Eng. Sci., 2022, **20**, 284–295

Engineered Science

DOI: https://dx.doi.org/10.30919/es8d731

Partial Weighted Count Tree for Discovery of Rare and Frequent Itemsets

Shwetha Rai,[1] Geetha M.,[1,*] Preetham Kumar[2] and Giridhar B.[1]

Abstract

Time and space utilization for discovering interesting patterns from a database plays an important role in analyzing information for major sectors like education, medicine, and e-business. Association rule mining (ARM) technique is used to discover associations among the patterns from large volumes of data. In most ARM algorithms, rare and frequent itemsets discovery is optimized by mining pruned databases stored in the main memory. However, in this case, any change in requirements would necessitate re-scanning of the database. Weighted count tree (WC-Tree), and Single scan pattern tree (SSP-Tree) store the database in the main memory without pruning. WC-Tree stores the entire transaction as a node in the tree. However, if the weight is large, the actual information may be lost due to the precision error. In the current work, an efficient data structure, Partial weighted count tree (PWC-Tree), is proposed to store the database as a complete and compact structure in the main memory without losing the information. The work revealed that PWC-Tree construction is in $O(n^2)$ for n transactions in the database. The experimental results show that, for a large dataset, the PWC-Tree is time as well as space-efficient when compared with WC-Tree and SSP-Tree.

Keywords: Association Rule Mining; PWC-Tree; Frequent itemsets; Rare itemsets; Tree data structure.
Received: 18 March 2022; Revised: 25 May 2022; Accepted: 27 May 2022.
Article type: Research article.

1. Introduction

Data is generated by digital devices or users using the digital devices in various sectors like healthcare, education, government, online stores, and social media.[1-6] The raw data is processed to identify trends and change the dynamics of decision-making based on those trends. Various data analytics companies offer services to industries to make their way in business.[7] Data mining (DM) is one of the phases in the process of data analytics which comprises several methods and algorithms used to discover interesting patterns in databases.[8] Various techniques in data mining are used for prediction and analysis in healthcare, weather forecast, and business.[9,10] Association rule mining (ARM) is an unsupervised technique in DM to study the relationships between the itemsets in a database that has attracted significant attention among DM researchers.[11] An itemset can be classified as either rare or frequent based on its occurrence in the database. The interpretation of the relation between the itemsets can assist in the discovery of precise knowledge about the problem under consideration.

The application of frequent and rare itemset mining in ARM span various fields and it is an intermediate step in the analysis of data for various problems.[12] Several algorithms were implemented to discover the associations among itemsets, and the efficiency of these algorithms depends on the data structures that are used. Most algorithms require multiple database scans to discover interesting patterns that incur additional costs due to data transfer from secondary memory to main memory.[13] In recent years few algorithms have been proposed that scan the database only once to discover interesting patterns.[14–18] Apriori algorithm, the first algorithm to implement the ARM technique scans the database multiple times to discover frequent itemsets. Several algorithms were proposed based on the Apriori algorithm which scans the database multiple times to discover interesting patterns.[11] Direct Hashing & Pruning algorithm utilizes a hash-based technique to store the itemsets.[19] Dynamic Itemset Counting is a variation of the Apriori algorithm that scans the database multiple times requiring more I/O operations and hence is not cost-effective.[20] Partition algorithm, a two-scan algorithm

[1] *Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India.*
[2] *Department of Information and Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India.*
**E-mail:* geetha.maiya@manipal.edu (G. Maiya)

generates all potentially large itemsets by logically dividing the database into mutually exclusive partitions.[21] Frequent pattern tree, a non-candidate generation prefix tree structure algorithm requires two scans of the database to mine frequent itemsets.[22] UF-tree and Transaction mapping algorithm are variations of FP-tree which scan the database twice and use a similar tree structure.[23,24]

The Perfect Hashing and Pruning algorithm use a hash table to discover the itemsets. It uses a recursive hashing procedure similar to the Lexicographical Ordered tree.[25] Eclat uses a non-conventional vertical representation to store the input database. There are several variations in the algorithm such as tidset and diffset to discover the frequent itemsets.[26] Pattern-Count tree algorithm is constructed in one database scan from which a Lexicographically Ordered FP-Tree was constructed to discover frequent itemsets.[27] A Pattern-tree is constructed in a single scan by arranging the transaction items in alphabetical order and then inserting them into the tree.[28] A compact and complete tree called as Weighted Count-Tree stores the transaction items in the main memory and discovers all closed frequent concepts. Each item in the transaction is represented using a unique prime number. The product of these prime numbers is stored in a single node of the tree i.e., each node represents a transaction. The memory space used to store the dataset in the main memory is reduced significantly in the WC-Tree algorithm.[29] Single Scan-Frequent Itemset Mining algorithm scans the database once to discover all frequent itemsets. It uses a hash-based approach to store the powerset of all transaction items with its count.[16] An improved apriori algorithm scans the database and creates a 0-1 matrix where the row represents the transactions, and the column represents the items in the transaction. 0 represents the absence of the item and 1 represents its presence in the transaction.[17] ARIMA, a two-step algorithm, generates all frequent and infrequent candidate itemsets.[30] AfRIM algorithm is used to mine rare itemsets using a top-down approach. The itemsets are pruned based on anti-monotone property. The algorithm considers itemsets with zero support count, the cases which never occurred, as rare itemsets.[31] ARANIM algorithm classifies the itemsets as frequent, rare, and non-present itemsets. The algorithm was used to identify suspicious activities in the weblog.[32] Rare Pattern Tree (RP-Tree) mining was designed based on FP-Tree to discover rare itemsets based on two support values, lower and upper.[33] A hyper-linked data structure was implemented to mine rare itemsets from a sparse database using two database scans.[34] CARM algorithm generates association rules in which there can be only one item consequent, but more than one antecedent item.[35] Postdiffset algorithm, based on Diffset algorithm, discovers frequent itemsets where the initial looping is based on tidsets process and second looping is for obtaining the result diffset.[36,37] Single Scan Pattern-Tree is constructed and restructured based on the frequency of the itemsets. The header table stores the itemsets and their support count in sorted order. The header table is updated to maintain the sorted order and the tree is also restructured accordingly.[17,38] R-Eclat algorithm discovers rare or infrequent itemsets from the database based on the Eclat algorithm which uses a vertical representation of the transaction database.[39]

Based on data dimensions, its size, and user-defined thresholds, the space utilized to store and discover frequent and/or rare itemsets can vary from a few kilobytes to a situation where the main memory is insufficient. Further, the time taken to discover the itemsets may also vary based on the nature of the dataset. Motivated by the challenge of designing an efficient algorithm to store the itemsets in the main memory for discovering frequent and/or rare itemsets, a novel Partial Weighted Count Tree (PWC-Tree) algorithm to store and retrieve rare and frequent itemsets without information loss is proposed in this article.

2. Methodology

PWC-Tree stores the entire database in the main memory in a single database scan without information loss. The data structure is designed to enable the efficient discovery of frequent and rare itemsets from PWC-Tree even when the support threshold is changed without any loss of information. The algorithm uses the uniqueness of prime numbers to represent the items in the transaction of the database, called the weight of a transaction. Each item in the transaction is mapped to a prime number and the product of these prime numbers is considered the weight of the transaction. For example, if there are 3 items in the transactions then, they are mapped to the first three prime numbers 2, 3 and 5 respectively and the weight of the transaction is calculated as $2 \times 3 \times 5 = 30$. Also, given any weight, the item numbers can be retrieved since the prime factors of the weight are always unique. For example, if the weight of the transaction is 30 then, its prime factors are 2, 3, 5 and corresponding item numbers are 1, 2 and 3.

2.1 Structure of a node in Partial Weighted Count Tree

Every node in PWC-Tree stores the Weight and the corresponding count. There are two types of nodes in PWC-Tree contains two types of nodes, NODE and REMNODE, to store the transactions in the compact form. The structure of NODE is shown in Fig. 1, which contains Weight, Count, Flag to indicate the existence of REMNODE, a Next_Pointer to the sibling, or RMNODE based on Flag value, and a Child_Pointer to the child node. NODE will hold the maximum possible weight and the remaining constituent elements are stored in REMNODE pointed by Next_Pointer of the current node. The structure of REMNODE is shown in Fig. 2 and it contains Partial_Weight, Flag to indicate the presence of REMNODE, and Next_Pointer to either sibling or the next REMNODE.

2.2 Construction of Partial Weighted Count Tree

PWC-Tree is constructed in one database scan and the procedure is shown in algorithm 1. The steps to construct the

tree are as follows:

1. Read items from the transaction, one at a time, and map every item number of the transaction to a corresponding prime number.
2. If the tree is empty, find the weight of the transaction. If the weight is lesser than the limit (INT MAX), create a NODE with the flag 'n', the associated weight, count as 1, and attach it to the child of the root. If the weight is greater, create a NODE with flag = r' and REMNODEs attached to it, then attach the NODE to the child of the root.
3. If the tree is not empty, start traversing from the child of the root and traverse the tree based on the following cases until the transaction is inserted.

a. If the traversed node's value is equal to the transaction weight, increment the count of the traversed node by a value of 1.

b. If the transaction's weight is divisible by the traversed node's value, move to the child of the traversed node, unless it is NULL. If the child of the traversed node is NULL, remove all common factors between parent and transaction, create a node similar to step 2, and add this created node as the child of the traversed node.

c. If the traversed node's value is divisible by the transaction's weight, create a node similar to step 2 and add this as the parent of the traversed node. Move all the siblings of the traversed node whose values are not divisible by the transaction's weight as the siblings of the created node. Remove the factors that are common to the parent and child from the child node.

d. If transaction weight and traversed node's value is not divisible, move to the right of the traversed node unless it is NULL. If it is NULL, create a node similar to step 2 and add it to the right of the traversed node.

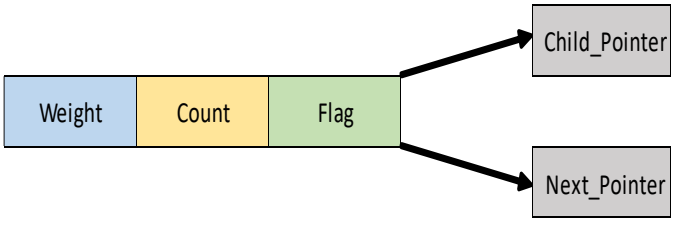

Fig. 1 Structure of NODE with Weight, Count, Flag, Child_Pointer and Next_Pointer.

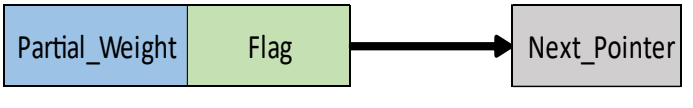

Fig. 2 Structure of REMNODE with Partial_Weight, Flag, and Next_Pointer.

2.3 Mining itemsets from Partial Weighted Count Tree

The itemsets from the PWC-Tree are discovered using the following steps:

1. Create a Hash Table to store the itemsets along with its support.
2. For each node in the tree generate the constituent primes of the traversed node and a subset for this is generated. If the node is a child node, then perform a join operation with a subset generated from constituent primes of the parent node.
3. Add the constituent primes of each subset, hash this sum using the hash and add them to the respective position in the Hash Table.
4. If the value is already present in the hash table, then add the count in the hash table with the count of the traversed node. Otherwise, insert the value to the hash table initializing its count to the count of the traversed node.
5. Once all the nodes are traversed, traverse through the hash table and display all the table entries whose count is between the threshold value set by the user.

2.4 Illustration of the Partial Weighted Count tree

Table 1 shows a sample database with 8 transactions and the transactions' corresponding weight. PWC-Tree is constructed for the sample database in Table 1 and is shown in Fig. 3. Here, the data type for storing the weight is assumed to be unsigned int.

Table 1. Sample Database with transactions' Weight.

TID	Transaction items	Weight of Transaction
1	1,2	2×3=6
2	1,3	2×5=10
3	3,4	5×7=35
4	4	7
5	1,2,3	2×3×5=30
6	1,2,3,65,66,67	2×3×5×313×317×331= 98,52,64,530
7	1,2,3,57,58,59,60,61	2×3×5×269×271×277×281×283= 4,81,74,29,30,40,870
8	1,2,3,62,63,64	2×3×5×293×307×311= 83,92,42,830

At the beginning of the algorithm, the first transaction is read, and its corresponding Weight is calculated. It is then inserted as the first child of the root node and the support count is set to 1. In the next step, the second transaction is read, and its Weight is calculated, Since the Weights are not divisible, it is inserted as a sibling node of the first transaction. The insertion of the third transaction is similar to the second transaction. The fourth transaction is inserted as the parent of the third transaction because Weight (T4) divides Weight (T3). The common factor {4} is removed from the Weight of transaction 4 and the support of the node is incremented to 2 before inserting transaction 4. Transaction 3 is attached as the child of transaction 4 and the support count is retained as 1. Similarly, the remaining nodes are inserted into the tree after checking for the divisibility conditions between the Weight of

transactions as mentioned in the steps. *{itemset: Support_count}* represents the combination itemset and its support count in the database. The itemsets, frequent and rare, discovered from PWC-Tree shown in Fig. 3 with support thresholds min_{rare} = 20% and min_{freq} = 50% are:

1. Frequent: {{1:6},{2:5},{3:6},{1,2:5},{1,3:5}}
2. Rare:{4:2}

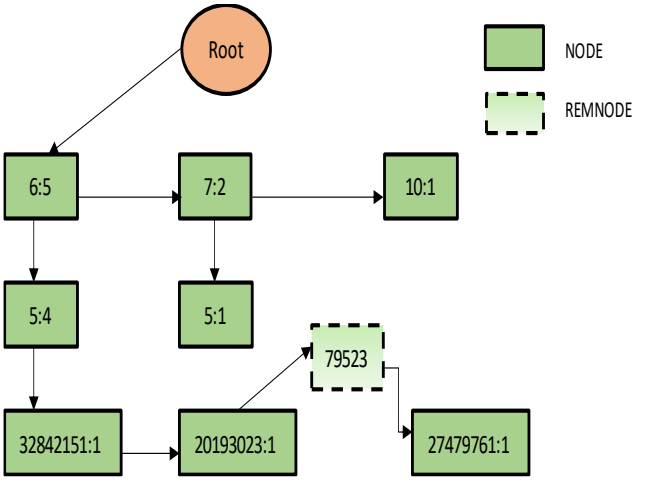

Fig. 3 Partial Weighted Count Tree for the transactions in Table 1.

3. Results and discussion

3.1 Datasets used

For the experimental evaluation of algorithms, five commonly employed secondary datasets are used to discover frequent and rare itemsets. These datasets are downloaded from the Sequential Pattern Mining Framework repository, and their description is given in Table 2.[40] $\#T_{Items}$ gives the number of items in the dataset, $\#D_{Transaction}$ gives the number of transactions in the dataset, and *Avg.length* is the average number of items in a transaction in the dataset.

Table 2. Description of the datasets.

Dataset	$\#T_{Items}$	$\#D_{Transaction}$	Avg. length	Type
Skin	11	245057	4	Real
Mushroom	119	8124	23	Real
KDDcup99	135	927393	16	Real
Connect	129	67557	43	Real
Chess	75	3196	37	Real

3.2 Running environment

The algorithms are implemented in C++ and executed on a computer with Ubuntu 18.04.5 LTS x64 operating system, 8GB RAM, and an Intel Core i5 at 3GHz processor.

3.3 Results

The execution performance of PWC-Tree is compared with WC-Tree and SSP-Tree, the algorithms that store the complete database in the main memory using a single database scan. SSP-Tree is one of the recent algorithms that was proved to be efficient when compared to other state-of-the-art algorithms in the same area. PWC-Tree and WC-Tree belong to the same family of algorithms in which the transactions are represented using Weights in the node. WC-Tree stores smaller Weights in a smaller capacity node and larger numbers are represented as the quotient of the number and the largest value that can be represented in the algorithm. The algorithms are executed by varying the data dimension and size.

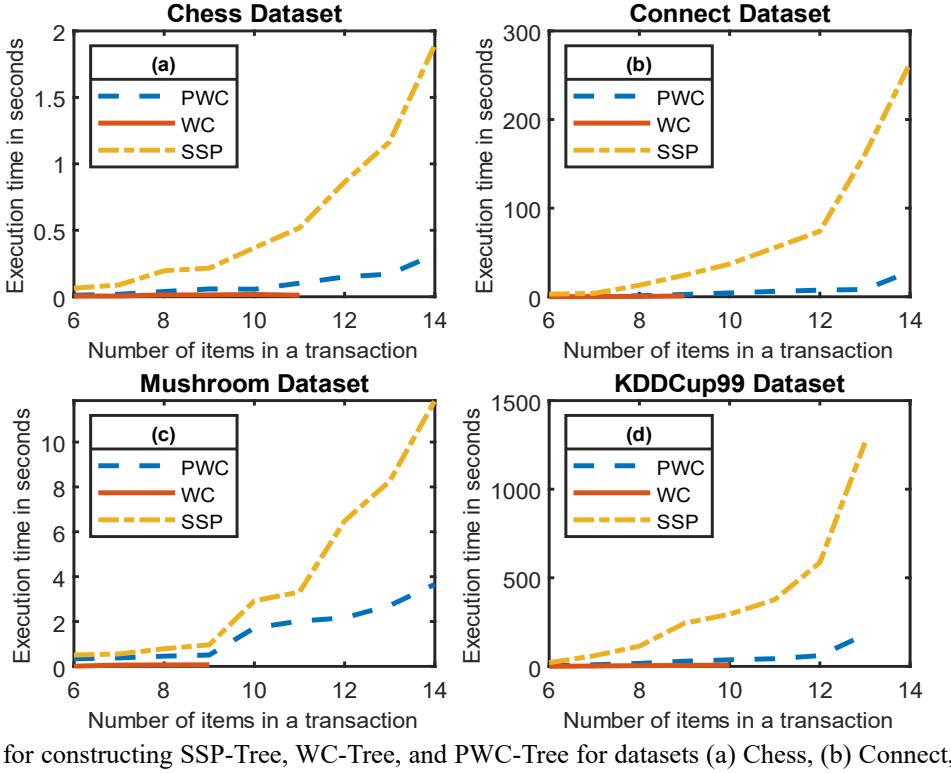

Fig. 4 Execution time for constructing SSP-Tree, WC-Tree, and PWC-Tree for datasets (a) Chess, (b) Connect, (c) Mushroom, and (d) KDDCup99.

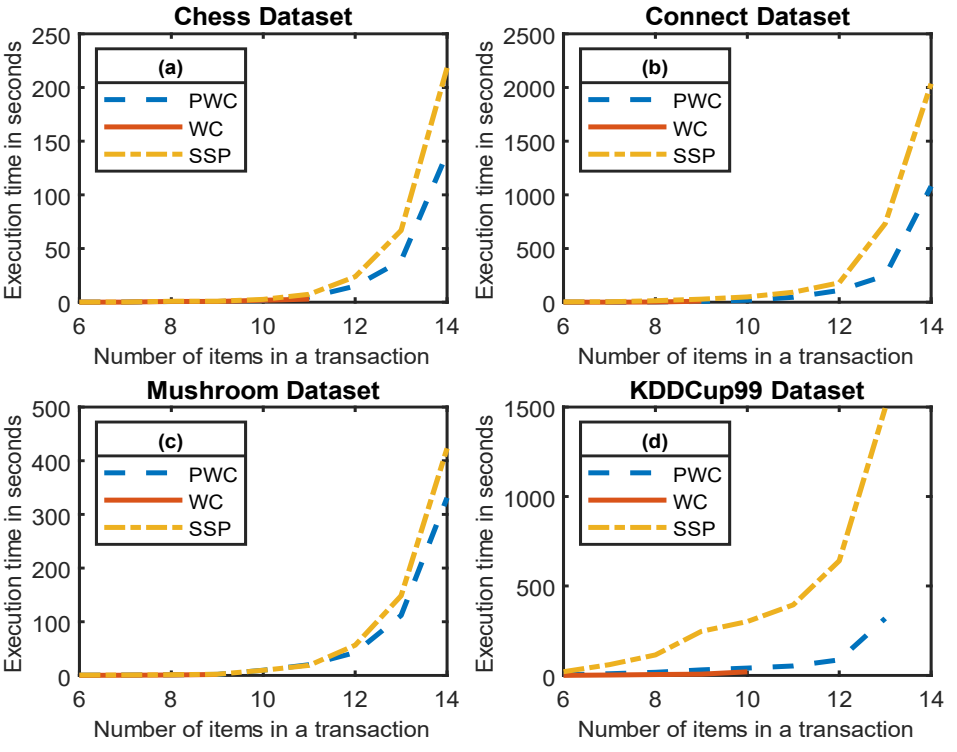

Fig. 5 Execution time for constructing and discovering itemsets from SSP-Tree, WC-Tree, and PWC-Tree for datasets (a) Chess, (b) Connect, (c) Mushroom, and (d) KDDCup99.

3.3.1 Execution results by varying data dimension

The experiment is conducted to understand the behavior of WC-Tree, PWC-Tree, and SSP-Tree, for time and space efficiencies, by varying the number of items in the transaction from 4 to 14. While experimenting only rare itemsets are discovered by setting the threshold values of min_{rare} and min_{freq} to 0 and 100 respectively. The algorithms were executed on all datasets except the Skin dataset since it has only four items per transaction and the minimum dimension considered is 4. While SSP-Tree and PWC-Tree algorithms are executed on all other datasets, the WC-Tree algorithm is executed only on Chess, Connect, KDDCup99, and Mushroom datasets due to the limitation of storing large weights in the tree.

Fig. 4 shows the time taken to construct PWC-Tree, WC-Tree, and SSP-Tree. It is observed that WC-Tree is efficient when compared to PWC-Tree and SSP-Tree in terms of tree construction time but after a certain number of items in the transaction, WC-Tree cannot store the items in the node due to its limitation to store the weight of the transaction that is greater than the size of the datatype used during implementation. The construction time of the SSP-Tree increases exponentially with the increase in the number of items per transaction due to restructuring of the tree before inserting each transaction.

Fig. 5 shows the total time taken to construct and mine from WC-Tree, SSP-Tree, and PWC-Tree. WC-Tree performs better than PWC-Tree and SSP-Tree only when the number of items in the transaction is limited. PWC-Tree is found to be time-efficient as the number of items increases in the dataset. Fig. 6 shows the memory allocated in bytes to store the dataset in WC-Tree, SSP-Tree, and PWC-Tree respectively. It can be observed that PWC-Tree utilizes a lesser amount of memory when compared with the amount of memory utilized by WC-Tree and SSP-Tree.

3.3.2 Execution results by varying data size

The experiments are conducted by considering the dimensions of the datasets such that WC-Tree stores the weight of the transaction without losing the information. WC-Tree, SSP-Tree, and PWC-Tree are executed on all datasets.

Fig. 7 shows the time taken for the construction of WC, SSP, and PWC trees. WC- Tree is 79% more efficient when compared to PWC-Tree and SSP-Tree in terms of tree construction provided there is a limited number of items per transaction. PWC-Tree shows 88% better performance when compared with SSP-Tree during the tree construction.

Fig. 8 shows the total time taken for the construction and mining of WC-Tree, SSP-Tree, and PWC-Tree. During the discovery of frequent and rare itemsets by setting min_{freq} threshold to 20% and min_{rare} to 10%, it is found that PWC-Tree is 28% more time-efficient than WC-Tree as the redundant subset generation is avoided by storing the common factors in the parent node. But SSP-Tree is 68% more efficient than the PWC tree because the itemsets in the nodes of the SSP-Tree are already sorted before insertion and hence all the nodes need not be visited if the itemsets' support does not satisfy the user-defined support threshold. Whereas in PWC-Tree all nodes must be visited to check the support of each itemset.

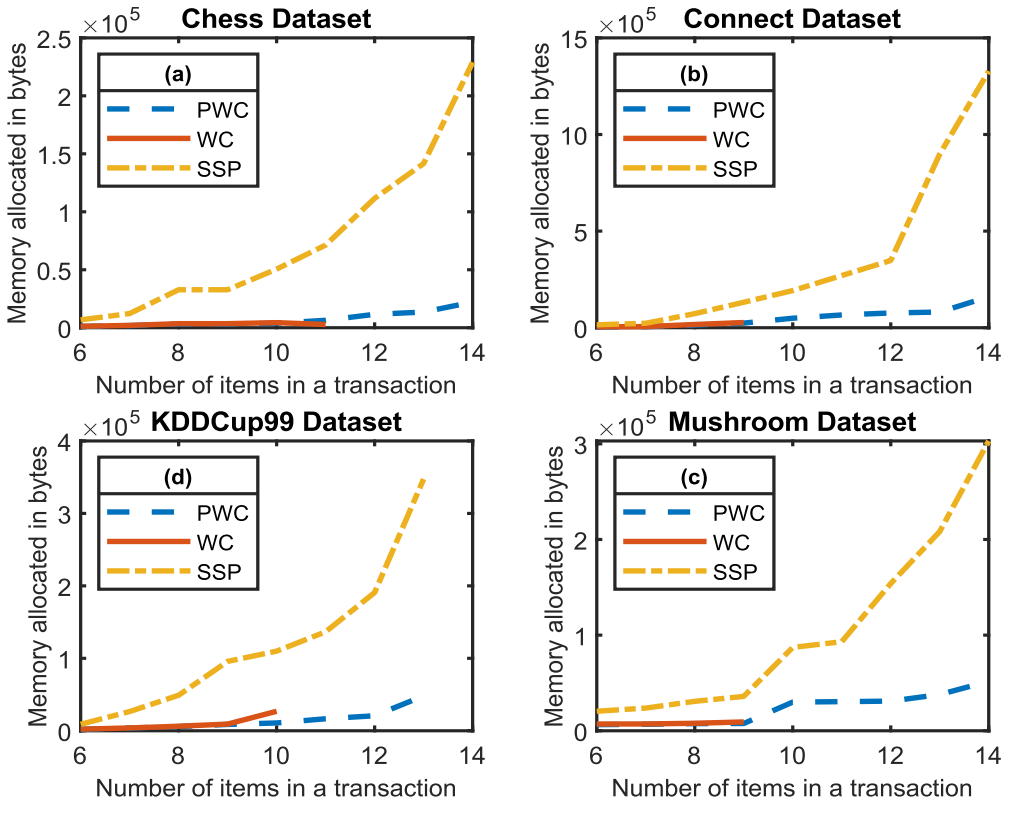

Fig. 6 Space utilization in main memory to store the dataset using SSP-Tree, WC-Tree, and PWC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, and (d) Mushroom.

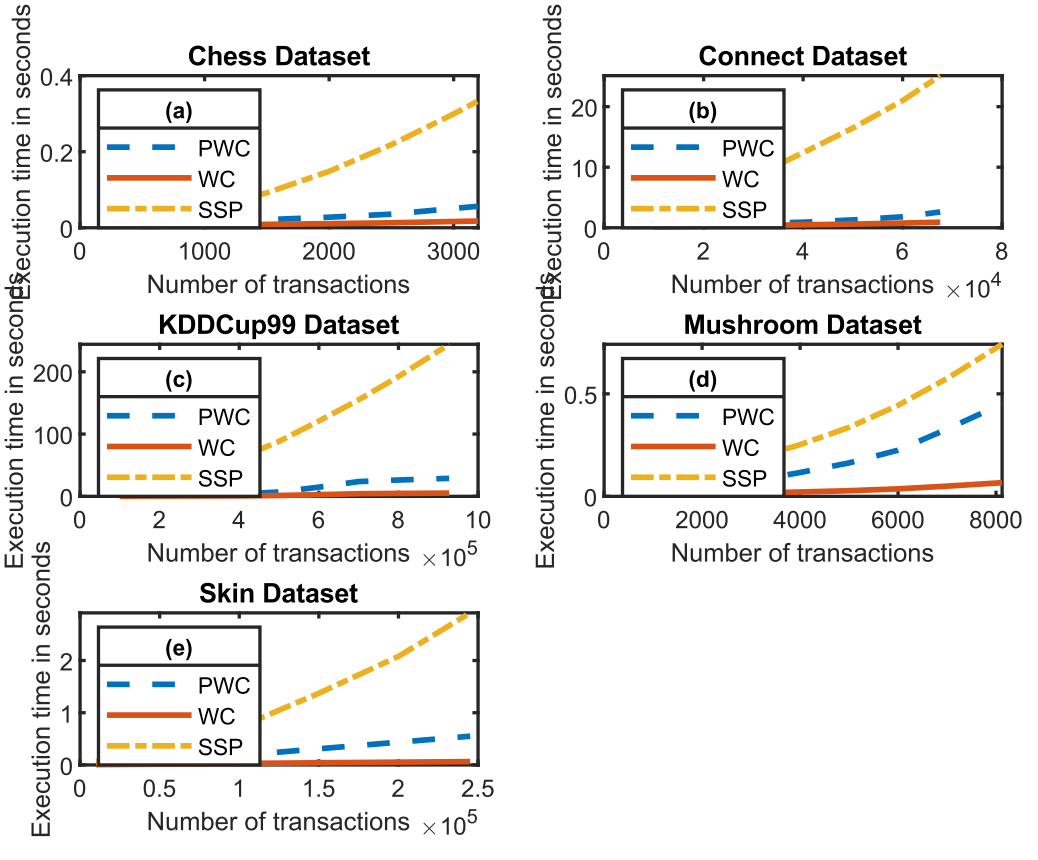

Fig. 7 Execution time to construct SSP-Tree, WC-Tree, and PWC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, (d) Mushroom, and (e) Skin.

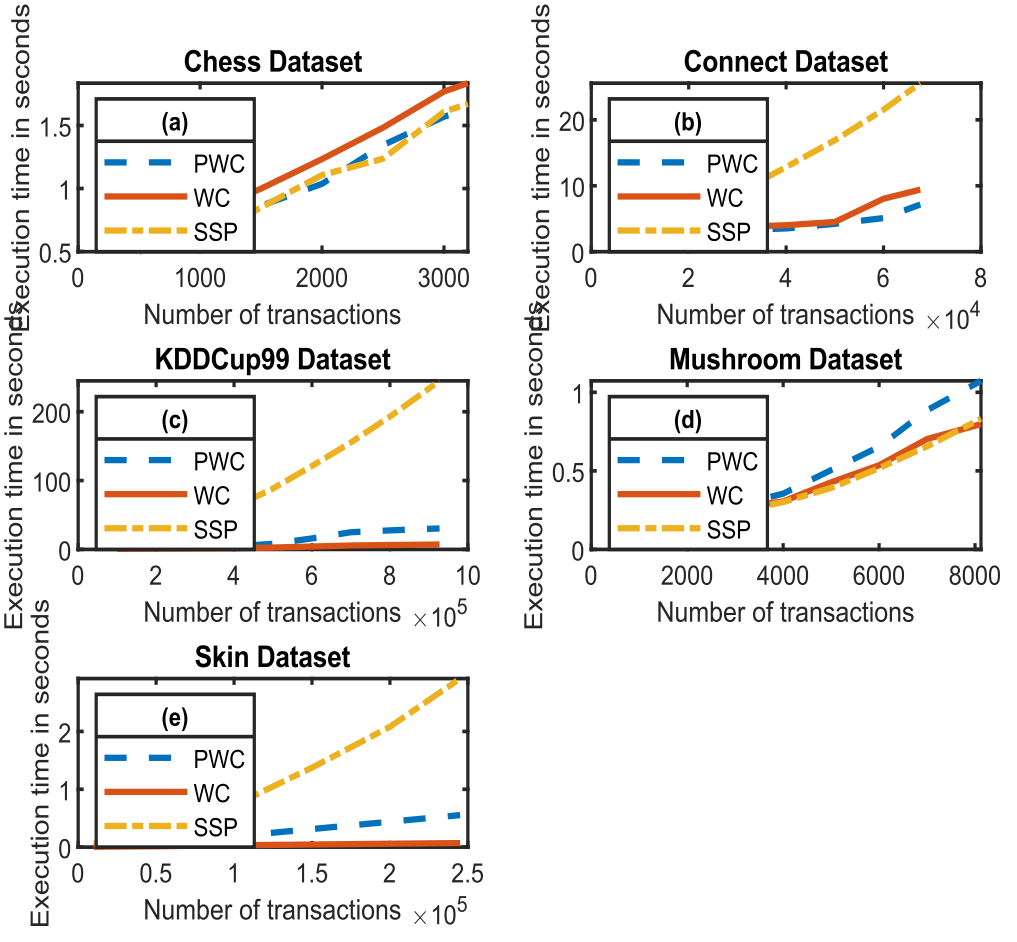

Fig. 8 Time needed to construct and mine itemsets from SSP-Tree, WC-Tree, and PWC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, (d) Mushroom, and (e) Skin.

Fig. 9 shows the memory consumed for the construction of WC-Tree, SSP-Tree, and PWC-Tree respectively in terms of bytes. PWC-Tree is found to be 10% more space-efficient than WC-Tree and 86% more space-efficient than SSP-Tree.

The results show that, for a dataset containing transactions with smaller weights, the WC-Tree algorithm would be preferable. SSP-Tree is more suitable if the dataset is small and the number of itemsets to be discovered is less. But if the dataset size and the number of itemsets to be discovered are large then, PWC-Tree is the best choice.

4. Discussion

PWC-Tree algorithm is a compact and complete representation of the entire database in the main memory. It is compared against the existing WC-Tree[28] and SSP-Tree[38] algorithms that store the whole database in the main memory in a single database scan. The algorithms are executed on different datasets mentioned in Table 1 and the experimental results are recorded based on varying dimensions and sizes of the datasets. Several interesting features related to time and space efficiency are observed and discussed in this section. The number of itemsets per transaction is varied during the tree construction and the result in Fig. 4 shows that WC-Tree is 79% more time-efficient than PWC-Tree during tree construction, provided the Weight of the transaction is a small number. If the Weight of the transaction is a large number, then it exceeds the capacity of the node in WC-Tree and the information is lost while retrieving the original information from the Weight. PWC-Tree uses REMNODE to store the partial weight of the transaction if the weight is too large, hence preserving the information. The construction of PWC-Tree outperforms SSP-Tree construction, and it is found to be 70% more time-efficient than the SSP-Tree algorithm. SSP-Tree is restructured during the insertion of every transaction into the tree based on the support of the itemset which is a time-consuming process. In PWC-Tree, the transactions are inserted without restructuring the tree making it time-efficient to store the dataset for further processing.

Rare itemsets are discovered from PWC-Tree, WC-Tree, and SSP-Tree by setting the min_{freq} threshold to 100% and min_{rare} to 0%. It is found that PWC-Tree is 30% more time-efficient than WC-Tree and 64% more time-efficient than SSP-Tree respectively. WC-Tree stores redundant Weights in the parent and child node whereas, PWC-Tree removes the common factor from the child node and stores it in the parent node. Hence during the discovery of the itemsets redundant subsets are not generated. PWC-Tree outperforms SSP-Tree because the maximum number of nodes to be visited during

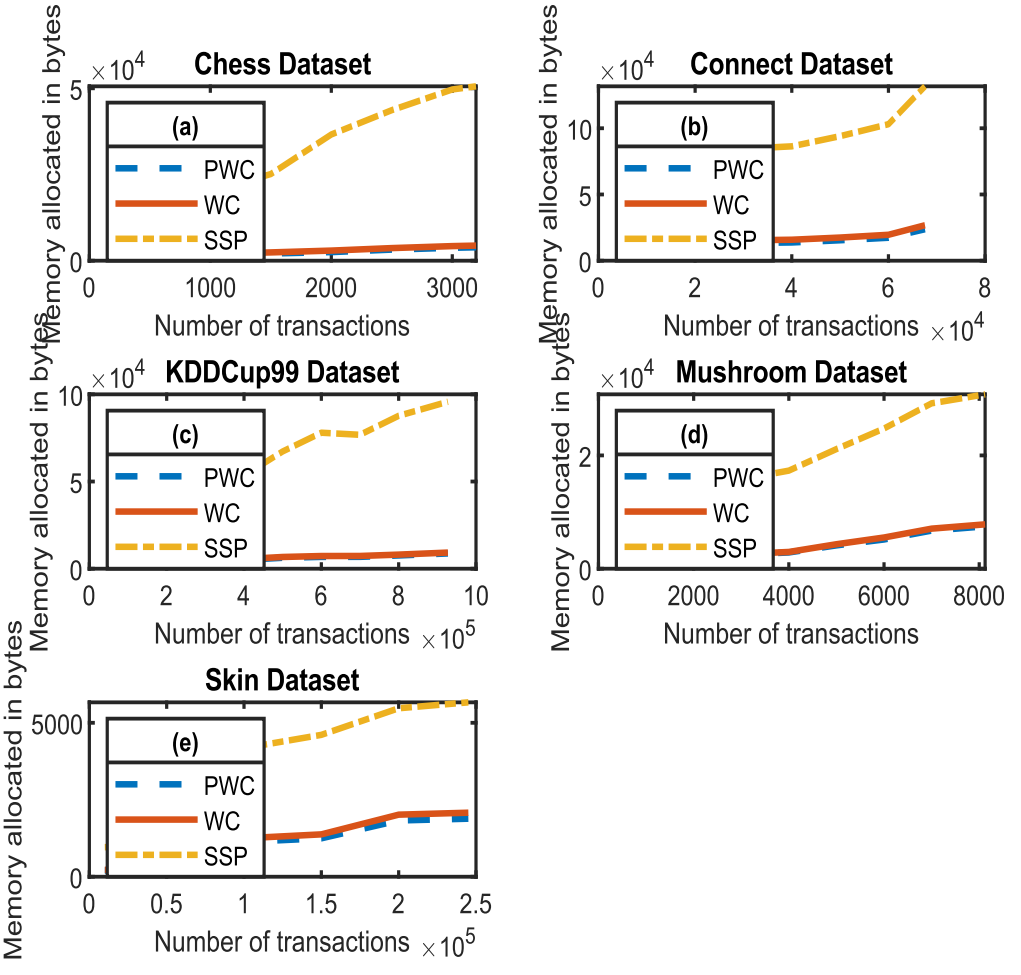

Fig. 9 Space utilized to store the entire dataset in main memory using SSP-Tree, WC-Tree, and PWC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, (d) Mushroom, and (e) Skin.

the discovery of itemsets in PWC-Tree is n, where n is the number of transactions in the dataset. Whereas in SSP-Tree, each item is stored as a node in the tree hence the traversal time is always greater than n. WC-Tree performs better than PWC-Tree and SSP-Tree if both construction and mining time are considered together. Therefore, WC-Tree can be a good choice only if the Weight of the transaction is within the range of the datatype used to store the Weight. When the dataset is large then PWC-Tree can be employed for efficient discovery of the itemsets without losing the original information.

PWC-Tree is found to be 9% more space-efficient than WC-Tree and 86% more space-efficient than SSP-Tree respectively. WC-Tree consumes a few extra bytes when compared to PWC-Tree because it stores redundant data in child nodes. SSP-Tree stores each item in the transaction as a node[38] whereas PWC-Tree stores the weight of the items in a node which reduces the construction of additional nodes significantly.

Additionally, the results attained by varying the number of transactions in the dataset are analyzed. The analysis revealed that WC- Tree is 79% more efficient when compared to PWC-Tree and SSP-Tree in terms of tree construction. PWC-Tree shows 88% better performance when compared with SSP-Tree during the tree construction. During the discovery of frequent and rare itemsets, it is found that PWC-Tree is 28% more time-efficient than WC-Tree. But SSP-Tree is 68% more efficient than the PWC tree because the itemsets in the nodes of the SSP-Tree are already sorted before insertion and hence all the nodes need not be visited if the itemsets' support does not satisfy the user-defined support threshold. Whereas in PWC-Tree all nodes must be visited to check the support of each itemset. PWC-Tree is found to be 10% more space-efficient than WC-Tree and 86% more space-efficient than SSP-Tree. PWC-Tree is more suitable for a large dataset with a low support threshold whereas SSP-Tree is suitable for a small dataset with a high threshold to discover the itemsets. A detailed comparison of percentage improvement for PWC against WC and SSP tree is given in Table 3.

4.1 Theoretical analysis

The PWC-Tree and WC-Tree are constructed for a transaction database with n transactions containing i unique itemsets and if the tree is skewed then, the time required to insert a node is in O (n) and to construct the tree is in O (n^2). Whereas SSP-Tree is in O (n × i) to insert a node and to construct the tree it is in O (n^2i). The memory allocated for PWC-Tree is ((n × y)

Table 3. Comparison of percentage improvement for the PWC, WC, and SSP Trees.

Dataset	Operation	Varying Data Dimension			Varying Data Size		
Chess	Tree Construction	WC (25%) <	PWC	< SSP (80%)	WC (63%) <	PWC	< SSP (82%)
	Construction + Mining	WC (8%) >	PWC	<SSP (34%)	WC (10%) >	PWC	<SSP (0.9%)
	Memory	WC (9%) >	PWC	<SSP (90%)	WC (13%) >	PWC	<SSP (92%)
Connect	Tree Construction	WC (65%) <	PWC	< SSP (89%)	WC (56%) <	PWC	< SSP (91%)
	Construction + Mining	WC (10%) >	PWC	<SSP (78%)	WC (20%) >	PWC	<SSP (70%)
	Memory	WC (13%) >	PWC	<SSP (80%)	WC (12%) >	PWC	<SSP (83%)
Mushroom	Tree Construction	WC (88%) <	PWC	< SSP (36%)	WC (83%) <	PWC	< SSP (47%)
	Construction + Mining	WC (40%) <	PWC	<SSP (16%)	WC (16%) <	PWC	<SSP (25%)
	Memory	WC (6%) >	PWC	<SSP (72%)	WC (4%) >	PWC	<SSP (80%)
KDDCup99	Tree Construction	WC (80%) <	PWC	< SSP (86%)	WC (81%) <	PWC	< SSP (88%)
	Construction + Mining	WC (76%) <	PWC	<SSP (85%)	WC (73%) <	PWC	<SSP (87%)
	Memory	WC (5%) >	PWC	<SSP (88%)	WC (6%) >	PWC	<SSP (90%)
Skin	Tree Construction	NA	NA	NA	WC (86%) <	PWC	< SSP (78%)
	Construction + Mining	NA	NA	NA	WC (85%) <	PWC	<SSP (78%)
	Memory	NA	NA	NA	WC (9%) >	PWC	<SSP (70%)

+ (m × z)) bytes, where n is the number of transactions, m is the total number of remaining nodes utilized, and y and z are the sizes of a node, and REMNODE in the tree respectively. The memory allocated for SSP-Tree is (n ×s ×i) bytes, where i is the number of items in the transaction and s is the size of a node in the tree. It can be noted that m is always less than i since each m contains the product of multiple items. Hence, PWC-Tree is more space-efficient than SSP-Tree. WC-Tree consumes (n × p) + (l × q) where l is the total number of Large nodes in the tree and p and q are the size of the node and Large node in the tree respectively. Since the common factors are stored in the parent node, the requirement of REMNODE is minimized in PWC-Tree. Also, the size of the Large node is greater than the size of REMNODE which reduces the amount of space utilized by PWC-Tree, and hence, PWC-Tree is more efficient than WC-Tree.

The application of frequent and rare itemset mining in ARM span various fields and it works as an intermediate step in the analysis of data for various problems. In the business sector, it is used to analyze the customer decision behavior while buying commodities. The analysis reveals the popularity of the commodities bought together which can be promoted to the customers. In the field of medicine, analysis can be made on the frequently or rarely occurring symptoms and their combinations to predict the disease at an early stage. Analysis of large databases can be expedited by storing the entire database in the main memory in PWC-Tree in a compact form hence facilitating efficient mining of rare and frequent itemsets.

5. Conclusion

The PWC-Tree algorithm proposed in this paper is designed to store the entire database in a complete and compact form in the main memory without information loss. The algorithm can be scaled up to any size and dimension without the loss of actual information. It can be implemented in the industries and sectors that demand real-time data analysis and requires algorithms that are both space-efficient and time-efficient without the cost of information loss.

The results obtained from the experiments conducted show that PWC-Tree performs better when a user-defined threshold is set low on large databases with high dimensions. Whereas WC-Tree loses actual information when executed on such databases and SSP-Tree requires more time to restructure itself before inserting the new transaction. Hence, it is proved that the PWC-Tree algorithm can be executed on large datasets

with high dimensions to prepare it for analysis and corresponding visual representation. However, if the dataset is dense and the number of items is huge then it will require multiple REMNODE nodes to represent the product of large prime numbers. SSP-Tree performs better than PWC-Tree in situations where the user-defined minimum frequent threshold is set high and the itemsets to be discovered are less in number. Nevertheless, from the experimental results it can be concluded that, for a large dataset, the PWC-Tree algorithm is both space and time-efficient algorithm when compared with WC-Tree and SSP-Tree algorithms.

In future work, a compact data structure can be developed to overcome the limitations of the Partial Weighted Count Tree that supports a large number of items in the database. An algorithm may be designed to mine the itemsets such that once the itemset is found to be frequent, the subset generation of the items from the remaining nodes that contain these itemsets is avoided.

Acknowledgments

The authors would like to acknowledge Mr. Nakul Shetty and Dr. Salmataj S. A. for their assistance while writing this research paper. The authors would like to thank the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal for providing the lab facilities to conduct the experiments.

Conflict of Interest

The authors declare no conflict of interest.

Supporting information

Applicable.

References

[1] R. Agrawal, S. Prabakaran, Big data in digital healthcare: lessons learnt and recommendations for general practice, *Heredity*, 2020, **124**, 525-534, doi: 10.1038/s41437-020-0303-2.

[2] N. Naik, Y. Rallapalli, M. Krishna, A. S. Vellara, D. K. Shetty, V. Patil, B. Z. Hameed, R. Paul, N. Prabhu, B. P. Rai, P. Chłosta, B. K. Somani, Demystifying the advancements of big data analytics in medical diagnosis: an overview, *Engineered Science*, 2022, **19**, 42-58, doi: 10.30919/es8d580.

[3] A. Namoun, A. Alshanqiti, Predicting student performance using data mining and learning analytics techniques: a systematic literature review, *Applied Sciences*, 2020, **11**, 237, doi: 10.3390/app11010237.

[4] Ghodousi, Alesheikh, Saeidian, Pradhan, Lee, Evaluating citizen satisfaction and prioritizing their needs based on citizens' complaint data, *Sustainability*, 2019, **11**, 4595, doi: 10.3390/su11174595.

[5] G. Suchacka, G. Chodak, Using association rules to assess purchase probability in online stores, *Information Systems and e-Business Management*, 2017, **15**, 751-780, doi: 10.1007/s10257-016-0329-4.

[6] H. Si, J. Zhou, Z. Chen, J. Wan, N. N. Xiong, W. Zhang, A. V. Vasilakos, Association rules mining among interests and applications for users on social networks, *IEEE Access*, 2019, **7**, 116014-116026, doi: 10.1109/access.2019.2925819.

[7] D. Beer, Envisioning the power of data analytics, *Information, Communication & Society*, 2018, **21**, 465-479, doi: 10.1080/1369118X.2017.1289232.

[8] C. Romero, S. Ventura, Educational data mining and learning analytics: An updated survey, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2020, 10, e1355, doi: 10.1002/widm.1355.

[9] B. Musunuri, S. Shetty, D. K. Shetty, M. K. Vanahalli, A. Pradhan, N. Naik, R. Paul, Acute-on-chronic liver failure mortality prediction using an artificial neural network, *Engineered Science*, 2021, **15**, 187-196, doi: 10.30919/es8d515.

[10] P. R. Rane, S. Vincent, Landslide susceptibility mapping using machine learning algorithms for nainital, India, *Engineered Science*, 2021, **17**, 142-155, doi: 10.30919/es8d600.

[11] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, *Proceedings of the 1993 ACM SIGMOD international conference on Management of data. May 25 - 28, 1993, Washington, D.C., USA. New York: ACM*, 1993, 207-216, doi: 10.1145/170035.170072.

[12] C. C. Aggarwal, Applications of frequent pattern mining. *Frequent Pattern Mining. Cham: Springer International Publishing*, 2014, 443-467, doi: 10.1007/978-3-319-07821-2_18.

[13] R. Agarwal, R. Srikant, Fast algorithms for mining association rules, *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, **1215**, 487–499.

[14] J. Sun, Y. Xun, J. Zhang, J. Li, Incremental frequent itemsets mining with FCFP tree, *IEEE Access*, 2019, **7**, 136511-136524, doi: 10.1109/access.2019.2943015.

[15] Y. Djenouri, D. Djenouri, J. C.-W. Lin, A. Belhadi, Frequent itemset mining in big data with effective single scan algorithms, *IEEE Access*, 2018, **6**, 68013-68026, doi: 10.1109/access.2018.2880275.

[16] L.-N. Sun, An improved apriori algorithm based on support weight matrix for data mining in transaction database, *Journal of Ambient Intelligence and Humanized Computing*, 2020, **11**, 495-501, doi: 10.1007/s12652-019-01222-4.

[17] A. Borah, B. Nath, Incremental rare pattern based approach for identifying outliers in medical data, *Applied Soft Computing*, 2019, **85**, 105824, doi: 10.1016/j.asoc.2019.105824.

[18] Y. Djenouri, D. Djenouri, J. C.-W. Lin, A. Belhadi, Single scan polynomial algorithms for frequent itemset mining in big databases, *2019 IEEE Congress on Evolutionary Computation (CEC). New York: ACM*, 2019, 1453-1460, doi: 10.1109/CEC.2019.8790127.

[19] J. S. Park, M.-S. Chen, P. S. Yu, An effective hash-based algorithm for mining association rules, *ACM SIGMOD Record*, 1995, **24**, 175-186, doi: 10.1145/568271.223813.

[20] S. Brin, R. Motwani, J. D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, *Proceedings of the 1997 ACM SIGMOD international conference on Management of data - SIGMOD '97. May 11-15, 1997. Tucson,*

Arizona, USA. New York: ACM Press, 1997, 255-264, doi: 10.1145/253260.253325.
[21] A. Savasere, E. Omiecinski, S. B. Navathe, An efficient algorithm for mining association rules in large databases, *Georgia Institute of Technology*, 1995, 432–444.
[22] D. Pan, S. Luo, Y. Feng, X. Zhang, F. Su, H. Liu, C. Liu, X. Mai, N. Naik, Z. Guo, Highly thermally conductive 3D BN/MWCNTs/C spatial network composites with improved electrically insulating and flame retardancy prepared by biological template assisted method, *Composites Part B: Engineering*, 2021, **222**, 109039, doi: 10.1016/j.compositesb.2021.109039.
[23] C. K.-S. Leung, M. A. F. Mateo, D. A. Brajczuk, A tree-based approach for frequent pattern mining from uncertain data, *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining. May 20 - 23, 2008, Osaka, Japan. New York: ACM*, 2008, 653-661, doi: 10.5555/1786574.1786639.
[24] M. Song, S. Rajasekaran, A transaction mapping algorithm for frequent itemsets mining, *IEEE Transactions on Knowledge and Data Engineering*, 2006, **18**, 472-481, doi: 10.1109/TKDE.2006.1599386.
[25] H. Najadat, A. Shatnawi, G. Obiedat, A new perfect hashing and pruning algorithm for mining association rule, *Communications of the IBIMA*, 2011, doi: 10.5171/2011.6512178.
[26] M. Zaki, S. O. Parthasarathy, M. Ogihara, W. li, New algorithms for fast discovery of association rules, *Proceedings of the 3rd Int'l Conf. on Knowledge Discovery and Data Mining (KDD97)*, 1997, **97**, 283–286.
[27] V. S. Ananthanarayana, D. K. Subramanian, M. N. Murty, Scalable, distributed and dynamic mining of association rules. *High Performance Computing — HiPC 2000. Berlin, Heidelberg: Springer Berlin Heidelberg*, 2000, 559-566, doi: 10.1007/3-540-44467-x_51.
[28] H. Huang, X. Wu, R. Relue, Association analysis with one scan of databases. *2002 IEEE International Conference on Data Mining, 2002. Proceedings. December 9-12, 2002, Maebashi City, Japan. IEEE*, 2003, 629-632, doi: 10.1109/ICDM.2002.1184015.
[29] M. Geetha, R. D'Souza, A dynamic approach for discovering maximal frequent itemsets, *2009 International Conference on Computer Engineering and Technology.* IEEE, 2009, **2**, 62–66, doi: 10.1109/ICCET.2009.153.
[30] L. Szathmary, A. Napoli, P. Valtchev, Towards rare itemset mining, *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Volume 01. New York: ACM*, 2007, 305-312, doi: 10.1109/ICTAI.2007.180.
[31] M. Adda, L. Wu, Y. Feng, Rare itemset mining, *Sixth International Conference on Machine Learning and Applications (ICMLA 2007). December 13-15, 2007, Cincinnati, OH, USA. IEEE*, 2008, 73-80, doi: 10.1109/ICMLA.2007.106.
[32] M. Adda, L.Wu, S. White, Y. Feng, Pattern detection with rare item-set mining, *arXiv preprint arXiv:1209.3089*, 2012, **1**, 1–17, doi: 10.48550/arXiv.1209.3089.
[33] S. Tsang, Y. S. Koh, G. Dobbie, RP-tree: rare pattern tree mining, *Data Warehousing and Knowledge Discovery. Berlin, Heidelberg: Springer Berlin Heidelberg*, 2011, 277-288, doi: 10.1007/978-3-642-23544-3_21.
[34] A. Borah, B. Nath, Mining rare patterns using hyper-linked data structure. *Lecture Notes in Computer Science. Cham: Springer International Publishing*, 2017, 467-472, doi: 10.1007/978-3-319-69900-4_59.
[35] A. Soltani, M.-R. Akbarzadeh-T, Confabulation-inspired association rule mining for rare and frequent itemsets, *IEEE Transactions on Neural Networks and Learning Systems*, 2014, **25**, 2053-2064, doi: 10.1109/tnnls.2014.2303137.
[36] M. Man, W. A. W. A. Bakar, M. M. A. Jalil, J. A. Jusoh, Postdiffset Algorithm in Rare Pattern: An Implementation via Benchmark Case Study, *International Journal of Electrical and Computer Engineering*, 2018, **8**, 4477-4485, doi: 10.11591/ijece.v8i6.
[37] M. J. Zaki, K. Gouda, Fast vertical mining using diffsets, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. August 24 - 27, 2003, Washington, D.C. New York: ACM*, 2003, 326-335, doi: 10.1145/956750.956788.
[38] A. Borah, B. Nath, Identifying risk factors for adverse diseases using dynamic rare association rule mining, *Expert Systems With Applications*, 2018, **113**, 233-263, doi: 10.1016/j.eswa.2018.07.010.
[39] M. Man, J. A. Jusoh, S. I. A. Saany, W. A. W. A. Bakar, M. H. Ibrahim, Analysis study on R-Eclat algorithm in infrequent itemsets mining, *International Journal of Electrical and Computer Engineering (IJECE)*, 2019, **9**, 5446, doi: 10.11591/ijece.v9i6.pp5446-5453.
[40] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, H. T. Lam, The SPMF open-source data mining library version 2, *Machine Learning and Knowledge Discovery in Databases. Cham: Springer International Publishing*, 2016: 36-40, doi: 10.1007/978-3-319-46131-1_8.

Author Information

Shwetha Rai *is an Assistant Professor in the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal India. She is currently pursuing her Ph.D. from Manipal Academy of Higher Education in the area of Data mining.*

Geetha M. *is a Professor in Dept. of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal India. She obtained her Ph.D. from NITK, Surathkal in 2010. Her current research includes Data Mining, Text Mining in Healthcare and Financial sectors. She has presented several papers in national and international conferences, and her work has been published in several international journals.*

Preetham Kumar *is Deputy Registrar-Academics(Technical) at Manipal Academy of Higher Education and Professor in the Department of Information & Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal. His research interests include Data Mining, Image processing, Bioinformatics, Advanced Database Management Systems, Operating Systems, Software Architecture, Software Engineering. He received his PhD in Data Mining from National Institute of Technology, Karnataka.*

Giridhar B. *was an undergraduate student in Department. of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal India.*

Publisher's Note: Engineered Science Publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.