Computer Science • 23(2) 2022 https://doi.org/10.7494/csci.2022.23.2.3776

Noorollah Karimtabar
Mohammad Javad Shayegan Fard

FINDING FREQUENT ITEMS: NOVEL METHOD FOR IMPROVING APRIORI ALGORITHM

Abstract *In this paper, we use an intelligent method for improving the Apriori algorithm in order to extract frequent itemsets. PAA (the proposed Apriori algorithm) pursues two goals: first, it is not necessary to take only one data item at each step – in fact, all possible combinations of items can be generated at each step; and second, we can scan only some transactions instead of scanning all of the transactions to obtain a frequent itemset. For performance evaluation, we conducted three experiments with the traditional Apriori, BitTableFI, TDM-MFI, and MDC-Apriori algorithms. The results exhibited that the algorithm execution time was significantly reduced due to the significant reduction in the number of transaction scans to obtain the itemset. As in the first experiment, the time that was spent to generate frequent items underwent a reduction of 52% as compared to the algorithm in the first experiment. In the second experiment, the amount of time that was spent was equal to 65%, while in the third experiment, it was equal to 46%.*

Keywords Apriori algorithm, frequent itemset, intelligent method

Citation Computer Science 23(2) 2022: 161–177

1. Introduction

A large amount of data has been collected in various areas such as health, sports, banking, business, environmental protection, security, politics, and many others. Such data is often used for analysis and discovering useful information. Typically, datasets grow exponentially when they take on a large number of complex features. Generally, these datasets have a relatively low information density in the context of association rule mining. Robust, simple, and computationally efficient tools are required to extract information from such datasets. Data mining is defined as the extraction of knowledge from a very large-sized database. Data mining is also called knowledge discovery in databases (KDD) [9].

Association rule mining is the most convenient method or association knowledge discovery. The associations between data values in a transaction database are complicated, and most of them are implicit. Association rule mining (a component of data mining) is a research field that has been proposed proposed earlier [14].
For example, individuals who are buying diapers are likely to buy baby powder as well. Association rule mining is used to find relationships among certain items [9].
A typical association rule that can result from such a study might be "90 percent of all customers who buy diapers also buy powder." The aim of association rule mining is to make associations among sets of items in transaction databases or other data repositories [1]. Currently, Apriori algorithms play a significant role in identifying frequent itemsets and deriving rule sets out of them, which enumerate all of the frequent itemsets. Apriori algorithms require minimum support and a minimum number of thresholds to find frequent patterns from a transactional database.

1.1. Classic Apriori algorithm

The classic Apriori algorithm (CAA) includes two key processes: a connecting step, and a pruning step [2].

The connecting step is as follows: to get L_k (frequent k-itemset), connect L_{k-1} with itself. Set this candidate as C_k, and assume L_1 and L_2 are the item sets of L_{k-1}. $L_{i[j]}$ is the j^th item of L_i. Assume that the affairs and items of the itemset are in alphabetical order. Execute the $L_{k-1} \infty L_{k-1}$ connection in which the elements of L_{k-1}, L_1, and L_1 are connectable if they have the same first $(k-2)^{th}$ items; that is, the elements of L_{k-1}, L_1, and L_1 are connectable if $(L_{1[1]}{=}L_{2[1]}) \wedge (L_{1_{[}2]} = L_{2[2]}) \wedge (L_{1[k-2]} = L_{2[k-2]}) \wedge (L_{1[k-1]} = L_{2[k-1]})$. The requirement of $(L_{1[k-1]} < L_{2[k-1]})$ simply assures no repetition. After connecting L_1 and L_2, the resulting itemset is $L_{1[1]}\ L_{1[2]}\ \ L_{1[k-1]}\ L_{2[k-1]}$.

The pruning step is as follows: C_k encompasses k-itemsets (which are either frequent or not), but L_k is a subset of C_k and includes those members that are all k-frequent-itemsets. Scan the database and clear the counters of each candidate itemset of C_k to assure L_k. However, C_k might be very large and, thus, consume a huge amount of the calculation. To decrease C_k, the following method would be

considered using the Apriori proprieties; any infrequent itemsets with k_1 items are not a subset of the frequent itemsets with k items. Consequently, if the (k-1) subset of a candidate itemset with k items is not in L_{k-1}, the candidate itemset is not frequent and can be deleted in C_k [2].

The CAA pseudo-code for discovering frequent itemsets for mining is given below (Algorithm 1) [9]:

Algorithm 1 CAA

```
1: k = 1
2: Generate frequent itemsets of length 1
3: Repeat until no new frequent itemsets are identified
4: Generate length (k+1) candidate
5:     Itemsets from length-k frequent itemsets
6:     Prune candidate itemsets that contain length-k subsets that are infrequent
7:     Count support of each candidate by scanning DB
8:     Eliminate candidates that are infrequent, leaving only those that are frequent
```

The time complexity of the CAA algorithm is equal to $O(2^d)$, where d stands for the total number of unique items in the transaction dataset [9]. Suppose that there is a test dataset (TD) that contains six transactions (as is shown in Table 1).

Table 1
Transaction database TD

TID	Itemset(A,B,C,D,E)
1	A,C, D,E
2	B, C, E
3	A, B, C, E
4	B, E
5	A, B, C, E
6	D

*TID = Transaction Identification

Here, we set the minimum support as 0.6 that supports 3. It is possible to find a frequent itemset for TD using CAA. This process is demonstrated in Figure 1. CAA achieves good performance when a dataset is sparse and the itemsets have short lengths. Nonetheless, this method suffers from the nontrivial costs that are caused by generating huge numbers of candidate itemsets and extra scans over the datasets for the support computation [15]. For instance, if there are 104 frequent 1-itemsets, it may need to generate more than 107 candidates into 2-lengths, which in turn will be tested and accumulated. Furthermore, it will have to generate 21,000 candidate itemsets to detect a frequent pattern of a size of 1000 (e.g., V1, V2...V1000); this yields a costly wasting of time in candidate generation as it checks for many more

sets from the candidate itemsets – moreover, it scans the database many times when searching for candidate itemsets [5].

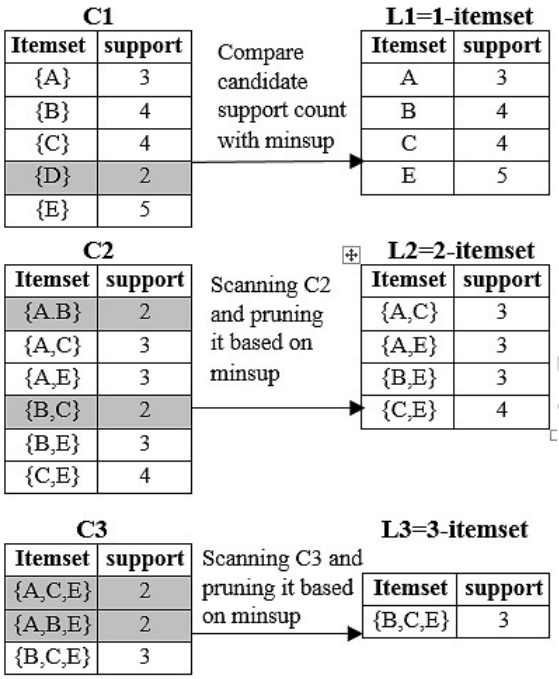

Figure 1. Operation process of CAA

1.2. Motivation

The Apriori algorithm is very slow and inefficient when a large number of transactions limits a computer's memory capacity, as it must scan a database many times to find candidate itemsets. In this paper, an algorithm is proposed to improve CAA by using an intelligent method along with a matrix data structure. In the proposed Apriori algorithm (PAA), creating a new matrix prevents the creation of unnecessary itemsets and reduces the number of transaction scans to create frequent itemsets.

2. Related Works

L.n. Sun [13] proposed an improved Apriori algorithm that builds a 01 transaction matrix by scanning a transaction database for obtaining its weighted support confidence. The items and transactions are weighted to reflect the importance of the transaction database.

In [6], the problem of designing a different private frequent itemset-mining algorithm that can simultaneously provide high levels of data utility and data privacy was

studied.truncating large transactions may causes decline the performance and data loss. The solution is that the long transactions are split but not truncated. The small-weighted technique is proposed to divide long transactions into sub-transactions.

N. C. Benhamouda et al. [4] developed a new recursive algorithm based on Apriori that aims to substantially reduce runtimes in order to increase its efficiency while preserving its effectiveness. This algorithm is based on the "divide and conquer" technique, which consists of partitioning a whole database into small ones and then applying Meta-Apriori if the database is huge or Apriori if it is of a reasonable size. By merging the achieved results, the outcomes for the whole database will be obtained.

In [3], a dynamic programming approach is utilized to facilitate fast candidate-itemset generation, reduce the number of database scans, and eliminate duplicate candidate itemsets.

H. V. Duong et al. [8] proposed an equivalence relationship by using the closure of an itemset to partition a solution set into disjoint equivalence classes as well as a new efficient representation of the rules in each class based on a lattice of the closed itemsets and their generators. They developed a new algorithm (called MAR-MINSC) to rapidly mine all of the constrained rules from the lattice instead of mining them directly from the database.

The properties and application requirements that are associated with the intelligence data in cyberspace were scrutinized by Zhang Jie et al. [10]. In this study, a new improved algorithm is recommended whose functioning is based on the Apriori algorithm. Frequent itemsets and non-frequent itemsets are indeed extracted here through setting double thresholds. Meanwhile, there was a reduction in the quantity of non-frequent itemsets; afterward, confidence, threshold judgment, and non-frequent itemsets were employed. Moreover, they conducted the mining of positive and negative association rules. Correspondingly, they recognized the integration of a massive amount of information data in cyberspace. By inducing and filtering the integrated information data, they excavated the association rules while attaining the useful information. Lastly, they identified the impacts of "assistant decision-making."

What follows is a brief overview of the three algorithms that were used in comparison with PAA. In the first algorithm (which is called BitTableFI), a particular data structure (named BitTable, which contains a set of integers whose bits represent an item) was used horizontally and vertically to compress a database for quick candidate-itemset generation and support count. In order to compress the candidate itemsets and the database, BitTable was utilized in BitTableFI for candidate-itemset compression. As such, if candidate itemset C contains item i, bit i of the BitTable element will be marked as one; if not, it will be marked as zero. It is of note that BitTable will be utilized vertically for database compression [7].

The second algorithm to be compared with PAA is TDM-MFI. A directed graph is utilized in this algorithm in order to store the information that is associated with the frequent itemsets of transaction databases. This approach provides the trifurcate-linked list storage structure of the directed itemsets graph. With the directed itemsets

graph, the mining process of the maximal frequent itemsets ultimately converts into the process of traversing the directed itemsets graph. Primarily, the first node will be chosen as the starting node of the whole traversing process. This is where the adjacent node is visible. Subsequently, it is possible to see its adjacent node when starting from the mentioned node. Such a visiting procedure ensues in anticipation of the latest adjacent node or a supporting number that will be smaller than the minimum support degree that is being attained. Along these lines, a maximal frequent itemset is mined and stored into the set of the maximal frequent itemsets.

Finally, the path recurrently returns to the upper layer for visiting the other nodes. If the subsequent extracted frequent itemsets are a subset of the previous maximal frequent itemsets, they are not stored. This feature prevents the algorithm from repeating the process. By redoing such an operation, the whole traversing process will not be ended until each of the nodes of the directed itemset graph is chosen as a starting node [11].

The third algorithm to be compared with PAA is MDC-Apriori. This algorithm builds a 0-1 transaction matrix through scanning the database one time while establishing the AE arrays in order to weigh each column in the matrix and calculate the completely-weighted support for the frequent itemsets. Both the minimum support and the strategy of the double correlation are employed to prune the frequent itemsets as well as the negative itemsets. As a final point, the profit constraint extensional association rules will be mined and the strategy degree of the correlation specified using the correlation coefficient [12].

Despite the variety of works to improve the Apriori algorithm, one of the major concerns is the slowness of the algorithm due to repeated scans of the entire dataset. In the algorithm that is proposed in this paper (PAA), an intelligent method is used to prevent an entire dataset search.

3. Research Approach

A new intelligent method is presented in this paper that is based on the dynamic programming strategy of minimizing the number of searches in a dataset. The proposed algorithm is compared with the CAA algorithm as well as three others; namely, the BitTableFI, TDM-MFI, and MDC-Apriori algorithms. These algorithms were briefly described in the previous section. The three algorithms encompass data that varies in terms of a record's numbers and variables while providing us with the appropriate conditions for running the experiment and establishing the required comparisons. These experiments were performed on the Mushroom, Congressional Voting, and Car Evaluation datasets; all of these datasets were attained from the UCI Machine-Learning Repository. The attribute (item) types of these datasets were categorical. This paper considers a different value of each expanded data instance as a transaction. For example, the Car Evaluation dataset consists of 6 items, and each item has 3 or 4 values; however, the formed dataset contains 21 columns. The details of each of these datasets are depicted in Table 2.

Table 2
Dataset characteristics

Dataset Name	Number of items	Number of records
Mushroom	120	8416
Congressional Voting	17	435
Car Evaluation	19	1728

4. Proposed Apriori Algorithm (PAA)

In this section, the transaction-storage table is described, followed by an explanation of how the frequent itemsets are generated (with an example). Finally, the numbers of searched records are compared.

4.1. Transaction Storage Table

Since the Apriori algorithm is a breadth-first algorithm, the entire dataset must be scanned from left to right and top to bottom in order to find the support of an item (based on Table 1). To avoid scanning the entire dataset to find support, a change in the transaction storage table (the changed TD) is made. In the first step, each column represents an item and each row represents a purchase. For example, consider the TD dataset (Table 1). Based on this dataset, the number of columns will be five, and the number of rows will be six. To calculate the support for each item, the number of occurrences is counted in each item's column. Also, the TID transactions that contain these items are stored (Table 3).

Table 3
Changed TD

TID	Item A	Item B	Item C	Item D	Item E
T_{100}	A		C	D	E
T_{200}		B	C		E
T_{300}	A	B	C		E
T_{400}		B			E
T_{500}	A	B	C		E
T_{600}				D	
Support	3	4	4	2	5
TID-Support	T_{100} T_{300} T_{500}	T_{200} T_{300} T_{400} T_{500}	T_{100} T_{200} T_{300} T_{500}	T_{100} T_{600}	T_{100} T_{200} T_{300} T_{400} T_{500}

For instance, there is an Item A in each of the T_{100}, T_{300}, and T_{500} transactions. In the 1-itemset (Table 3), these TIDs are stored in the last row of each column (item). For the other k-itemsets (k >1), a column is added for each itemset according to Figure 2, and the TIDs that contain this item are stored.

As can be observed in Table 3, the set of transactions that contain this item is stored for each item.

4.2. Generating Frequent Itemsets

In CAA, the dual itemsets are created first; then, some items are pruned based on the value of the minimum support (minsup). In the next step, an item is added to the itemsets; this process continues until it finally reaches one or more frequent itemsets. However, we strive to act intelligently in PAA (besides, there is no need to add the items one at a time). This indicates that there will be multiple items instead of dual items in the C2 step. For this purpose, we first sort each column of the table by the value of the support for this column (from large to small – Table 4). After sorting, the column whose support value is less than the minsup is removed; then, a frequent 1-itemset (L1) is created.

Table 4
Frequent 1-itemset (L1)

TID	Item E	Item B	Item C	Item A
T_{100}	E		C	A
T_{200}	E	B	C	
T_{300}	E	B	C	A
T_{400}	E	B		
T_{500}	E	B	C	A
T_{600}				
Support	5	4	4	3
TID-Support	T_{100} T_{200} T_{300} T_{400} T_{500}	T_{200} T_{300} T_{400} T_{500}	T_{100} T_{200} T_{300} T_{500}	T_{100} T_{300} T_{500}

As can be seen from Table 4, Items C and B have the same support value. Since Items B and C are in the same position, we combine all of the items (namely B, C, and E) instead of just combining B with E. In other words, multiple items can be combined in each step instead of one item – provided that the added items have an equal value of support. Subsequently, EBC is combined with A, and EBCA is created.

The next step is to generate the frequent 2-itemsets from L1. To receive the value of support for each itemset, the itemset is searched among the TIDs that are

stored in this item instead of searching in the total dataset records. Between two items, the transactions of the item that has the minimum number of stored TIDs are considered, and the itemset search to count the value of the support is performed only between these transactions. For example, considering the first item in Figure 2 (E, B) based on the original Apriori (CAA), we scan all six transactions to find itemset (E, B). However, we split itemset (E, B) into E and B in PAA and receive the minimum support between them by using L1. Here, B has the smallest minimum support.

Next, itemset E, B is searched in the transactions that are contained in Item B (these TIDs were stored in Table L1). Therefore, we only search for itemset E, B in transactions T_{200}, T_{300}, T_{400}, and T_{500}. This method results in the fact that the number of steps for running the algorithm are significantly reduced, as are the number of scanned transactions for creating the itemsets. The steps for performing the operation are displayed in Figure 2.

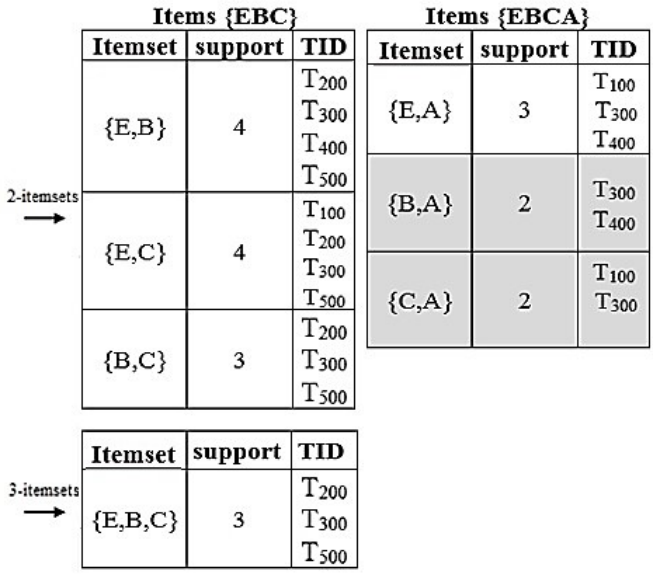

Figure 2. Operation process of CAA

According to Figure 2, all of the dual combinations are created in the 2-itemsets, and the duplicate itemsets are removed. Then, the value of the support is obtained, and the pruning is done (those rows that had less support than the minsup are in gray). In the 3-itemsets, there is only one itemset whose support value is 3; therefore, this item is a frequent itemset. In fact, PAA is an intelligent approach that discovers all of the frequent itemsets.

The PAA pseudo-code for discovering the frequent itemsets for mining is given below (Algorithm 2):

Algorithm 2 PAA pseudo-code

```
start
Define minimum support and k_i
Create 1-itemset = Create new table where each column represents item and each row represents transaction
For i = 1 in range (Number of items):
    Obtain support value for this item
    Store transactions that contain this item
Prune 1-itemset
Sort 1-itemset based on value of support
Combine items from 1-itemset that have same support value
While ki-itemset    :
    Obtain all compounds with ki number
    For i = 0 in range (length [compounds]):
        Obtain support value for each itemset from TIDs
    Prune ki-itemset
    Store transactions
```

To express this more clearly, the PAA is described by means of a flow chart in Figure 3; then, the PAA pseudo-code will be presented.

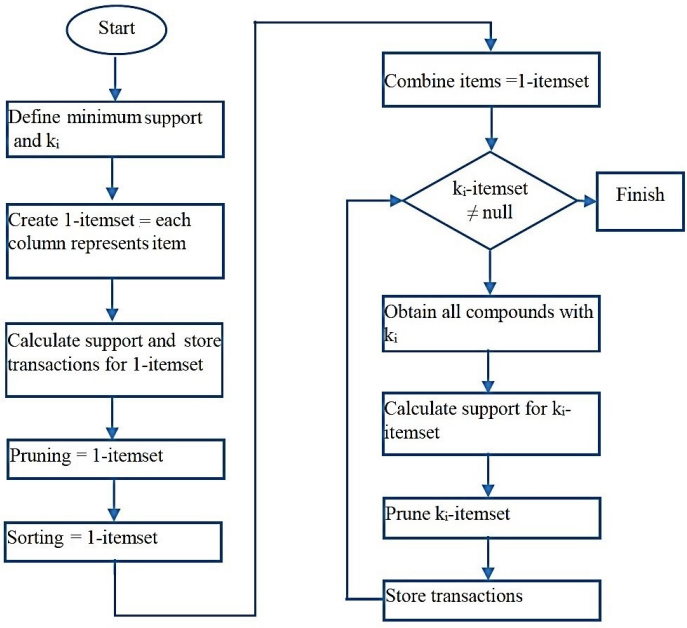

Figure 3. Flowchart of proposed approach

The time-complexity analysis of an Apriori algorithm is affected by the minsup, the number of items, and the number of transactions: if the minsup value is low,

the number of itemsets in each step will increase; if this value is high, the number of itemsets in each step will decrease (and the program's run speed will increase). Likewise, the number of items and the number of transactions affect the execution time of the algorithm. In PAA, creating the 1-itemset (Line 5) is of time order O(NM), where M is the number of items, and N is the number of transactions. The pruning time (Line 6) is equal to the number of times the support value is checked for each item; this is the same as the number of items, so it is equal to O(M). The time order that is related to the 1-itemset sort that comes with the columnar displacements (Line 7) is equal to O(M^2). The combination of the items (Line 8) in the worst case (if the items do not have the same support value) is equal to O(M). The ninth step is to create the frequent itemsets (Line 9). In the worst case, none of the items may have the same value of support; in this case, the total number of compounds that are made in each step is O(M^2). In addition, the number of steps is equal to the number of items (M). The time that is required to obtain the support value for each compound is N; therefore, the total time order of Line 6 is O(NM^3).

4.3. Comparing Numbers of Searched Records

In CAA, all of the dataset records must be repeatedly scanned in order to obtain every frequent k-itemset (Lk). In the example given in Section 1.1, 30 records were compared to obtain the L1s (number of items * number of dataset records). To obtain the L2s, the number of searched records was 36; for the L3s, the number was equal to 9 records. The total number of scanned records equaled 75. To obtain every frequent k-itemset (Lk) in PAA, it is not necessary to scan an entire dataset at every step. Instead, we need to scan the first step (L1) of the entire dataset (which will not happen in the following steps). It should also be noted that the number of records that were searched for obtaining the L1s was equal to 30. In the subsequent step, to obtain the L2s and L3s, 21 and 3 records were to be searched, respectively (in the third step, there was only one 3-itemset member – including EBC). Its two-membered subsets included EB, EC, and BC. Of these subsets, we selected a two-member set whose minsup had been kept to a minimum, and we performed a search on its TIDs (here, the BC was selected, and the search was performed on the T_{200}, T_{300}, and T_{500} transactions). Finally, the total number of records that were searched in this example with PAA was found to be equal to 54. Table 5 shows a comparison between CAA and PAA in terms of the number of scanned transactions.

Table 5
Number of scanned transactions

k-itemset	CAA	PAA
1-itemset	30	30
2-itemset	36	21
3-itemset	9	3
Total	75	54

By storing the number of records for each item, the PAA algorithm stops the entire dataset from being scanned at each step. This innovative method minimizes the number of references to each record. Moreover, it decreases the algorithms execution time when generating a frequent itemset.

4.4. Evaluation of memory usage

Table 6 exhibits the amount of memory that was used for each dataset that had different minimum supports.

Table 6
Memory usage

Mushroom		Congressional Voting		Car Evaluation	
Min-sup	Memory Usage (Byte)	Min-sup	Memory Usage (Byte)	Min-sup	Memory Usage (Byte)
0.06	131.829	0.15	43.419	0.02	13.140
0.08	72.540	0.20	25.028	0.03	3.984
0.10	59.371	0.25	18.696	0.04	3.984
0.12	47.914	0.30	5.672	0.05	3.984
0.14	38.612	0.35	2.748	0.06	3.984
0.16	26.112	0.40	1.760	0.07	3.680
0.18	18.732	0.45	972	0.08	3.680
0.20	14.020	0.5	432	0.09	836

5. Experimental Result

In the current study, three experiments were implemented to evaluate the performance of the proposed algorithm. Each experiment used different datasets (which varied widely in terms of the numbers of records and items) and provided us with the appropriate conditions for running the experiment and establishing the required comparisons. The density of the datasets (the number of non-empty entries divided by the total number of entries) was also quite varied. The three experiments were conducted to prove the efficiency of this algorithm as follows.

5.1. Results of Experiment 1

In Experiment 1, PAA was compared with the BitTableFI algorithm. The dataset that was used in this experiment was Mushroom, which contained 8416 records and 120 items. This simulation was implemented in C++ and compiled with Microsoft Visual C++ on a personal Intel computer with 2.4 GHz CPU and 1GB memory and running in the Windows XP operating environment. Table 7 and Figure 4 illustrate the results that are related to Experiment 1.

Table 7
Time-consuming comparison of CAA, MDC-Apriori algorithm, and PAA

minsup	CAA (sec)	BitTableFI (sec)	PAA (sec)	Time-reduction Rate of PAA Relative to BitTableFI (%)
0.04	550	91.4	5.06	95
0.06	251	31.4	4.42	88
0.08	161	18.76	3.83	82
0.1	142	11.34	3.14	75
0.12	95	5.23	3.01	48
0.14	49.72	3.41	2.82	23
0.16	21.4	3.32	2.72	24
0.18	18.93	3.12	2.49	24
0.20	14.32	2.91	1.91	34

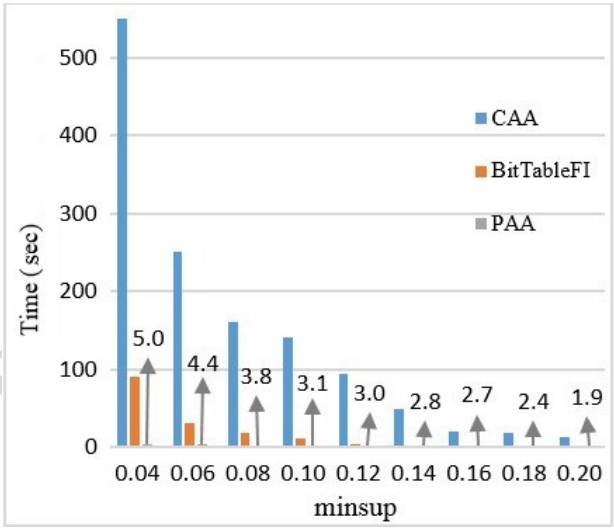

Figure 4. Comparison among different values of minimum support and time consumption for CAA, BitTableFI algorithm, and PAA

As can be seen in Table 7, the execution time of the PAA algorithm was less than BitTableFI and CAA with different minsups.

5.2. Results of Experiment 2

In Experiment 2, PAA was compared with the TDM-MFI algorithm. The dataset that was used in this experiment was Congressional Voting, which contained 435 records and 17 items. This experiment was run on a PC with P4-3 GHz and 1 GB of main

memory. The programs were written in C++. Table 8 and Figure 5 show the results of Experiment 2.

Table 8
Time-consuming comparison of CAA, TDM-MFI algorithm, and PAA

minsup	CAA (sec)	TDM-MFI (sec)	PAA (sec)	Time-reduction Rate of PAA Relative to TDM-MFI (%)
0.2	14.2	0.74	0.54	27
0.3	4.1	0.59	0.16	73
0.4	1.9	0.52	0.08	85
0.5	0.5	0.12	0.02	84

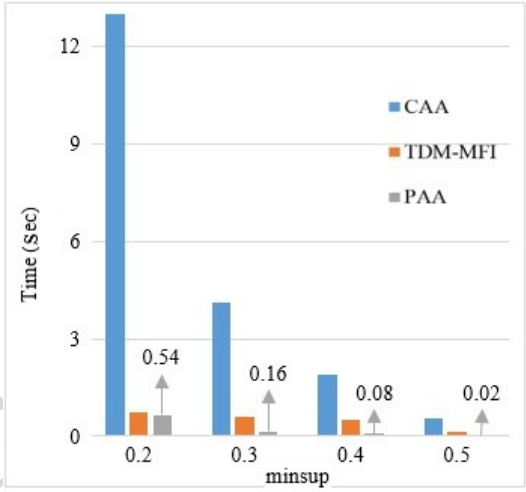

Figure 5. Comparison among different values of minimum support and time consumption for CAA, TDM-MFI algorithm, and PAA

Based on the results that are shown in Table 7, the implementation of the PAS with a different dataset still yielded better results than the algorithm that was compared in this experiment.

5.3. Results of Experiment 3

In Experiment 3, PAA was compared with the MDC-Apriori algorithm. The dataset that was used in this experiment was Car Evaluation, which contained 1728 records and 19 items. This experiment was run on a PC with Core i5, 2.67 GHz CPU, and 2 GB of memory. The programs were written in Java. Table 9 and Figure 6 represent the results that are related to Experiment 3.

Table 9
Time-consuming comparison of CAA, MDC-Apriori algorithm, and PAA

minsup	CAA (sec)	MDC-Apriori (sec) (sec)	PAA (sec)	Time-reduction Rate of PAA Relative to MDC-Apriori (%)
0.02	71.4	4.8	3.80	21
0.04	28.62	2.5	1.28	49
0.06	27.34	2.3	1.30	43
0.08	15.31	1.1	0.86	22
0.10	4.93	0.93	0.26	72
0.12	3.7	0.7	0.22	69

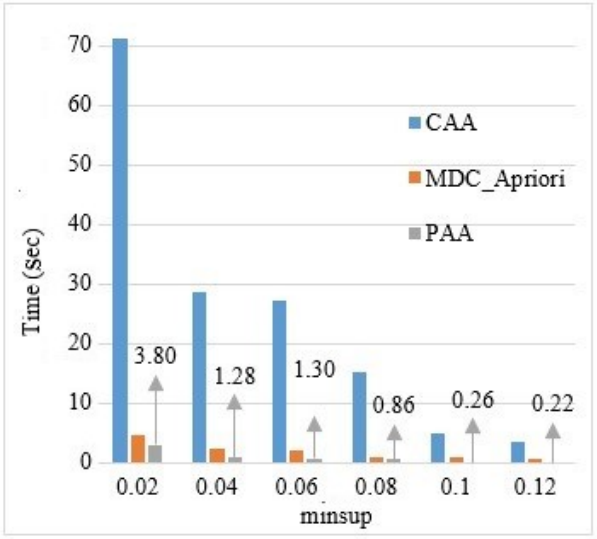

Figure 6. Comparison among different values of minimum support and time consumption for CAA, MDC-Apriori algorithm, and PAA

With a rise in minimum support, there will be a decline in the number of frequent itemsets; this greatly reduces the algorithm's running time. The experimental results acknowledge that, when compared with the classical Apriori, BitTableFI, TDM-MFI, and MDC-Apriori algorithms, the execution times of PAA are greatly reduced. In Table 10, the average PAA runtime reduction is shown as compared to the other three experiments.

Table 10
Time-consuming comparison of CAA, MDC-Apriori algorithm, and PAA

Proposed Algorithm	Average Reduction in Execution Time (%)
BitTableFI	52
TDM-MFI	65
MDC-Apriori	46

Therefore, the execution time of the PAA algorithm was significantly reduced when compared to the other algorithms.

6. Conclusions

A major challenge in data mining is to apply the Apriori algorithm for detecting frequent items. Bearing this in mind, this research has introduced a method for improving the performance of the Apriori algorithm in which those items that are similar to the values of their supports are merged while storing the TID transactions of each item. Afterwards, the search operation was performed to create the next itemset in only the saved transactions. This significantly minimized the number of scans in the dataset, and the entire duplicate items were iltimately discovered. In order to quantitatively and qualitatively evaluate the performance of the proposed improved Apriori algorithm, three experiments were implemented with the BitTableFI, TDM-MFI, and MDC-APRIORI algorithms. It was observed that the average runtimes of the PAA algorithm were reduced by 52, 65, and 46%, respectively, when compared to the three mentioned algorithms. With such reduced runtimes, the proposed Apriori algorithm can be very useful for large datasets. In the future, there can be a further enhancement in the performance by merging this algorithm using the BitTable method.

References

[1] Agrawal R., Imielinski T., Swami A.: Mining association rules between sets of items in large databases. In: *ACM-SIGMOD International Conference Management of Data*, pp. 207–216, 1993.

[2] Agrawal R., Imielinski T., Swami A.: Research of an Improved Apriori Algorithm in Data Mining Association Rules. In: *International Conference on Service Systems and Service Management*, 2007.

[3] Benhamouda N.C., Drias H., Hirche C.: An efficient way to find frequent pattern with dynamic programming approach. In: *Nirma University International Conference on Engineering*, 2013.

[4] Bhalodiya D., Patel K.M., Patel C.: Meta-Apriori: A New Algorithm for Frequent Pattern Detection. In: *International Conference on Information and Communication Technologies*, 2016.

[5] Bhandari A., Gupta A., Das D.: Improvised Apriori algorithm using frequent pattern tree for real time applications in data mining. In: *International Conference on Information and Communication Technologies*, 2014.

[6] Cheng X., Su S., Xu S., Li Z.: DP-Apriori: A deferentially private frequent itemset mining algorithm based on transaction splitting, *Computers and Security*, vol. 50, pp. 74–90, 2015.

[7] Dong J., Han M.: BitTableFI: An efficient mining frequent itemsets algorithm, *Knowledge-Based Systems*, vol. 20, pp. 339–335, 2007.

[8] Duong H.V., Truong T.C.: An efficient method for mining association rules based on minimum single constraint, *Vietnam Journal of Computer Science*, vol. 2, pp. 67–83, 2015.

[9] Han J., Kamber M.: *Data Mining Concepts and Techniquesr*, Morgan Kaufmann Publishers, 2006.

[10] Jie Z., Gang W.: Intelligence Data Mining Based on Improved Apriori Algorithm, *Journal of Computers*, vol. 14, pp. 52–62, 2019.

[11] Liu X., Zhai K., Pedrycz W.: An improved association rules mining method, Expert Systems with Applications, *Expert Systems with Applications*, vol. 39, p. 13621374, 2012.

[12] Liu Y., Li Y., Yang J., Ren Y., Sun G., Li Q.: An Improved Apriori Algorithm Based on Matrix and Double Correlation Profit Constraint, *Communications in Computer and Information Science*, vol. 901, 2018.

[13] Sun L.: An improved Apriori algorithm based on support weight matrix for data mining in transaction database, *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, pp. 495–501, 2020.

[14] Yu H., Wen J., Wang H., Jun L.: An Improved Apriori Algorithm Based On the Boolean Matrix and Hadoop, *Procedia Engineering*, vol. 15, pp. 1827–1831, 2011. doi: https://doi.org/10.1016/j.proeng.2011.08.340.

[15] Yu H., Wen J., Wang H., Jun L.: Association rule mining algorithms on high-dimensional datasets, *Artificial Life and Robotics*, vol. 23, pp. 420–427, 2018.

Affiliations

Noorollah Karimtabar
The University Of Isfahan, Faculty Of Computer Engineering, Isfahan, Iran; e-mail: karimtabar@eng.ui.ac.ir

Mohammad Javad Shayegan Fard
The University of Science and Culture, Department of Computer Engineering, Tehran, Iran; e-mail: Shayegan@usc.ac.ir

Received: 29.04.2020
Revised: 01.06.2021
Accepted: 09.07.2021