Eng. Sci., 2022, **17**, 185–194

Engineered Science
DOI: https://dx.doi.org/10.30919/es8d602

Binary Count Tree: An Efficient and Compact Structure for Mining Rare and Frequent Itemsets

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

Abstract

The discovery of rare and frequent itemsets is done efficiently if the datasets to be processed are stored within the main memory. In recent years, various data structures have been developed to represent a large dataset in a compact form, which otherwise cannot be stored as a whole within the main memory. Binary Count Tree (BIN-Tree), a tree data structure proposed in this paper, represents the entire dataset in a compact and complete form without any information loss. Each transaction is encoded and stored as a node in the tree, in contrast to the existing algorithms that store each item as a node. The efficiency of the BIN-Tree for datasets of varying size and dimensions was evaluated against Single Scan Pattern Tree (SSP-Tree) and Weighted Count Tree (WC-Tree). The results obtained revealed BIN-Tree to be 95% and 75% more space-efficient than SSP-Tree and WC-Tree, respectively. The BIN-Tree construction and discovery of itemsets from a large dataset were found to be 93% and 22% more time-efficient than SSP-Tree and WC-Tree, respectively. BIN-Tree is equally efficient to discover rare and frequent itemsets from a small dataset in the main memory.

Keywords: Association Rule Mining; BIN-Tree; Frequent itemsets; Rare itemsets; Tree data structure.
Received: 30 October 2021; Revised: 15 November 2021; Accepted: 23 November 2021.
Article type: Research article.

1. Introduction

Rare and frequent itemsets mining is a data mining task for discovering associations among itemsets in the database and forms an integral part of the association rule mining (ARM) algorithm. Although inspired by supermarket-basket analysis, ARM has found applications in a variety of fields and has become a domain-independent technique.[1] It draws association rules among frequent and/or rare itemsets and aims to derive knowledge from different transactional databases.[2,3] The knowledge derived has contributed immensely to achieving excellence in respective fields of business, science, and technology. In later years, ARM was extended to find the association among the rare itemsets to uncover the previously ignored, but important relationships among the itemsets. An itemset is said to be frequent if it satisfies the user-defined support threshold,[1] otherwise, it is considered rare.[4] Most ARM algorithms are designed for small static datasets and require scanning the database multiple times to discover frequent and/or rare itemsets.[5,6]

In the current era, the volume of data generated has increased exponentially with the rigorous use of digital technologies. These data are acquired from various sources in the form of invoices, medical reports, social network platforms, and scientific observations among others.[7] The majority of educational, healthcare and industry sectors have been forced to carry out their activities online due to the pandemic and this shift from conventional mode has led to an enormous amount of data being generated.[8] Interesting information can be obtained by analyzing the patterns in these data. However, the volume and variety of data generated cannot be manually processed and multi-scan-based ARM algorithms require an enormous amount of time to derive the associations among various itemsets. To accelerate the computations, a single scan-based technique can be employed to discover rare and frequent itemsets from the dataset.

Data structures play a key role in the storage and discovery of interesting and meaningful information from the input dataset. Useful information can be discovered efficiently if the entire dataset is stored in the main memory and the number of database scans is reduced. Several data structures and

[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 (M. Geetha)

algorithms were presented to discover rare and frequent itemsets in a single scan. Perfect hashing and pruning algorithm (PHP), which is similar to direct hashing and pruning algorithm, stores the data in a hash table without collision. PHP utilized recursive hashing, a method similar to the lexicographically ordered tree to discover frequent itemsets.[9,10] Eclat algorithm represented the database in vertical format and added the transaction identifier to the item i if and only if i was present in that transaction. Several variations of this algorithm such as Mindset and Diffset have been implemented to discover the frequent itemsets.[11] A Pattern-tree was constructed by inserting the alphabetically sorted transaction items in a single scan. After the entire database was read, the list containing the items was sorted, and based on the updated list, each path in the tree was restructured.[12]

Ananthanarayana *et al*. implemented a Pattern Count tree algorithm to discover frequent itemsets from the general and complete tree in a single database scan. From this tree, a Lexicographically Ordered FP-Tree was constructed to discover frequent itemsets.[13] Geetha M. *et al*. designed and implemented a weighted count (WC) tree using the prime number concept to store all transaction items in the main memory to discover frequent concepts. Each transaction item was encoded using a unique prime number and its product was stored in the node of the tree. The memory space used to store the dataset in the main memory is reduced significantly as each node represents a transaction in contrast with other algorithms where each node represents an item. But there was information loss when the number of items per transaction was more.[14] The compact graph was implemented using a graph data structure to store the database efficiently in the main memory. Each node in the graph represented an item and the edge between the nodes represented the transactions using data encoding. It was reported to be more memory efficient than the Frequent Pattern Tree.[6] However, the tree construction and mining were inefficient due to the need for encoding/decoding of data in the graph.[15] Single Scan Frequent Itemset Mining algorithm implemented by Youcef Djenouri *et al*. discovered frequent itemsets in a single scan. The transactions were read from the database sequentially and the subsets of all the items in the transaction generated were stored in the hash table along with its corresponding support count.[16] Full Compression Frequent Pattern tree used a compressed tree data structure to store all itemsets from the database. Two head tables were created for storing frequent and infrequent itemsets, respectively in the sorted order with respect to their support count. The itemsets in the tree were restructured based on their position in the head table. All single paths in the FCFP tree were compressed to a single node by storing all the itemsets in that path in the string format.[17] An Improved Apriori algorithm scans the database and constructs a 2-dimensional matrix, called a 0-1 matrix. Each transaction was represented as a row and each column represented the items in the transaction. The presence of an item in the transaction database was represented as 1 and its absence as 0.[18] The Postdiffset algorithm discovered frequent itemsets similar to the Diffset algorithm for obtaining the result differently.[19,20] R-Eclat algorithm discovered rare itemsets from the database using the vertical representation of the transaction database. If the support(itemset) ≤ min_freq then in IF-Tidset, $(i \cap (i+1))$ was computed for i^{th} and $(i+1)^{th}$ columns, whereas in IF-Diffset, different $(i \cap (i+1))$ was computed for i^{th} and $(i+1)^{th}$ columns, and in IF-Sortdiffset, different $(i \cap (i+1))$ was computed for i^{th} and $(i+1)^{th}$ columns after sorting the itemsets in descending order depending on the largest to the smallest value in the equivalence class of the itemset.[21] A single scan pattern tree (SSP-Tree) was constructed using a single scan of the database. The tree was restructured based on the frequency of the itemsets stored in the header table before the insertion of the next transaction into the tree to maintain the order of the items in the tree. The SSP-Tree algorithm was also used to store the data and find outliers in the medical records using incremental mining.[22,23]

Several existing ARM algorithms discover either frequent or rare itemsets and the association among them. However, very few ARM algorithms discover both rare and frequent itemsets while storing the entire database in the main memory without losing the actual information. Some of the algorithms require restructuring of the tree or re-scanning of the database due to changes in the support threshold. Hence, a novel binary encoded data structure, a Binary count tree, to store the entire database in the main memory is proposed in this paper. A binary Count Tree is a complete and compact representation of the entire database to discover rare and frequent itemsets using a single database scan without loss of any information. It does not require re-scanning of the database when there is a change in the support threshold. The steps for binary count tree construction and discovery of rare and frequent itemsets from the tree are presented in section 2. The experimental results and their corresponding analysis are discussed in section 3. Section 4 concludes the paper and provides some input on future research directions.

2. Methodology

A binary count tree (BIN-Tree), a compact and complete data structure, is proposed in this work for storing the entire database in main memory in a single database. The data structure enables the discovery of both rare and frequent itemsets efficiently from the tree without loss of information even when the support threshold is changed.

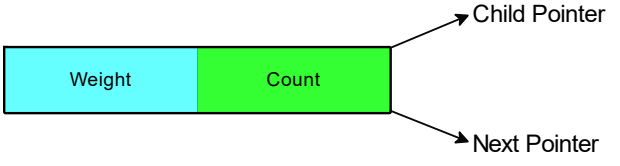

Fig. 1 Structure of a node with Weight, Count, Child, and Next pointer in BIN-Tree.

2.1 Structure of a node in Binary count tree

The structure of a node in BIN-Tree contains four fields as shown in Fig. 1. The Weight field in the node holds the information of a transaction in the database, the count field holds the support count of the transaction, the Child pointer holds the address of child node and next pointer holds the address of sibling node. Hence, each node in the tree represents a transaction in the data set.

2.2 Construction of Binary Count Tree

BIN-Tree is constructed in a single database scan where the transactions are read sequentially and encoded in binary format. The steps to construct a Binary count tree are as follows:

1. Read items in the transaction one at a time and indicate their presence or absence in the bitset using 1 and 0 respectively. The bitset forms the Weight of the transaction.

For example, if items are numbered 1, 3, 4, and 6 for a particular transaction then the corresponding bitset representation would be 1011010.

2. If the tree is empty then, create a node with the associated Weight, initialize the count to 1 and attach it 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 all transactions are inserted.
    (a) If the traversed node's Weight is equal to the transaction Weight, increase the count of the traversed node by a value of 1.
    (b) If the traversed node's Weight is a proper subset of the Weight of the transaction, then move to the child of the traversed node, if it is not NULL. Otherwise, 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 the created node as the child of the traversed node.
    (c) If the transaction's Weight is a proper subset of the Weight of the traversed node, then 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 a proper subset of the Weight of the transaction as the siblings of the created node. Lastly, remove the factors common to the parent and child from the child node.
    (d) If transaction Weight and Weight of the traversed node are a proper subset of each other, move to the right of the traversed node, if it is not NULL. If it is NULL, then create a node similar to step 2 and add it to the right of the traversed node.

2.3 Mining itemsets from Binary Count Tree

The rare and frequent itemsets from the BIN-Tree are discovered using the following steps:

1. Create a Hash Table with a prime number based on the dataset and create a secondary Hash Table for each entry in the primary hash table using a different prime number.
2. For each itemset in a node in the tree, generate the subsets of items marked as present. If the node is a child node, then perform a join operation with a subset generated from the Weight of the parent node.
3. Add the elements of each subset, map this sum using the above two primes and update it 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.
5. Otherwise, insert the value to the hash table initializing its count to the count of the traversed node.
6. Once all the nodes are traversed, iterate through the hash table, and display all the table entries whose count is between the two threshold values set by the user.

2.4 Illustration of the Binary Count Tree

BIN-Tree constructed for the sample database in Table 1 is shown in Fig. 2. The first transaction with items {1, 2} is read and a vector with sixty-four zeroes followed by 110 is created. The vector of zeroes and ones forms the Weight of the transaction. The first transaction is inserted as the first child of the root node and its support is initialized to 1. The Weight of the second transaction is calculated and inserted as a sibling node of the first transaction because the superset ($\succ$) relation (refer to definition 1.9 in supplementary material) does not apply to transaction 1 and transaction 2. Transaction 3 is also inserted similarly to transaction 2. Transaction 4 is inserted as the child of transaction 1 because of the relation Weight(T4) $\succ$ Weight(T1). The common factors {1, 2} are removed from the Weight of transaction 4 and the support of node containing transaction 1 is incremented by 1 before inserting transaction 4. The remaining nodes are inserted into the tree based on $\succ$ the relation between the Weight of transactions.

Table 1. Sample database with transactions' Weight.

TID	Transaction items	Binary Representation (Transaction Weight)
1	1,2	00...0110
2	3,4	00...011000
3	1,3	00...01010
4	1,2,3	00...01110
5	4	00...010000
6	1,2,3,65,66,67	1110...01110
7	1,2,3,57,58,59,60,61	00...0111110...01110
8	1,2,3,62,63,64	00...01110...01110

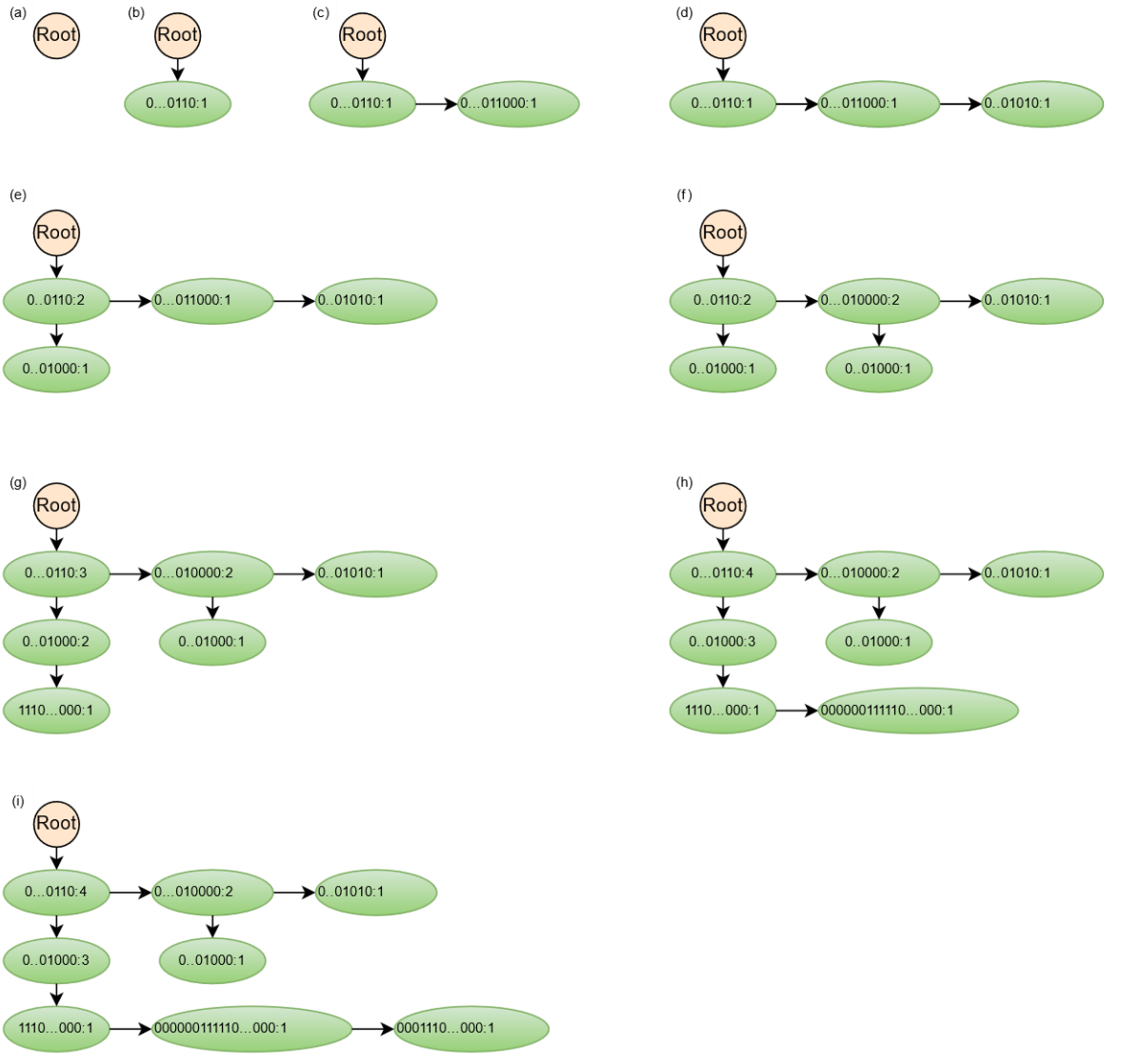

Fig. 2 Binary Count Tree for the sample database in Table 1. (a) Root Node (b) BIN-Tree after the first transaction, (c) BIN-Tree after the second transaction, (d) BIN-Tree after the third transaction, (e) BIN-Tree after the fourth transaction, (f) BIN-Tree after the fifth transaction, (g) BIN-Tree after the sixth transaction, (h) BIN-Tree after the seventh transaction, (i) BIN-Tree after the eighth transaction.

The rare and frequent itemsets generated for the sample database given in Table 1 with support thresholds min_{freq} = 50% and min_{rare} = 20% are given as a combination of *{itemset: Support_count}*.

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

3. Results and discussion

3.1 Datasets used

The algorithms SSP-Tree, BIN-Tree, and WC-Tree are executed on five secondary datasets, which include both dense and sparse datasets. The description of the datasets is given in Table 2.

These datasets are downloaded from the Sequential Pattern Mining Framework site and are most commonly used in the experimental analysis of rare and frequent itemset mining algorithms.[24] Out of these datasets, Chess, Connect, Skin and Mushroom are dense datasets, whereas KDDCup99 is a sparse dataset. In Table 2, the *#Items* column represents the total number of items in the dataset, *#Transaction* represents the total number of transactions in the dataset, and *Avg. length* is the average number of items in a transaction in the dataset under consideration.

Table 2. Description of the datasets.

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

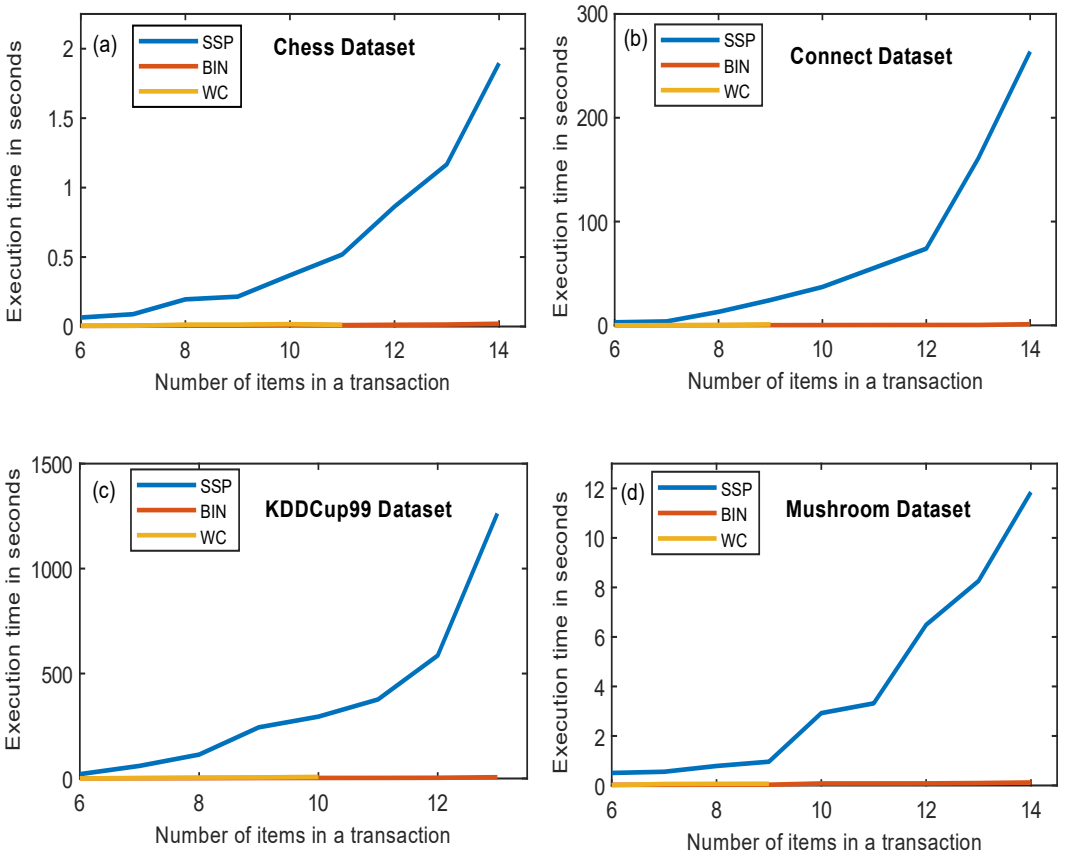

Fig. 3 Execution time for constructing SSP-Tree, BIN-Tree, and WC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, and (d) Mushroom.

3.2 Running environment

All algorithms used for comparison with the proposed algorithms are implemented in C/C++ and executed on an Ubuntu 18.04.5 LTS x64 operating system with 8GB RAM, Intel Core i5 at 3GHz processor.

3.3 Results

The performance of BIN-Tree is evaluated against SSP-Tree and WC-Tree, both of these store the entire database in the main memory using a single database scan. SSP-Tree was one of the recent algorithms published in the year 2018 and it was proved to be efficient when compared against other state-of-the-art algorithms in the same area. The WC-Tree algorithm is used in the evaluation because the BIN-Tree algorithm belongs to the same family of algorithms where each transaction is represented using an encoded form in the node. The experiments are conducted by varying the data dimension and varying the data size.

3.3.1 Results based on data dimension

Space and time efficiencies of WC, SSP, and BIN trees on Chess, Connect, KDDCup99, and Mushroom datasets are analyzed by conducting different experiments. The skin dataset is excluded from this experiment since it has only four attributes.

Figure 3 shows the execution time taken to construct WC, SSP, and BIN trees. It can be observed that BIN-Tree performs relatively better than WC and SSP during tree construction. SSP-Tree is restructured before the insertion of each transaction in every iteration. This makes the algorithm inefficient, and it can also be noted that execution time increases exponentially with the increase in transaction length. For a dataset with a smaller transaction length, WC-Tree execution time overlaps with BIN-Tree execution time. However, as the transaction length increases, it fails to store the valid encoded value in the tree leading to a loss of actual data during the decoding step. Thus, as the number of attributes increases indefinitely, the performance of the WC-Tree also degrades.

The time taken for tree construction and mining shown in Figure 4 reveals that BIN-Tree is more efficient than WC-Tree and SSP-Tree for large datasets such as Connect and KDDCup99 and for small datasets, such as Chess and Mushroom, it does not show much difference in time efficiency. Hence, for a small dataset, any of the three algorithms may be used to discover rare and frequent itemsets. Memory space utilized by SSP-Tree, BIN-Tree, and WC-Tree is shown in Fig. 5. It can be observed that BIN-Tree is the most efficient data structure among the three tree data structures considered for analysis. The reason for this is the technique followed for representing the data in BIN-Tree. The items in the transaction are represented as a bit vector which increases

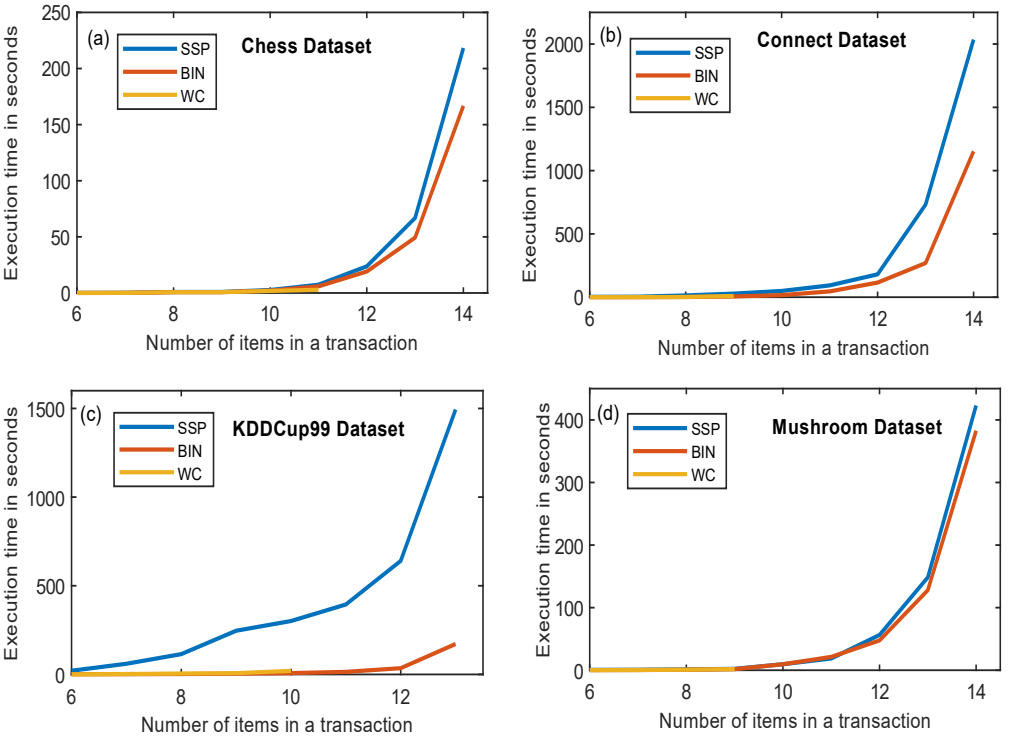

Fig. 4 Execution time for constructing and discovering itemsets from SSP-Tree, BIN-Tree, and WC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, and (d) Mushroom.

the node's capacity to store more information related to the transactions.

3.3.2 Results based on data size

An extensive study to analyze the space and time efficiencies of WC, SSP, and BIN trees is carried out based on the varying size of the dataset. These algorithms are executed on Chess, Connect, KDDCup99, Mushroom, and Skin datasets are shown in Fig. 6, which represents the execution time taken to construct SSP-Tree, BIN-Tree, and WC-Tree for various datasets. It can be observed that in most cases where the number of itemsets in the transaction is large, BIN-Tree outperforms WC and SSP trees. It is also revealed that in the Skin dataset, since the average length of the transaction is four, WC-Tree is found to be efficient for tree construction.

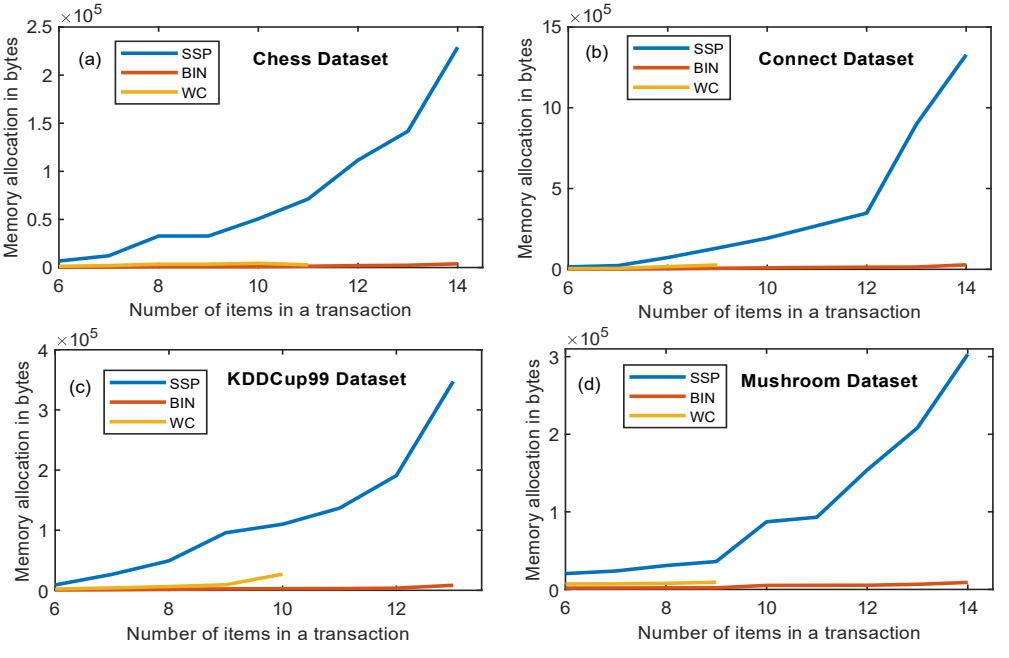

Fig. 5 Space utilized to store the entire dataset in main memory using SSP-Tree, BIN-Tree, and WC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, and (d) Mushroom.

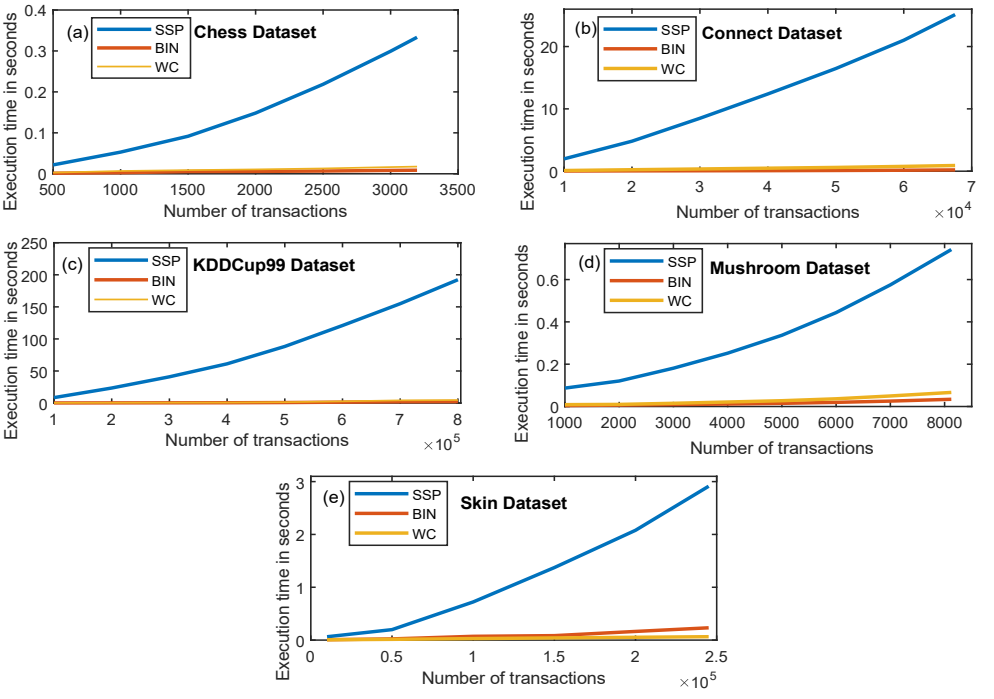

Fig. 6 Execution time taken to construct SSP-Tree, BIN-Tree, and WC-Tree for datasets(a) Chess, (b) Connect, (c) KDDCup99, (d) Mushroom, and (e) Skin.

Figure 7 shows the execution time for tree construction and the mining of required information from different trees. In large datasets like Connect, KDDCup99, and Skin data, BIN-Tree and WC-Tree show almost an equivalent efficiency to perform the operation and they are efficient when compared to SSP-Tree. If the dataset is small like Chess and Mushroom, then SSP-Tree performs better than WC-Tree and BIN-Tree. It can also be observed that the time taken for the construction and discovery of itemsets from the Mushroom dataset using SSP-Tree dips at the second value along the x-axis. This is caused since the data of interest to be discovered is towards the root of the tree in addition to the restructuring of nodes.

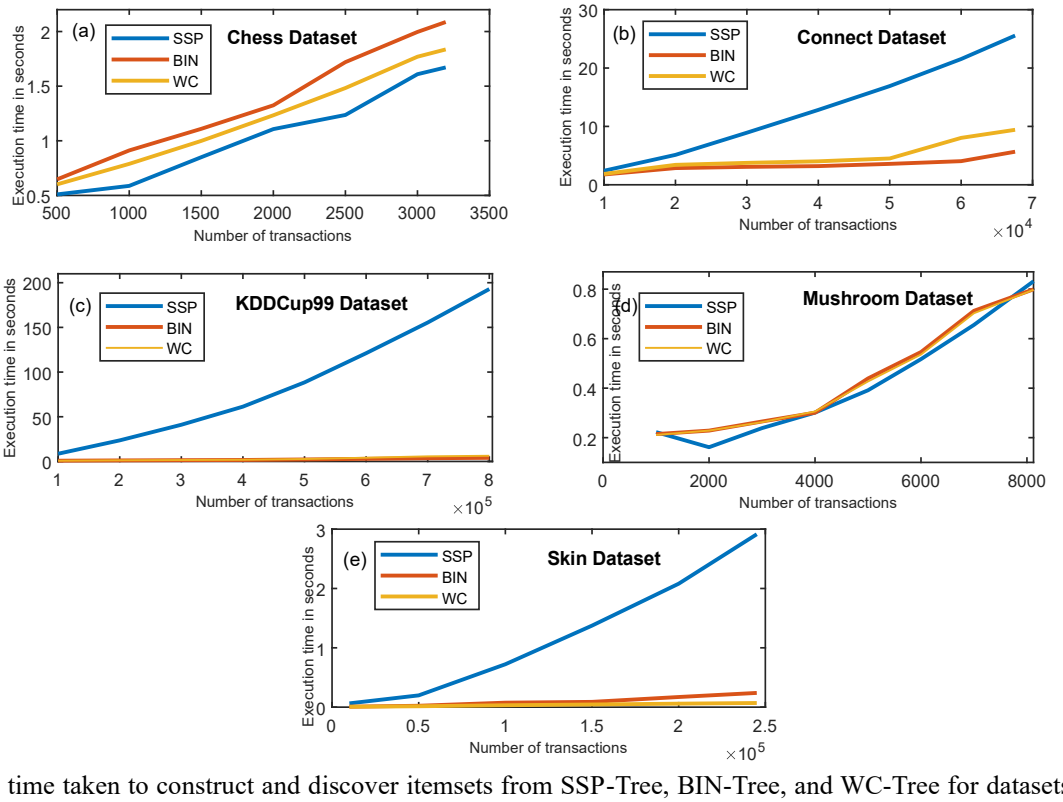

Fig. 7 Execution time taken to construct and discover itemsets from SSP-Tree, BIN-Tree, and WC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, (d) Mushroom, and (e) Skin.

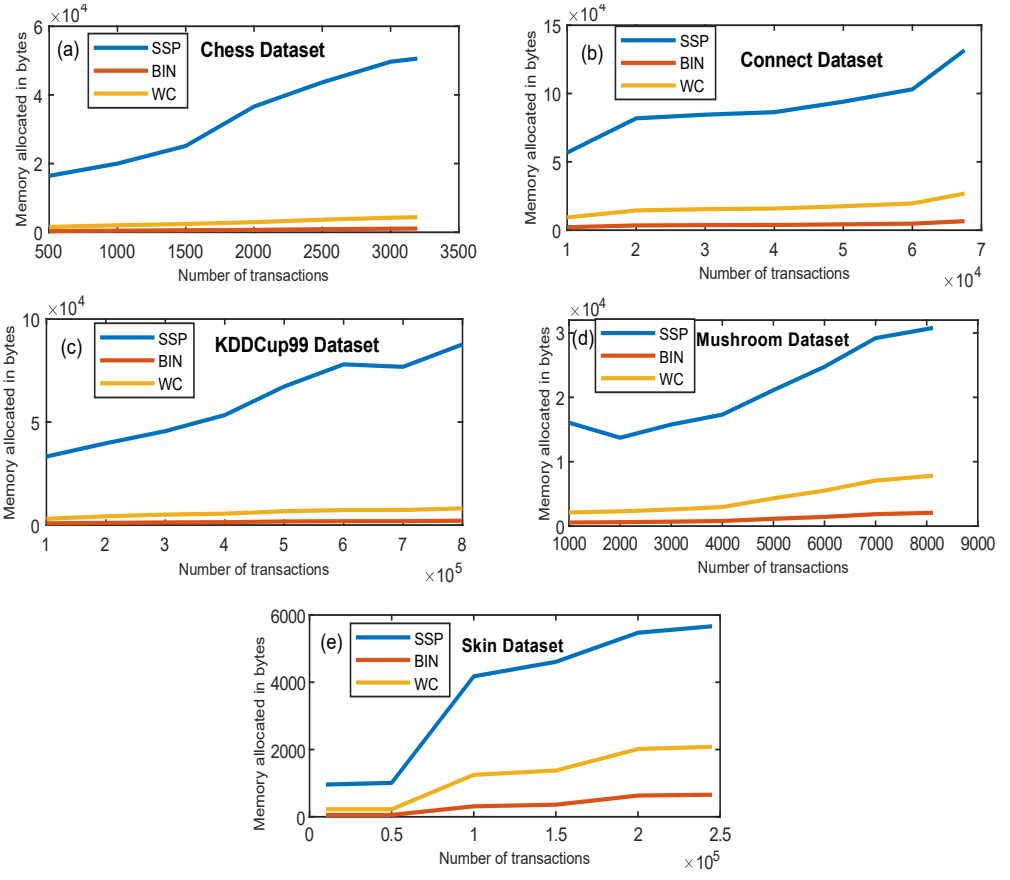

Fig. 8 Space utilized to store the entire dataset in main memory using SSP-Tree, BIN-Tree, and WC-Tree for datasets (a) Chess, (b) Connect, (c) KDDCup99, (d) Mushroom, and (e) Skin.

Figure 8 shows that BIN-Tree is space efficient when compared to WC and SSP trees. It can be verified that BIN-Tree is suitable for any type of dataset irrespective of its size.

3.4 Analysis and discussion

Space and time efficiencies are crucial for the discovery of rare and frequent itemsets and during this process, it is also necessary to preserve the information of the original database. Multiple scans of the database or restructuring of the data structure in the main memory are inefficient when the database is large. The proposed BIN-Tree algorithm addresses these limitations by representing the entire database as a compact structure in the main memory and discovers rare and frequent itemsets without information loss. The performance of the BIN-Tree algorithm to discover rare and frequent itemsets is evaluated and compared with WC-Tree[14] and SSP-Tree[22] algorithms by executing on different datasets. Several key points are observed from the experimentation carried out by varying the data dimension and data size.

The results obtained by varying the data dimension showed that the BIN-Tree algorithm is 98% more time efficient than SSP-Tree and 52% more time efficient than the WC-Tree algorithm to construct the trees in the main memory. SSP-Tree is restructured before the insertion of every transaction and WC-Tree requires the items to be replaced by their corresponding prime number. These additional operations are eliminated in BIN-Tree by storing the transaction in the form of a bitset. The item's presence in the transaction is indicated by one in the bitset as and when it is read from the database, hence making it efficient to store the entire database in the main memory.

The time for construction and mining from BIN-Tree is found to be 98% and 32% more efficient than SSP and WC trees, respectively. In SSP-Tree, mining rare and frequent itemsets are faster when the minimum frequent support threshold is set high since all the itemsets are sorted before inserting to the tree. The frequent itemsets are found near the root node and hence, the entire tree is not traversed. However, the time taken for SSP-Tree construction is considerably large due to which the total construction and mining time is greater than BIN-Tree. In WC-Tree, the itemsets in the transaction are retrieved by finding the factors of the Weight stored in each node. This additional step contributes to the cost of mining when compared to the method implemented in BIN-Tree. In addition to the mining cost, the information in the original database is lost while decoding the data stored in the WC-Tree when the Weight of the transaction is more than the size of the datatype used to store the Weight in the WC-Tree.

BIN-Tree is 95% space efficient than SSP-Tree and 75% more efficient than WC-Tree for the datasets taken into consideration. SSP-Tree stores each item in the transaction as a node and WC-Tree stores the Weight of the transaction,

which is a product of the prime numbers corresponding to each item in the transaction. Whereas BIN Tree represents each item as a bit which is the smallest unit of memory in the system. Each node stores the Weight of the transaction, which is a collection of these bits representing the presence or absence of items in that transaction thereby, reducing the amount of memory space utilized considerably.

A similar analysis of the results obtained by varying the number of transactions performed showed that BIN-Tree construction is 98% and 50% more time efficient than SSP and WC tree algorithms, respectively. The time for construction and mining process in BIN-Tree is 93% more efficient than SSP-Tree and 22% more efficient than WC-Tree for the datasets used in the study. It is also observed that the BIN tree is 96% and 75% more space efficient than SSP and WC trees respectively for the datasets under consideration.

3.5 Theoretical analysis

This section gives the theoretical analysis of the time and space efficiencies of BIN-Tree. The node insertion operation in BIN-Tree is in O(n) for a database with n transactions. Before inserting any node, it is compared with the nodes present in the tree. If all the (n-1) nodes present in the tree are unique then it requires (n-1) comparisons before inserting a node into the tree. Thus, it requires n comparisons to insert n such nodes, and hence, the tree construction operation is in $O(n^2)$. Also, the memory space utilized is n × y bytes, where n is the number of transactions, and y is the size of one node in the tree.

4. Conclusion and future work

The BIN-Tree algorithm presented in this paper stores the entire database in main memory and discovers rare and frequent itemsets efficiently without losing the actual information. The algorithm can be applied to any dataset in any sector like industry, education, and healthcare. BIN-Tree is the most space efficient among the algorithms of its family that stores the entire dataset in the main memory to discover rare and frequent itemsets. BIN-Tree overcomes the limitations of other algorithms such as restructuring of the paths before insertion of a new node, information loss due to the large value of Weight, and multiple-scan of the database due to changes in support threshold. It can be scaled up to any dimension and size of the data without compromising efficiency. Since the entire dataset is stored in the main memory, re-scanning of the dataset from the secondary memory is not required even if the support threshold is changed, unlike other existing algorithms which store partial datasets in the main memory.

As an enhancement to the existing work, the mining process can be accelerated to efficiently discover the required information from the dataset stored in the memory. BIN-Tree employs a time-consuming subset generation technique to find the support of all k-itemsets. This technique can be replaced with other methods for efficient discovery of rare and frequent itemsets.

Acknowledgments

The authors would like to acknowledge Dr. Salmataj S. A. and Mr. Nakul Shetty 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

There are no conflicts to declare.

Supporting information

Not Applicable.

References

[1] C. Sammut, G. I. Webb, Encyclopedia of Machine Learning, 2011, doi: 10.1007/978-0-387-30164-838.

[2] S. Raj, D. Ramesh, M. Sreenu, K. K. Sethi, *Knowledge and Information Systems*, 2020, **62**, 3565-3583, doi: 10.1007/s10115-020-01464-1.

[3] S. Darrab, D. Broneske, G. Saake, *International Journal of Machine Learning and Computing*, 2021, **11**, 208-218, doi: 10.18178/ijmlc.2021.11.3.1037.

[4] M. Adda, L. Wu, Y. Feng, 2007, *Cincinnati*, 73-80, doi: 10.1109/ICMLA.2007.106.

[5] R Agarwal, R Srikant, Proc. of the 20th VLDB Conference, 1994, 487-499, doi: 10.1.1.40.7506.

[6] J. Han, J. Pei, Y. Yin, *ACM SIGMOD Record*, 2000, 29, 1-12, doi: 10.1145/335191.335372.

[7] M. Letras, L. Bustio-Martínez, R. Cumplido, R. Hernández-León, C. Feregrino-Uribe, *Expert Systems with Applications*, 2020, **157**, 113440, doi: 10.1016/j.eswa.2020.113440.

[8] R. E. Reigal, J. L. Pastrana-Brincones, S. L. González-Ruiz, A. Hernández-Mendo, J. P. Morillo-Baro, V. Morales-Sánchez, Frontiers in Psychology, 2020, **11**, 588843, doi: 10.3389/fpsyg.2020.588843.

[9] H Najadat, A Shatnawi, G Obiedat, *Communications of the IBIMA*, 2011, 2011, 1–8, doi: 10.5171/2011.6512178.

[10] J. S. Park, M.-S. Chen, P. S. Yu, *ACM SIGMOD Record*, 1995, **24**, 175-186, doi: 10.1145/223784.223813.

[11] Z P Ogihara, M Zaki, S Parthasarathy, M Ogihara, W Li, *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, 1997, 283-286.

[12] H. Huang, X. Wu, R Relue, *2002 IEEE International Conference on Data Mining, 2002, Proceedings*, 2002, 629-632, doi: 10.1109/ICDM.2002.1184015.

[13] V Ananthanarayana, D Subramanian, M N Murty, *International Conference on High Performance Computing*, 2000, 559-566, doi: 10.1007/3-540-44467-X51.

[14] M Geetha, R D'Souza, *Social Informatics and Telecommunications Engineering 2012*, 2012, **539**, 367-370.

[15] N. Shahbazi, R. Soltani, J. Gryz, *Memory efficient frequent itemset mining*, 2018, 16-27, doi: 10.1007/978-3-319-96133-0_2.

[16] Y Djenouri, M Comuzzi, D Djenouri, *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2017, 644-654, doi: 10.1007/978-3-319-57529-250.
[17] J. Sun, Y. Xun, J. Zhang, J. Li, *IEEE Access*, 2019, **7**, 136511-136524, doi: 10.1109/access.2019.2943015.
[18] S Ln, *Journal of Ambient Intelligence and Humanized Computing*, 2020, **11**, 495–501, doi: 10.1007/s12652-019-01222-4.
[19] M Man, W A Wan Abu Bakar, M M A Jalil, J A Jusoh, *International Journal of Electrical and Computer Engineering*, 2018, **8**, 4477-4485, doi: 10.11591/ijece.v8i6.pp4477-4485.
[20] M J Zaki, K Gouda, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD'03*, 2003, 326-335, doi: 10.1145/956750.956788.
[21] M. Man, J. A. Jusoh, S. I. Ahmad Saany, W. A. W. Abu Bakar, M. H. Ibrahim, *International Journal of Electrical and Computer Engineering*, 2019, **9**, 5446, doi: 10.11591/ijece.v9i6.pp5446-5453.
[22] A. Borah, B. Nath, *Expert Systems With Applications*, 2018, **113**, 233-263, doi: 10.1016/j.eswa.2018.07.010.
[23] A. Borah, B. Nath, *Applied Soft Computing*, 2019, **85**, 105824, doi: 10.1016/j.asoc.2019.105824.
[24] P. Fournier-Viger, J. C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, H. T. Lam, *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.*

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

