deduction using GCD

The proposed method depends on calculating GCD between the prime multiplications of each subset resulting from the partitioning and calculating cross GCD among subsets. Before starting the computation of itemset frequencies, a pool of subsets is created- as illustrated in the previous section- to give the distributed nodes, ALUs or processor cores the ability to handle subsets in parallel.

For each subset we start calculating GCD using Euclidean method. The Euclidean algorithm, is an efficient method for computing the greatest common divisor (GCD) of two numbers. It is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by the difference between it and the smaller number. For example, 21 is the GCD of 252 and 105 ($252 = 21 \times 12$ and $105 = 21 \times 5$), and the same number 21 is also the GCD of 105 and 147 ($147 = 252 - 105$). "Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until one of the two numbers reaches zero". "When that occurs, the other number (the one that is not zero) is the GCD of the original two numbers" (wikipedia, 2016).

The transactions in each subset are ordered lexicographically. The Algorithm calculates GCD between each transaction in the subset and the other transactions till we reach the exit criteria. The exit criteria depends on the string representation, since the string contains the transaction's items ordered ascendingly by the prime numbers values then if during calculation of GCD of a certain transaction with other ones the last item in the string representation of this transaction is less than the second item of the other transaction's representation, that means no GCD can be found between these two transactions and also between this transaction and all the subsequent transactions in the subset. Each GCD value with its frequency will be placed in a hash table structure. The frequency is incremented whenever this same GCD value is encountered again in a subsequent transactions.

As an example suppose we have the following subset which starts by 2: ((2,3,5) [30], (2,3,5,7) [210], (2,3,7) [42], (2,3,11) [66], (2,13) [26], (2,13,17) [442], (2,13,17,19) [8398]). The numbers show each transaction represented as string ordered representation followed by its representation as prime numbers multiplication. Note that any two transactions in the same subset have by definition a common item but during GCD frequency

calculation only two or more items in common (two or more prime numbers) are considered as the frequency of single items are already calculated from the preparation phase as illustrated in section 4.1. Calculations will start with transaction (2, 3, 5) computing the GCD between it and the rest of the transactions in the same subset first. To calculate GCD we use the prime representations of the two transactions – shown between curved brackets – and the resulted GCDs with their frequencies are stored in a hash table. For transaction (2, 3, 5) the exit criteria will be met when we reach transaction (2, 13) since the second item in transaction (2, 13) has greater value than the last number in transaction (2, 3, 5) then no GCD can be found between them except 2 and the lexicographic order means that no GCD can be found between (2, 3, 5) and any transaction after (2,13). After calculating the GCDs between all transactions in the subset we need to check GCDs between transactions across different subsets. As an example for transaction (2, 3, 5) by removing 2 we get a new transaction (3, 5). If (3, 5) already exists in the subset of 3 then we increase its frequency by one. Repeat this with (5, 7) also for the subset starting with 5.The result of this phase is one table containing all GCDs and their frequencies.

5 Experimental Results

We experimented with datasets from different applications. These datasets were obtained from the UCI repository of machine learning databases (C.L. Blake, 1998). **Table 5** below shows the characteristics of the datasets selected for the experiment. Our algorithm has the capability to work with datasets with a huge number of items (more than 15000 items per transaction). We used Retail dataset to show this capability for the proposed algorithm **POBPA**.

We compared the execution time of the proposed algorithm with FP-Growth and Parallel Apriori. The performance metric in the experiments is the total execution time taken and it was shown that POBA made a tremendous improvement especially with datasets that have huge number of items. We applied our experiments on the previously illustrated datasets using 30% to 70% minimum support threshold.

Table 6 compares the FP-Growth algorithm, Parallel Apriori algorithm and POBPA with respect to processing time using the Retail dataset. The Algorithms ran over the same machine (4 cores, 8 GB RAM).

Table 7, 8 and **9** make the same comparison using the Mushroom, Letter Recognition and Adult datasets respectively.

In **Table 6** we could observe the interesting behavior of POBPA with the Retail dataset which includes a huge number of items. Our algorithm by far outperformed the FP-Growth and the parallel Apriori algorithms. The processing time ranges of both algorithms vary greatly as our distribution feature which fits multiprocessors results in great time enhancement which is more evident with increasing the dataset size. **Tables 7, 8** and **9** also support the same observation.

Table 5: Datasets Used For the experiment.

Dataset Name	Number of Transactions	Number of Items
Mushroom	8124	90
letRecog	20000	106
Adult	48842	97
Retail	88146	16469

Table 6: Test Results with Retail Dataset.

	Time Consumed in Seconds		
Support Count	FP-Growth	Parallel Apriori	POBPA
30%	28	37	<0.05
40%	26	34	<0.05
50%	22.1	32.9	<0.05
60%	22.04	32.3	<0.05
70%	21.19	27	<0.05

Table 7: Test Results with Mushroom Dataset.

	Time Consumed in Seconds		
Support Count	**FP-Growth**	**Parallel Apriori**	**POBPA**
30%	59	13	0.88
40%	48	3.9	0.64
50%	32	1.9	0.15
60%	30	1.5	0.07
70%	28	1.4	<0.05

Table 8: Test Results with Letter Recognition.

	Time Consumed in Seconds		
Support Count	**FP-Growth**	**Parallel Apriori**	**POBPA**
30%	273	38	24
40%	200	23	14
50%	179	18	3
60%	150	18	1.2
70%	110	15	0.05

Table 9: Test Results with Adult dataset.

	Time Consumed in Seconds		
Support Count	**FP-Growth**	**Parallel Apriori**	**POBPA**
30%	400	24	1.8
40%	328	10	1.5
50%	300	8	0.82
60%	213	6	0.75
70%	200	4	0.5

6 Conclusion and future work

In this paper we have introduced a new technique for FIM. Our approach, POBPA, proved to excel other approaches found in the literature. POBPA is not an architecture dependent algorithm it could be applied using different architectures like distributed environments and multiple core processors. It aims to parallelize the Buddy Prima algorithm and also to optimize it to work more efficiently with Big Data exploiting the hardware parallelism capabilities. It depends on GCD calculation between different transactions in the transaction database. It assigns prime numbers in inverse proportional relationship to the item frequency which guarantees less value for prime multiplication and hence less storage and less computational complexity. POBPA achieves excellent results when comparing its execution time with FP-Growth and Parallel Apriori especially for dataset with large number of items.

Future work directions include applying POBPA to a distributed architecture with datasets containing millions of records. It could be applied using Hadoop Distributed File System (HDFS) and Map Reduce technique. Hadoop uses distributed file system method, which basically implements a mapping system to locate data in a cluster then MapReduce programming is used in the same servers which allows faster processing of data. It can deal with large volumes of unstructured data. Hadoop MapReduce takes minutes to process terabytes of data. POBPA could be applied on stream analytics to identify the response of social media to a certain topic.

Competing Interests

The authors have no competing interests to declare.

References

Agrawal, R, Imielinksi, T and **Swami, A** 1993 "Mining association rules between sets of items in large database" *The ACM SIGMOD Conference*. DOI: https://doi.org/10.1145/170035.170072

Almasi, G S and **Gottlieb, A** 1989 "Highly Parallel Computing" The Benjamins Publishing Company Redwood City, CA.

Blake, C L and **Merz, C J** 1998 "UCI Repository of Machine Learning Databases" Dept. of Information and Computer Science, University of California at Irvine, CA, USA.

Girotra, M, et al 2013 "Comparative Survey on Association Rule Mining Algorithms" *International Journal of Computer Applications*. DOI: https://doi.org/10.5120/14612-2862

Gosain, A and **Bhugra, M** 2013 "A Comprehensive Survey of Association Rules on Quantitative Data In Data Mining" IEEE Conference on Information and Communication Technologies (ICT). DOI: https://doi.org/10.1109/cict.2013.6558244

Gupta, D G 2011 "A Taxonomy of Classical Frequent Item set Mining Algorithms" *International Journal of Computer and Electrical Engineering*, 3rd ed.

Han, J, Pei, H and **Yin, Y** 2000 "Mining Frequent Patterns without Candidate Generation" Conf. on the Management of Data (SIGMOD'00, Dallas, TX), ACM Press, New York, NY, USA. DOI: https://doi.org/10.1145/342009.335372

Han, J, Pei, J, Yin, Y and **Mao, R** 2003 "Mining frequent patterns without candidate generation: A frequent pattern tree approach" Data Mining and Knowledge Discovery.

Hidber, C, et al 1999 "Online association rule mining" In Proc. of ACM SIGMOD Intl. Conf. on Management of Data.

Kumbhare, et al 2014 "An Overview of Association Rule Mining Algorithms" *International Journal of Computer Science and Information Technologies (IJCSIT)*, Vol (5).

Li, Z and **Zhang, R** 2012 "The Association Rule Mining on a Survey Data for Culture Industry International" Conference on Systems and Informatics (ICSAI).

Manimaran1, J and **Velmurugan, T** 2013 "A Survey of Association Rule Mining in Text applications" IEEE International Conference on Computational Intelligence and Computing Research.

Pramod, S and **Vyas, O P** 2015 "Survey on Frequent Item set Mining Algorithms" *International Journal of Computer Applications*, Volume (1).

Pudi, V and **Haritsa, J** 2001 "On the optimality of association-rule mining algorithms" Technical Report TR-2001-01, DSL, Indian Institute of Science.

Rajak, A and **Gupta, M K** 2012 "Association Rule Mining: Applications in Various Areas" *International Conference on Data Management*.

Sivanandam, S N, Sumathi, S, Hamsapriya, T and **Babu, K** 2004 "A Hybrid Parallel Frequent Item set mining algorithm for very large databases" *Academic open internet journal*.

Tiwari, A, Gupta, R K and **Agrawal, D P** 2009 "Cluster Based Partition Approach for Mining Frequent Itemsets" *International Journal of Computer Science and Network Security IJCSNS*, 9th ed.

Zhao, Q and **Bhowmick, S S** 2003 "Association Rule Mining: A Survey Technical Report" CAIS, Nanyang Technological University, Singapore.

Zhong, Y and **Liao, Y** 2010 "Research of Mining effective and Weighted Association Rules Based on Dual Confidence" CPS Fourth International Conference on Computational and Information Science.

How to cite this article: Gawwad, M A, Ahmed, M F and Fayek, M B 2017 Frequent Itemset Mining for Big Data Using Greatest Common Divisor Technique. *Data Science Journal,* 16: 25, pp. 1–10, DOI: https://doi.org/10.5334/dsj-2017-025

Submitted: 21 November 2016 **Accepted:** 26 April 2017 **Published:** 18 May 2017

]u[*Data Science Journal* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS