IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 December 2023, accepted 1 January 2024, date of publication 8 January 2024,
date of current version 16 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3350737

RESEARCH ARTICLE

WBIN-Tree: A Single Scan Based Complete, Compact and Abstract Tree for Discovering Rare and Frequent Itemset Using Parallel Technique

SHWETHA RAI[1], PREETHAM KUMAR[2], K. NAKUL SHETTY[3], M. GEETHA[1], AND B. GIRIDHAR[1]

[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
[3]Department of Electronics and Communication Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

Corresponding author: M. Geetha (geetha.maiya@manipal.edu)

ABSTRACT Data analytics is an integral part of strategic decision making in various fields but not limited to business, education and healthcare systems. Existing research works focus on the discovery of itemsets with rare antecedents and consequent or frequent antecedents and consequent. Analysis of association among itemsets with rare antecedents and frequent consequent is equally important to gain valuable insights before making crucial decisions. Mining these itemsets from large datasets is time and resource intensive process. Expedition in the process of mining aids in quick decision making and hence, the entire dataset needs to be stored in the RAM. In this paper, a novel Weighted Binary Count Tree (WBIN-Tree) is proposed and implemented in CUDA to exploit the power of GPU and discover rules with rare antecedent and frequent consequent using parallel approach. WBIN-Tree stores the entire dataset in an abstract, complete and compact form in the RAM using single database scan. WBIN-Tree is compared with existing sequential and parallel algorithms by varying the data size and dimension. The performance evaluation of WBIN-Tree showed promising results, proving to be the most time and space efficient algorithm to store the entire large dataset in the RAM. However, based on the size of the GPU, the performance drops when executed on datasets with large dimensions which could be handled by processing the attributes in batches. Additionally, a case study is included to understand the importance of mining association rules with rare antecedent and frequent consequent by executing the algorithm on breast cancer dataset.

INDEX TERMS Association rule mining, frequent itemsets, parallel mining, rare itemsets, WBIN-tree.

I. INTRODUCTION

Data analytics has emerged as an important area of research due to increasing competitions in various fields of science, business and education. Several companies hire data analysts to analyse raw data and discover hidden information from large and complex datasets to improve the performance and deliver better results based on strategic decisions. Association Rule Mining (ARM) discovers meaningful relationships among the items that appear either frequently or rarely in a dataset [1]. If the occurrence of the itemset satisfies the user-defined minimum support threshold, then it is considered as frequent itemset otherwise, it is considered rare [2]. Frequent Itemset Mining (FIM) and Rare Itemset Mining (RIM) are first step in discovering strong association rules from large volumes of data where FIM involves identifying items that occur together frequently and RIM involves identifying items that occur infrequently in a dataset [3], [4]. While FIM is

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

useful for discovering information based on most frequent behaviors, RIM is useful for identifying unusual or outlier behaviors. For example, identifying outliers in medical data is important for early detection of diseases by identifying unusual patient behavior. The detection of outliers also ensures the quality of medical data [5]. Apriori [6], FP-Tree [7], PWC-Tree [8] and BIN-Tree [9] are some of the ARM algorithms to discover frequent and rare itemsets in the fields of medicine [10], business [11] and education [12].

Data analysis on useful itemsets from large datasets can be expedited if it is available in the main memory for immediate processing based on the user request. One of the challenges in accomplishing this feat is the number of database scans required by an algorithm to read the dataset from secondary memory and store it in the main memory. Multiple database scans are time-consuming and resource-intensive, especially on large datasets, and single scan algorithms have a large footprint in the main memory [6], [13]. Furthermore, existing single scan based mining algorithms were implemented based on either candidate generation or prefix tree based techniques. Both techniques require huge amount of memory space. Further, if the database is large it will not fit in the main memory. Additionally, mining the itemsets using sequential algorithms requires less resources and are efficient for small datasets. However, the mining becomes tedious when the sequential algorithm is applied on large datasets. Hence, the parallel mining approaches mentioned in [14] and [15] helped in fast discovery of interesting itemsets from the datasets.

ARM algorithms discover strong and interesting association rules to uncover associations among the rare or frequent itemsets based on antecedent and consequent rule concept. Furthermore, association among rare antecedent and frequent consequent is also of equal importance with regard to the medical dataset, providing a way to identify previously unnoticed instances. Medical datasets containing patient records with their diseases and symptoms may contain rare symptoms but frequent occurring disease. Analysing such cases is important because any negligence will lead to crucial situations. Support, Confidence and Lift are traditional evaluation measures to discover strong association rules among the itemsets which are then analyzed and conclusions are drawn according to the dependency of the itemsets [5], [16], [17], [18]. Null-invariant evaluation measures are used in ARM to assess the significance of discovered strong rules. These measures take into account the prevalence of the antecedent-consequent itemsets in the dataset and are therefore robust to imbalanced datasets [19].

The need for a space efficient and single scan based algorithm to store the entire dataset in the RAM inspired the design and implementation of novel Weighted Binary Count Tree (WBIN-Tree) to discover rare and frequent itemsets (RFI) from large datasets. Additionally, there is no research work pertaining to association of itemsets with rare antecedents and frequent consequent which is addressed in this paper. The remainder of the paper are as follows. Existing works related to RFI mining are summarized in Section II and Section III discusses the terminologies and definitions used in the paper. The methodology followed to implement WBIN-Tree is described in the Section IV. The performance evaluation based on the results and its discussion is given in Section V and Section VI respectively. A case study is discussed in Section VII and Section VIII gives the concluding remarks and research gap for future enhancements.

II. RELATED WORK

The traditional ARM algorithms involve multiple database scans to discover RFIs. These algorithms were able to store partial dataset in the RAM and are inefficient to discover itemsets from large dataset. Hence, several algorithms were proposed by various authors to handle large datasets using single database scan.

Frequent itemsets were discovered using a recursive hashing procedure in Perfect Hashing and Pruning algorithm [20]. Weighted Count tree (WC-Tree) proposed by Geetha M et al. discover frequent concepts. WC-Tree stored the entire dataset from the database in the RAM using a single scan. The transactions were encoded and stored as a single node in the tree. There was a significant reduction in the main memory space utilized by WC-tree algorithm [21]. Shahbazi et al. designed SPFP-Tree using single database scan to discover frequent patterns and handle incremental datasets. However, the prefix-paths of the tree were restructured according based on two hash tables which introduces computation and memory overhead [22]. A a hash based technique was utilised in SS-FIM algorithm to store the power set of all items in the transaction with its count [23]. Vijayakumar Kadappa and Shivaraju Nagesh implemented a Local Support-based Partition Algorithm (LSPA) to discover frequent itemsets from the partitioned database. Local support counts were computed and global support count for the itemsets were determined from the local support counts. The experimental evaluation showed that LSPA outperformed Apriori, Partition and FP-Tree algorithms in terms of database access time [24].

FCFP-Tree is a single scan based algorithm to store the itemsets and avoids database re-scanning. The tree was restructured based on the position of the itemsets in the head table. A string format was used to store all single paths as a compressed single node [25]. An Improved Apriori algorithm creates a 0-1 matrix where '1' indicates the presence and '0' indicates the absence of the item in the transaction [26]. Postdiffset algorithm is similar to Diffset algorithm [27] that discovers frequent itemsets. The initial looping in Postdiffset algorithm was based on tidsets process and the result diffset was obtained in the second looping [28].

Rare itemsets were discovered by Rare-Eclat algorithm using a vertical representation of the database. In each iteration if the $Sp(itemset) \leq min_{freq}$, then it does three things on i^{th} and $(i^{th}+1)$ columns. In IF-Tidset, it finds the $(i \cap (i+1))$ and saves the result in the database. In IF-Diffset,

it finds diffset($i \cap (i+1)$) and saves the result in the database. In IF-Sortdiffset, the itemsets were sorted in the descending order depending on the itemsets' equivalence class to find diffset($i \cap (i+1)$) [29].

ARIMA generates all frequent and rare candidate itemsets. The union of all rare itemsets collected at each level of candidate itemset generation gives a list of rare itemsets in [30]. SSP-Tree was constructed based on the frequency of the itemsets stored in the header table. The header table maintains the sorted order of itemsets and the tree was restructured before every insertion step accordingly [13]. The work in paper [5] also uses the SSP-Tree to store the data for incremental mining and to find the outliers in the medical records. Shafiul Alom Ahmed and Bhabesh Nath implemented an updated SSP-Tree algorithm named Improved Single Scan Pattern Tree (ISSP-Tree) [31]. The algorithm avoids multiple database scans for the tree construction and stores the each transaction as a branch in the tree. ISSP-Tree has a header table that contains a link to connect same items in the tree. ISSP-Tree introduces another link in the header table that connects the last node of same item. However, ISSP-Tree also restructures the tree before transaction insertion consuming more time. Mahmoud A. Mahdi et al. designed FR-Tree for rare pattern mining and to identify essential frequent and rare rules from the dataset [32]. The algorithm was designed based on F-Tree to cluster the transactional data [33]. The rules were categorized into two classes, frequent itemsets with high confidence and rare itemsets with high confidence.

An empirical analysis was conducted by Anindita Borah and Bhabesh Nath on a tree-based rare and frequent pattern mining techniques [34]. It was found that the algorithms based on tree data structures are more suitable for dense datasets as the transactions will share a common prefix path and hence could be represented in a compact form. When the tree data structure was executed on sparse data, as there was limited or no common prefix paths, the tree data structure would itself be large. This feature is the limitation on the main memory capacity to store the entire dataset in the RAM. PWC-Tree and BIN-Tree were implemented by Shwetha et al. to discover both frequent and rare itemsets from the main memory without any information loss [8], [9]. However, PWC-Tree is space inefficient for a dataset with large dimension and BIN-Tree is time-inefficient during mining process.

Parallel mining algorithms such as Bigminer [14], Gminer [15], SGminer [35] and SS-FIM [36] were proposed to improve the mining process to discover the frequent itemsets. Even though there are several single-scan based algorithms to discover rare and frequent itemsets, none of these store the dataset in a complete, compact and abstract form without re-structuring the tree based on frequency of the items before insertion of the transaction into the tree. Hence, this paper proposes a complete, compact and abstract tree data structure to store the entire dataset using a single database scan and without re-structuring the tree based on item-frequency.

TABLE 1. Sample database, SD, with five transactions.

T_{id}	Transaction items
1	1,2,3,55,56,57
2	1,2,3,47,48,49,50,51
3	4
4	1,2
5	1,2,3

III. PRELIMINARIES

In this section, some of the basic terminologies and definitions related to rare itemsets, frequent itemsets and evaluation measures to discover strong and interesting rules are illustrated with examples based on the sample database, SD, given in the Table 1. The database contains 5 transactions and 57 items. The first column represents the transaction ID (T_{id}) and the second column represents the items in each transaction. For further understanding, minimum frequent support threshold, min_{freq}, is set to 0.5 and minimum rare support threshold, min_{rare}, is set to 0.2.

Definition 3.1: **Tree data structure:** A non-linear data structure with single node called as root that has zero or more sub-trees containing child nodes is called a tree data structure.

Definition 3.2: **Completeness:** A tree is said to be complete if the entire dataset that is stored in the tree data structure is retrieved without any loss in information.

Definition 3.3: **Compactness:** A tree is said to be compact if the space taken by entire dataset in the main memory is less than its actual size.

Definition 3.4: **Abstraction:** A tree is said to be abstract if the information in the entire dataset that is stored in the encoded format in the tree data structure.

Definition 3.5: **Transaction database:** A group of s transactions, T_{db}={t_1, t_2,….,t_s} is called a transaction database where each t_i, i=1,2,….s represents a transaction.

Definition 3.6: **Itemset:** A group of x items, I = {i_1, i_2,….,i_x}, in T_{db} is called as an itemset. Here i_1, i_2,….,i_x represents individual items in the transaction.
In SD, I={1,2,3,4,47,48,49,50,51,52,53,54,55,56,57}

Definition 3.7: **l-Itemset:** A group of any l items from I is called as l-Itemset.

Definition 3.8: **Support:** The support of an itemset, $Sp(I)$, from I represents the number of times it appeared in the T_{db}. The support computation is shown in equation (1).

$$Sp(I) = \frac{|\{t \in T_{db} : I \subseteq t\}|}{|T_{db}|} \tag{1}$$

In Example 1, Sp(1)= $\frac{4}{5}$ =0.8, Sp(2,3)= $\frac{3}{5}$ = 0.6 and Sp(4)= $\frac{1}{5}$ =0.2

Definition 3.9: **Rare Itemset:** The occurrence of an itemset in T_{db} is between min_{rare} and min_{freq}.
In SD, itemset {4} is considered to be rare itemset.

Definition 3.10: **Frequent Itemset:** The occurrence of the itemset in $T_{db} \geq$ user defined minimum threshold min_{freq}.

In SD, itemset {1} and {1,2} are considered to be frequent itemsets.

Definition 3.11: **Association Rule:** It is an inference of the form X → Y, where Y⊂I, X⊂I and X∩Y= ϕ.

Definition 3.12: **Strong Association Rule:** An association rule is said to be strong if X and Y are frequent and it satisfies the confidence threshold value.

Definition 3.13: **Confidence:** Confidence of a rule (X → Y) indicates the probability of the presence of Y in dataset along with X in the dataset. The confidence value may not always show two dependent items. Two unrelated items in the database may also show a high confidence value [6]. Confidence is computed as shown in equation (2).

$$Conf(X \rightarrow Y) = \frac{Sp(X \cup Y)}{Sp(Y)} \tag{2}$$

Definition 3.14: **Lift:** Lift is an indicator to evaluate the strength of a rule X→ Y. If the lift of X→Y is 1 then it indicates that the existence of X does not affect Y, if the value is less than one then it infers that the probability of the existence of Y when X is present is less. If the value is greater than 1 then the inference is that the probability of the existence of Y is more when X is present in the transaction [37]. Lift is calculated as shown in equation (3).

$$Lift(X \rightarrow Y) = \frac{Conf(X \rightarrow Y)}{Sp(Y)} \tag{3}$$

Definition 3.15: **Conviction:** The output of Conviction ranges from 0 to ∞; 1 indicates that the itemsets are independent, ∞ indicates that the rule holds 100% of the time and less than one indicates that the rule is of lower interest. Conviction is calculated as shown in equation (4).

$$Conv(X \rightarrow Y) = \frac{1 - Sp(Y)}{1 - Conf(X \rightarrow Y)} \tag{4}$$

Definition 3.16: **Kulczynski:** The value of Kulczynski ranges from 0 to 1. The value close to 1 indicates a strong association between the itemsets, value close to 0 represents a weak association and 0.5 means neutral which is typically an uninteresting rule. Kulczynski is calculated as shown in equation (5).

$$Kulc(X \rightarrow Y) = \frac{1}{2}(Conf(X \rightarrow Y) + Conf(Y \rightarrow X)) \tag{5}$$

Definition 3.17: **Cosine:** The value of Cosine ranges from 0 to 1. If the value is close to 1, then it indicates a stronger association between the two itemsets, if the value is close to 0, then the strength of association is lesser and 0.5 means there is no correlation between the itemsets. Cosine is calculated as shown in equation (6).

$$Cos(X \rightarrow Y) = \frac{Sp(X \cup Y)}{\sqrt{Sp(X) * Sp(Y)}} \tag{6}$$

Definition 3.18: **Strong and Interesting Association Rule:** An association rule, $X \rightarrow Y$, is said to be strong and interesting if $Cos(X \rightarrow Y) > 0.5$ and $Conv(X \rightarrow Y) = \infty$.

Definition 3.19: **Parallelism:** Executing one instruction on multiple data at same time is called parallelism.

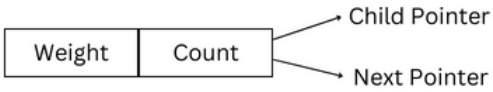

FIGURE 1. Structure of a node with Weight and Count.

IV. PROPOSED METHODOLOGY

The process of mining RFI from large databases can be accelerated if the entire dataset is stored in the RAM. Hence, an attempt is made to store the entire database in a compact, complete and abstract tree data structure in the RAM. WBIN-Tree, a tree data structure is proposed to store the entire dataset using a single scan of the database in the main memory. Rare and frequent itemsets are discovered from the data stored in the tree without any information loss using parallel processing. Further, strong and interesting association rules containing rare antecedents and frequent consequent are mined to uncover previously unknown information. The following subsections discuss the structure of the node in the proposed tree data structure, the tree construction process and a description of mining process to discover itemsets from the proposed tree data structure.

A. WEIGHTED BINARY COUNT TREE

Weighted Binary Count Tree (WBIN-Tree) is a compact, complete and abstract tree data structure used to store data in the primary memory. Each node in the tree represents a transaction from the dataset where each item in the transaction is represented using a bit. A collection of these bits represents the transaction and it is known as bitset. The presence of an item in the transaction is represented as '1' and absence as '0' in the bitset.

1) NODE STRUCTURE

A node in WBIN-Tree comprises Weight and Count fields. The 'Weight' field holds the bitset representation of the transaction and the Count field represents the occurrence of transaction in the database with same items. Hence each node in the tree represents a transaction with its occurrence in the database. The node structure is shown in Figure 1.

2) WBIN-TREE CONSTRUCTION

The proposed WBIN-Tree is constructed using a single database scan. Each transaction is read one by one and the Weight of the transaction is computed using bitset. If the tree is empty, then a node is created with the transaction Weight and its Count is initialised to 1. This node is inserted as the first child of the root node. If the tree not empty, then the node is inserted based on the following conditions: it is traversed based o is not empty then the t

The steps to construct WBIN-Tree is as follows:

1) Weight(traversed_node) is equal to new node: the count of traversed_node is increased by 1.
2) Weight(traversed_node) ⊂ Weight(current transaction):

a) Traversed_node has children: move to the child of the traversed_node.
b) Traversed_node does not have children: create a new node and insert the new node as the child of the traversed_node.

3) Weight(current transaction) $\subset$ Weight(traversed_node): create a new node and insert it as the parent of the traversed_node. Move all the siblings of the traversed_node as the siblings of the new node whose Weights are not proper subset of Weight(current transaction).
4) Weight(current transaction) and Weight(traversed node) are not proper subset of each other:
   a) If the traversed_node has sibling node to the right, then move to the right of the traversed_node.
   b) If the right of traversed_node is NULL, then create a new node and insert it its right.

Algorithm 1 describes the method to construct the WBIN-Tree. Line 2 in the algorithm represents the transaction in bitset format which is the Weight of the transaction. Lines 3 and 4 create a new node if the tree is empty and lines 5 to 30 describe how the node insertion step is carried out in WBIN-Tree based on different conditions given in Section IV-A2.

3) WBIN-TREE: MINING RARE AND FREQUENT ITEMSETS

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

1) Create a Hash Map, HM.
2) For each traversed node of WBIN-Tree, generate the subset of itemsets marked as present using parallel approach.
3) Insert the subset into HM if it is not present in the HM and initialise the count with the count of the traversed node.
4) Otherwise, add the count in the HM with traversed node and update the count in the HM with the result.
5) Iterate through HM at the end and display all the subsets whose count is between rare and frequent support threshold values.

Algorithm 2 represents a brief procedure regarding mining process designed to discover RFI from the WBIN-Tree. Algorithm 3 represents the procedure to identify the itemsets present in the bitset representation. The parallel subset generation method is described in Algorithm 4.

B. ILLUSTRATION OF THE WBIN-TREE

A sample database with the Weight of each transaction is given in Table 2 and corresponding WBIN-Tree constructed for the sample database. A detailed illustration of the steps followed to construct the tree is shown in the Figure 2.

To begin with, WBIN-Tree is empty and the first transaction, T1, is read from the database. The Weight of T1 is calculated as 00110, Count is set to 1 and inserted as the child of the WBIN-Tree root node. Transaction T2 is inserted as the sibling of T1 as the Weight of T2 is not a subset or a super set of Weight of T1. Transaction T3 is also inserted based on the same condition into WBIN-Tree. The next transaction, T4, is read and its Weight is compared with the Weight of T1. Weight T1 is a subset of Weight of T4 and hence, T4 is inserted as the child of T1. The Weight of transaction T5, is a

Algorithm 1 Weighted_Binary_Count_Tree()

Input: T_{db} a transactional database
Output: Weighted Binary Count Tree: a complete, compact and abstract tree data structure

```
1:  for each transaction T_k ∈ T_db do
2:    Weight_k ← bitset representation T_k
3:    if the WBIN-Tree is empty then
4:      N_new ← createNode(Weight_k)
5:    else
6:      complete ← 0
7:      traverse_node ← root.child
8:      while complete ≠ 1 do
9:        if Weight_k= Weight(traverse_node) then
10:         complete ← 1
11:         count(traverse_node) ← count(traverse_node) + 1
12:       else if Weight_k ∧ Weight(traverse_node)=Weight(traverse_node) then
13:         if traverse_node has no child then
14:           create a node N_new ← createNode(Weight_k)
15:           traverse_node.child ← N_new
16:           complete ← 1
17:         else
18:           insert traverse_node as child of traverse_node
19:         end if
20:       else if Weight_k ∧ Weight(traverse_node)=Weight_k then
21:         Node_new ← createNode(Weight_k)
22:         traverse_node is replaced by Node_new
23:         insert traverse_node as the child of Node_new
24:         complete ← 1
25:       else
26:         insert traverse_node to the right of traverse_node
27:       end if
28:     end while
29:   end if
30: end for each
```

TABLE 2. Sample database with transactions' weight using Bitset.

T_{ID}	Items	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

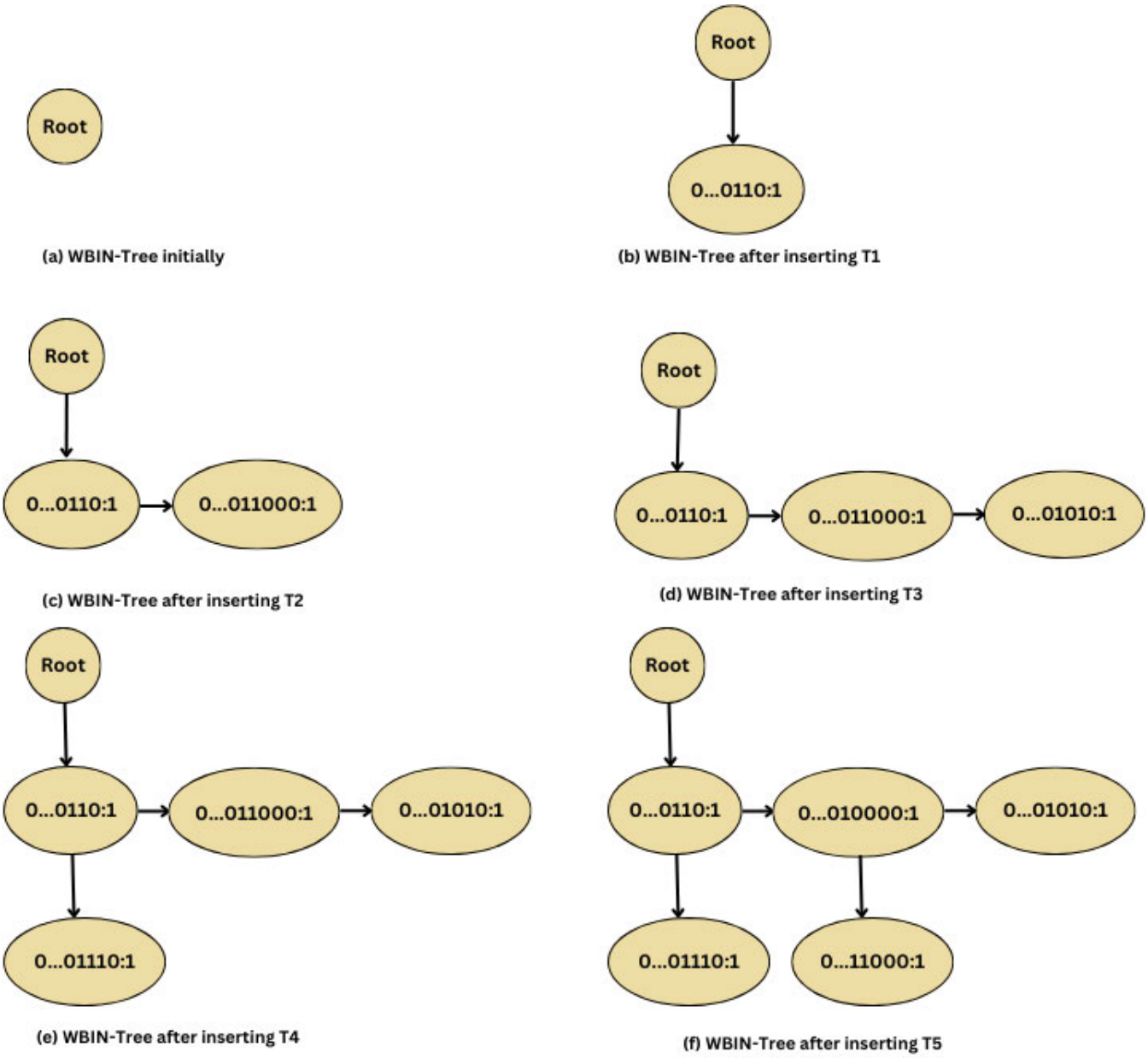

FIGURE 2. WBIN-Tree for the sample database in table 2.

subset of Weight of transaction T2. Hence, T2 is inserted as the child of T5 and T5 as the sibling of T1.

C. THEORETICAL ANALYSIS

WBIN-Tree algorithm is constructed for a T_{db} with t transactions containing i unique itemsets. Hence, the entire transactions are read in (t × i) time from the database. The insertion operation in WBIN-Tree is in O(t) for a T_{db} with t transactions. During insertion, if there are (t-1) nodes in the tree, then (t-1) comparisons are made before node insertion and (t × i) steps are taken to read each transaction from the secondary database. Thus, the tree construction operation is in O((t × i) + t^2)). Further, WBIN-Tree consumes (t×x) bytes to store the dataset where x is the size of one node in the tree.

V. PERFORMANCE EVALUATION

The performance evaluation of the proposed tree data structure is assessed based on time taken to construct the tree and the space taken to store the entire dataset in the RAM by varying the data size and dimension. The experiments are also extended to evaluate the time taken to discover RAM and rare itemsets using sequential and parallel approaches. Several experiments are conducted by executing WBIN-Tree algorithm on secondary datasets described in the Section V-A to analyse the algorithms' significance against existing sequential and algorithms. The experiments

Algorithm 2 Weighted_Binary_Count_Tree_Mining()

Input: WBIN_Tree, min_{rare}, min_{freq}
Output: Frequent and Rare itemsets

```
Create a Hash Map HT
for each Bitset B_T ∈ WBIN_Tree node N do
    arr ← findSetBit(B_T)
    subset_List ← Generate_Subset(arr)
    for each subset_element e ∈ subset_List do
        index ← HashKey(e)
        if e is not in HT then
            HT[index].element ← e
            HT[index].count ← N.qty
        else
            HT[index].count ← HT[index].count + N.qty
        end if
    end for each
end for each
for each HT[index].element in HT do
    if HT[index].count≥ min_freq then
        freq_list ← freq_list ∪ HT[index].element
    else if HT[index].count ≥ min_freq and
    HT[index].count < min_rare then
        rare_list ← rare_list ∪ HT[index].element
    end if
end for each
```

Algorithm 3 findSetBit()

Input: Bitset B_T
Output: An array *arr* containing the elements present in B_T

```
k ← 0
#pragma omp parallel shared(arr) do
    #pragma omp for
        for each bit i ∈ B_T do
            if B_T(i) is set then
                arr[k] ← i
            end if
            k ←k+1
        end for each
    end #pragma omp for
end #pragma omp parallel
```

Algorithm 4 Generate_Subset()

Input: List of node elements in array *arr*
Output: List of subsets *subset_list*

```
n ← number of elements in arr
No_of_Threads Th ← 2^n
par for i ← 0 to Th do
    for j ← 0 to n do
        if i & 1 ≪ j then
            subset_list[Th_i] ← subset_list[Th_i] ∪ arr[j]
        end if
    end for
end par for
```

are conducted in two ways to evaluate the time and space efficiency of the algorithms: by varying the data dimension and data size.

A. DATASET DESCRIPTION

The algorithms are executed on both sparse and dense datasets. The description of seven secondary datasets used in the experimentation is given in Table 3. The datasets are downloaded from SPMF site commonly used in experimental analysis of RFI mining algorithms [38]. The Acute Inflammation dataset is collected from UCI Repository [39] and

TABLE 3. Description of the datasets.

Dataset	#Transaction	#Items	Avg. length	Type
Acute Inflammation	120	11	8	Real
Breast Cancer	317	16	13	Real
Chess	3196	75	37	Real
Mushroom	8124	119	23	Real
Connect	67557	129	43	Real
Skin	245057	11	4	Real
KDDcup99	927393	135	16	Real

Breast Cancer dataset is collected from Kaggle website [40]. Chess, Acute Inflammation, Skin, Connect, Breast Cancer and Mushroom are dense datasets and KDDCup99 is a sparse dataset. In Table 3, *#Transaction* indicates the total number of transactions, *#Items* indicates the total number of items and *Avg.length* gives the average number of items in a transaction in the dataset.

B. EXPERIMENTAL SETUP

The implementation of all sequential algorithms are done using C++ and executed on a Ubuntu 18.04.5 LTS x64 OS with 8GB RAM, Intel Core i5 at 3GHz processor. The proposed parallel algorithm is implemented in CUDA and all parallel algorithms are executed on Google Colab.

C. EXPERIMENTAL EVALUATION: TREE CONSTRUCTION

The efficacy of the proposed WBIN-Tree in terms of space and time to store the entire dataset in the RAM is assessed in this section. The significance of WBIN-Tree algorithm against existing WC-Tree, PWC-Tree, BIN-Tree and SSP-Tree algorithms are analysed by running the experiment on various secondary datasets. The experiments are conducted in two ways to evaluate the time and space efficiency of the algorithms: varying data dimension and data size.

1) VARYING DATA DIMENSION

A comparative evaluation on time and space taken by WC-Tree [21], PWC-Tree [8], SSP-Tree [41], BIN-Tree [9] and WBIN-Tree algorithms to construct the trees and store the entire dataset in the RAM is done by conducting experiments based on varying data dimension of the datasets. The experiments are carried out to observe the performance of

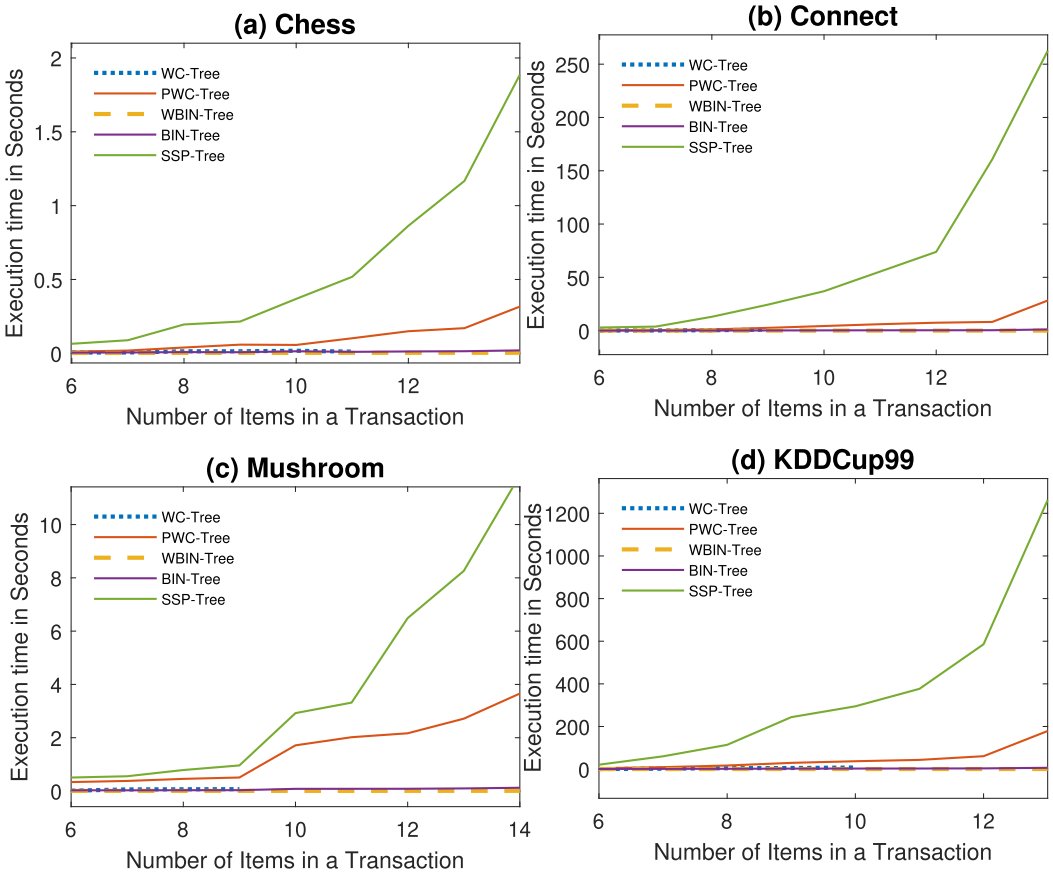

FIGURE 3. Figures 3(a)- 3(d): Time taken to construct WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree by varying data dimension.

the algorithms due to the number of items present in the dataset. The proposed and existing algorithms are executed on Mushroom, KDDCup99, Connect and Chess datasets. Since Skin dataset has only four data attributes per row, it is excluded from this experimentation.

The execution time of PWC, WC, BIN, SSP and WBIN trees to store various datasets in RAM is shown in Figure 3. It is observed from the figure that during tree construction step, WBIN-Tree outperforms BIN, WC, PWC and SSP tree algorithms. There is an exponential increase in the time to construct SSP-Tree as the number of items increase in the dataset. SSP-Tree is restructured before the insertion of every transaction which adds to the execution time which makes it time-inefficient. Further, the execution time of PWC-Tree increases gradually with the increase in transaction length. PWC-Tree requires extra time to create additional REMNODES to store larger Weights, hence takes more time than WBIN-Tree to construct the tree. For datasets with smaller transaction length, WBIN-Tree and WC-Tree shows similar performance during tree construction. However, WC-Tree fails to store a valid encoded value with the increase in transaction length leading to a loss of actual data. Thus, the performance of the WC-Tree drops as the number of attributes increases indefinitely. BIN-Tree algorithm uses a tree construction approach similar to PWC-Tree; during tree construction, in BIN-Tree, the common elements among the parent and child node are removed from child node before node insertion. This additional operation adds up to the cost of tree construction. Hence, BIN-Tree is inefficient compared to WBIN-Tree during tree construction.

Memory utilised by WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree is shown in Figure 4. There is an exponential increase in the memory consumed by SSP-Tree and PWC-Tree with the increase in number of items in the transaction. It can also be observed that the memory utilised by WC-Tree, BIN-Tree and WBIN-Tree are overlapping but, for a large KDDCup99 dataset there is a slight increase in the memory utilised by WC-Tree. Hence, BIN-Tree and WBIN-Tree are the most efficient data structures among the tree data structures designed and implemented to store the entire dataset in the RAM. SSP-Tree stores each item as a node of the tree, whereas WC-Tree and PWC-Tree store the Weight of the transaction, an integer, as a node of the tree in the RAM. BIN and WBIN trees represent each item as a bit and each transaction as a bit vector, the most basic unit of memory representation. The bit representation of each item is the most compact representation of the item compared to integer representation, Consequently, there is an increase in the node's capacity and tree's capacity to store more items and transactions respectively. Hence, BIN-Tree and WBIN-Tree

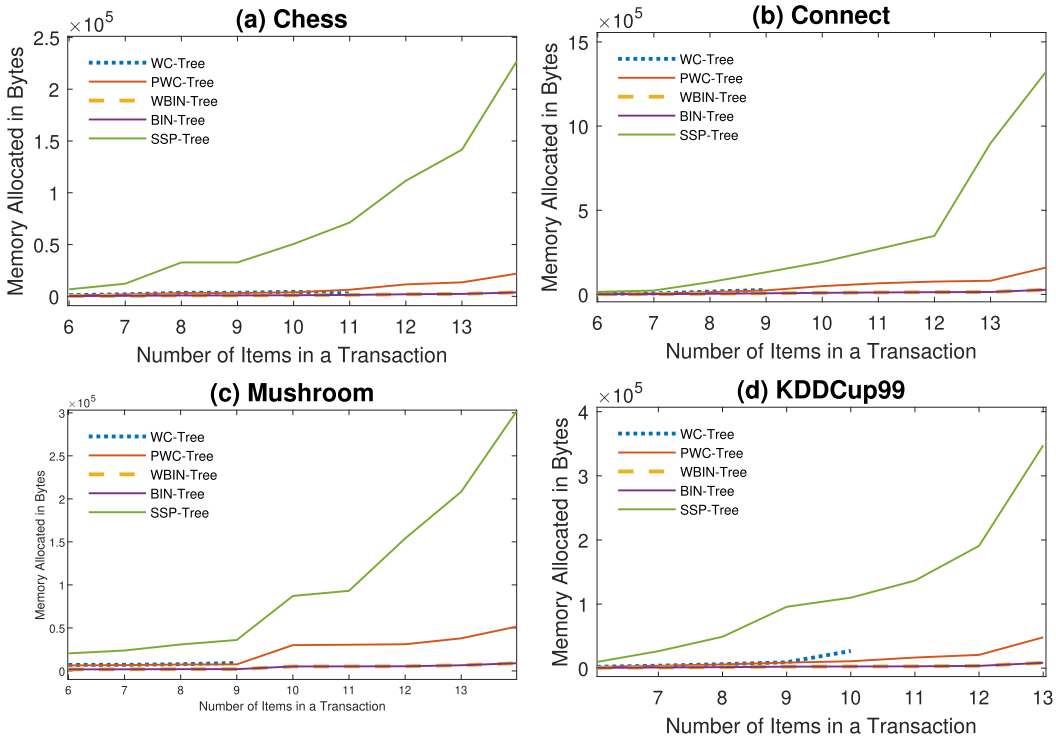

FIGURE 4. Figures 4(a) - 4(d): Memory space occupied by WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree by varying data dimension.

are space efficient than PWC-Tree, SSP-Tree and WC-Tree algorithms.

The outcome of time and space utilization of WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree based on varying data dimension show that for a smaller dataset with a fewer items WC-Tree and BIN-Tree algorithms are nearly as efficient as WBIN-Tree during tree construction though WC-Tree takes more space than BIN-Tree and WBIN-Tree. However, for a large dataset with large number of items, BIN-Tree and WBIN-Tree are equally space efficient. Nevertheless, WBIN-Tree is evaluated as the most time efficient algorithm to store the dataset in the RAM.

2) VARYING DATA SIZE

The size of the dataset is crucial in data mining and as the size of the dataset increases the performance of the mining algorithm also changes in terms of space and time. Therefore, an extensive study to understand the impact of varying dataset size on the efficiencies of WC-Tree, SSP-Tree, PWC-Tree, BIN-Tree and WBIN-Tree are carried out by executing the proposed algorithms on Connect, Chess, KDDCup99, Skin and Mushroom datasets. The significance of the proposed algorithms over existing algorithm is evaluated by the varying size of the dataset.

The time taken by WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree algorithms for reading the data from the secondary memory and storing it in the tree data structure is noted after executing these algorithms on Chess, Skin, Mushroom, Connect and KDDCup99 datasets. The analysis on the results show a few interesting insights regarding the time efficiency of the algorithms on varied type of datasets. It can be observed in Figure 5 that for smaller datasets with large number of items like Chess and Mushroom there is a subtle increase in the time taken by WC-Tree and BIN-Tree algorithms with the increase in the number of transactions. PWC-Tree shows a gradual increase whereas SSP-Tree depicts an exponential increase in time taken to construct the tree with the increasing number of transactions in the dataset. A similar behaviour is observed by these algorithms when executed on large KDDCup99 and Connect datasets. WBIN-Tree algorithm outperforms WC-Tree, PWC-Tree, BIN-Tree and SSP-Tree algorithms when executed on any type of dataset. It is also revealed that WC-Tree is efficient during tree construction of Skin dataset, where the average transaction length is four, despite its limitations on data dimension, when compared to SSP-Tree, PWC-Tree and BIN-Tree algorithms .

Another set of experiments are carried out to determine the space efficiency of WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree on increasing data size. The memory utilized in terms of bytes for various datasets is shown in Figure 6. A huge amount of memory is consumed by SSP-Tree to store the datasets of different size. There is a slight increase in the memory utilized by BIN-Tree and WBIN-Tree algorithms with the increase in the number of transactions of various datasets under consideration. It can also be observed that there is a gradual increase in the memory occupied by WC-Tree & PWC-Tree. Furthermore, BIN-Tree

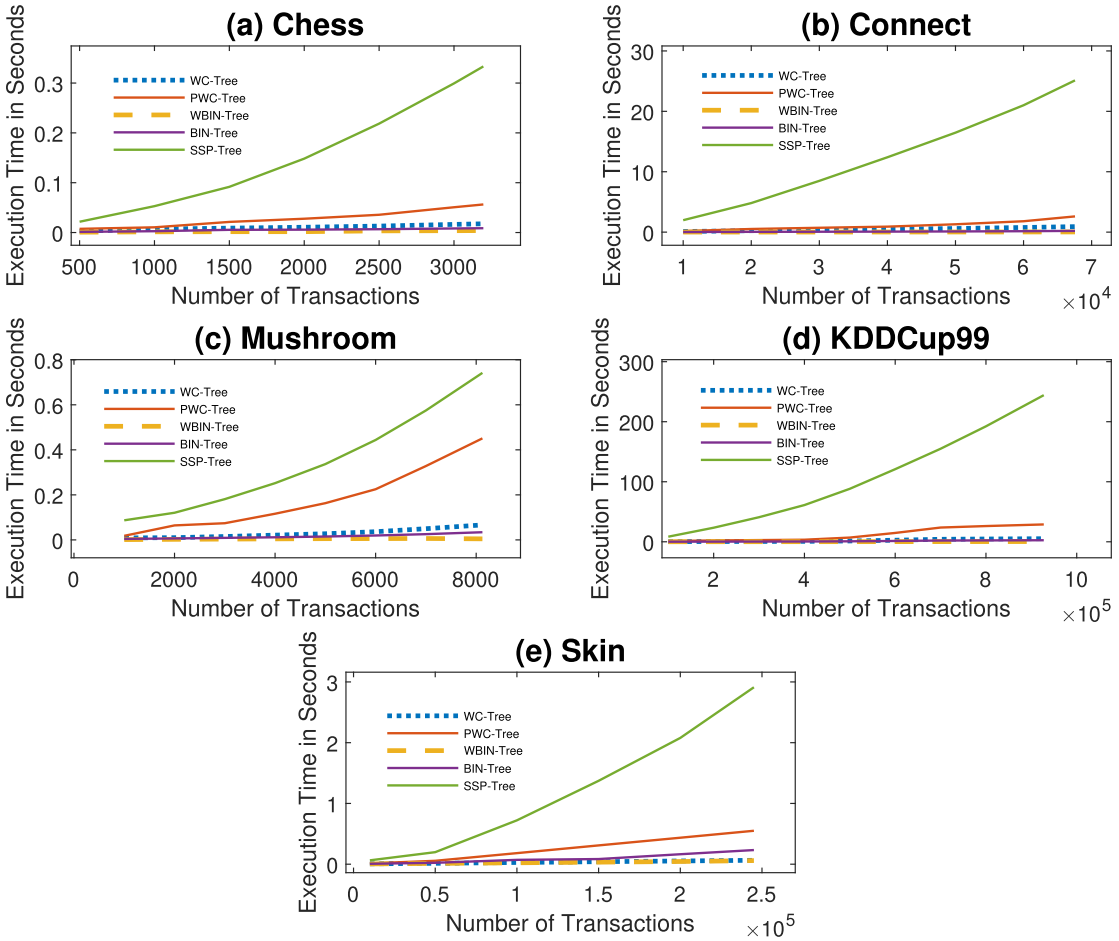

FIGURE 5. Figures 5(a)- 5(e): Time taken to construct WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree by varying data size.

and WBIN-Tree are are space efficient than PWC-Tree, WC-Tree and SSP-Tree with a prominent difference in the amount of space consumed to store the datasets. Thus, it can be concluded that BIN-Tree and WBIN-Tree are suitable for datasets regardless of the size. Furthermore, a dip in the amount of memory occupied by SSP-Tree when executed Mushroom dataset with 2000 transactions is noted. The tree is restructured based on the support of the itemset and number of prefix paths vary based on the support values. SSP-Tree is a prefix tree and if there are more common prefix paths, then the amount of memory occupied will be less in these type of trees which will lead to dip or increase in the amount of main memory utilization.

D. EXPERIMENTAL EVALUATION: MINING RARE AND FREQUENT ITEMSETS

Sequential algorithms are effective for the discovery of interesting itemsets from small datasets. As the size of the dataset increases it is tedious to use sequential algorithms due to the limited usage of resources by the algorithms. The limitation can be overcome by exploiting the available resources' capability using parallel algorithms. The effectiveness and limitations of WBIN-Tree are evaluated by comparing it with two sequential mining algorithms BIN-Tree and SSP-Tree and two parallel algorithms GMiner and Frontier Expansion to discover RFI. BIN-Tree is chosen for the comparison because it was found to be equally space efficient during tree construction and SSP-Tree is chosen due to its nature of storing each itemset as a node in the tree data structure. Furthermore, WBIN-Tree involves parallel technique during mining hence, it is compare with existing parallel algorithms.

The performance of WBIN-Tree against sequential algorithms are analysed by executing those on Connect, Chess, KDDCup99, Mushroom and Skin datasets. RFI are discovered by varying the minimum frequent threshold from 10% to 90% and keeping the minimum rare threshold constant at 0%. Execution time graphs for mining the itemsets in Figures 7(a) to 7(e) show that WBIN-Tree outperforms SSP-Tree and BIN-Tree when the thresholds are set low. However, if the average transaction length is less than 5, then SSP-Tree shows a better performance compared to WBIN-Tree and BIN-Tree algorithms. This is because, the time taken to send and receive the data to and from GPU is more compared to time needed to discover RFI. It can also be observed that, when the database is large and the number of itemsets to be discovered is also large, then WBIN-Tree performs better than SSP and BIN tree algorithms.

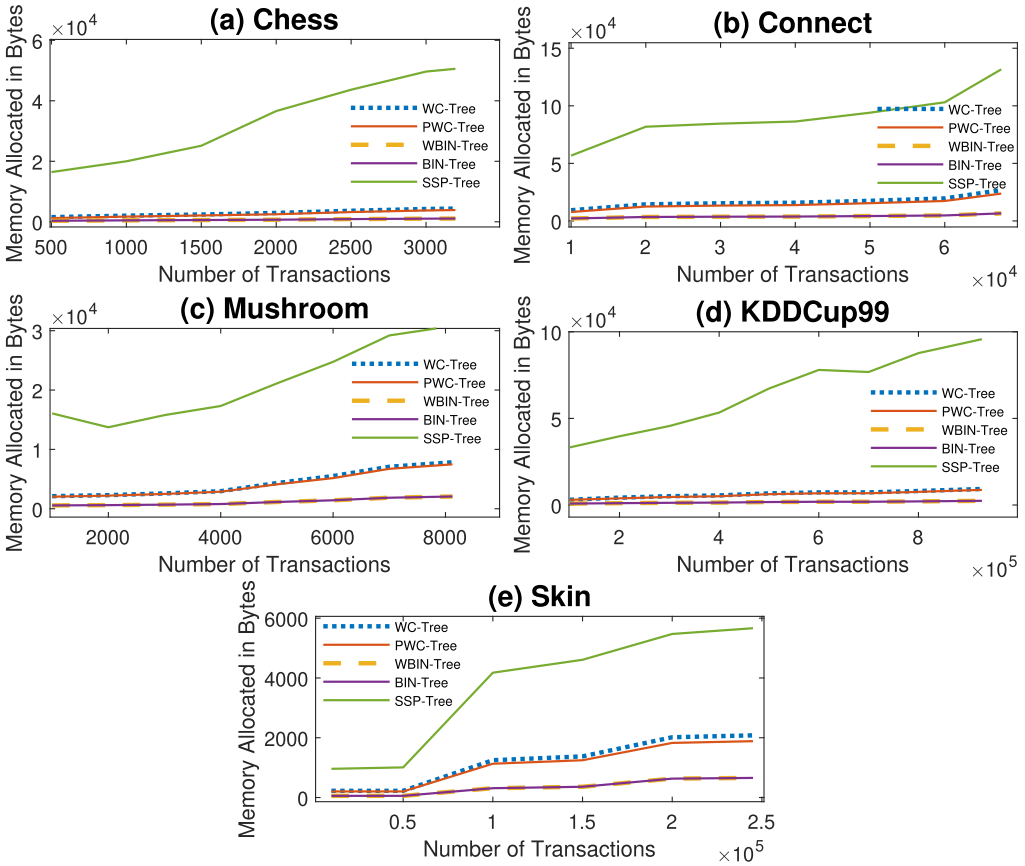

FIGURE 6. Figures 6(a) - 6(e): Memory space occupied by WC-Tree, PWC-Tree, BIN-Tree, WBIN-Tree and SSP-Tree by varying data size.

WBIN-Tree is proved to be efficient on large datasets when compared with sequential algorithms. Additionally, there are a few existing parallel algorithms implemented for the discovery of frequent itemsets from the dataset. Hence, the performance of WBIN-Tree is compared with existing two publicly available parallel algorithms for frequent itemset mining: GMiner [15] and Frontier Expansion [42]. GMiner is implemented on CUDA platform and its implementation is publicly available at [43]. Whereas, Frontier Expansion exploits the capacity of multicores and the implementation is publicly available at [44]. The algorithms are executed on Connect, Skin, Chess, Acute Inflammation and Breast Cancer datasets by setting the minimum frequent support count threshold to 40%.

The behaviour of the proposed and existing parallel mining algorithms to discover RFI are observed after conducting multiple experiments by varying the number of transactions in each dataset.

E. THEORETICAL EVALUATION

The theoretical analysis of time and space efficiencies of WBIN-Tree against existing algorithms are discussed in this section. All algorithms are constructed for a transaction database with n transactions containing i unique itemsets. The node insertion operation in WBIN-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 and (n $\times$ i) steps to read each transaction from the secondary database. Thus, the tree construction operation of WBIN-Tree is in O((n $\times$ i) + n^2)). The total tree construction time for constructing WC-Tree, BIN-Tree and PWC-Tree is in O((n $\times$ i) + n^2)). Hence, theoretically the complexity of proposed algorithm during tree construction is same as WC-Tree, BIN-Tree and PWC-Tree algorithms.

The memory allocated for PWC-Tree is ((n$\times$y) + (m$\times$z)) bytes, where n is the number of transactions, m is the total number of remaining nodes utilized, y is the size of a node and z is the size of a REMNODE in the tree respectively. WC-Tree occupies (((n-l)$\times$p) + (l$\times$q)) where l is the total number of LNodes in the tree and p and q are the size of the Node and LNode in the tree respectively. Whereas, the memory space utilized by BIN-Tree and WBIN-Tree is (n$\times$x) bytes, where x is the size of one node in the tree. The size of x is always less than y because the itemsets are represented a bitset where 1 bit represents 1-itemset whereas, y is the size of the node based on integer datatype. Hence, WBIN-Tree and BIN-Tree are space efficient when compared to WC-Tree and PWC-Tree.

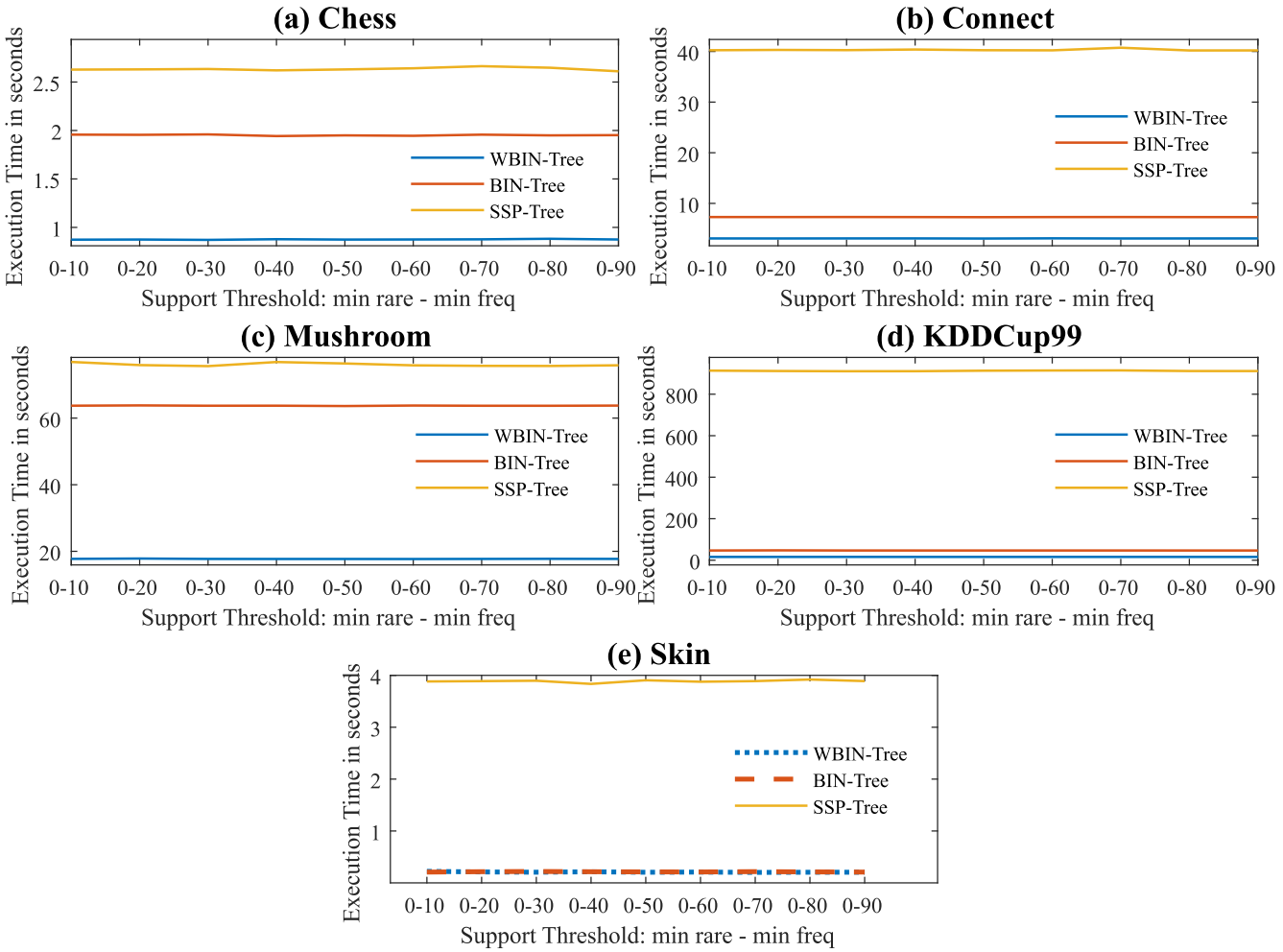

FIGURE 7. Figures 7(a) to 7(e) represent the execution time for discovering rare and frequent itemsets from SSP-Tree, BIN-Tree and WBIN-Tree for various datasets.

VI. DISCUSSION

The time efficiencies of BIN-Tree, WC-Tree, PWC-Tree and SSP-Tree with respect to WBIN-Tree to read the dataset from the secondary memory and store it in the main memory, based on varying data dimension, are given in Table 4. The representation, $x_{AB} \uparrow$ is read as algorithm B mentioned in the row of the table takes x% more time/space than algorithm A mentioned in the column. The representation, $x_{AB} \downarrow$ is read as algorithm A mentioned in the column takes x% more time/space than algorithm B given in the row of the table. The results in the table is self-evident that WBIN-Tree is the most efficient algorithm among other algorithms considered in the experiments to read the dataset from the secondary memory and store it entirely in the main memory without any information loss. WC-Tree takes longer time to construct the tree when compared to WBIN-Tree as each item in the transaction must be mapped to its corresponding prime number and then the Weight is calculated whereas, in WBIN-Tree the item's corresponding position in bitset is set if it is present in the transaction. The efficiency of WBIN-Tree is greater than 95% when compared with PWC-Tree and SSP-Tree for varying data dimension. In addition to the mapping of items to its corresponding prime number, common factors between parent and child node must be removed from the child node before its insertion into the tree which adds to the time taken to construct the tree. SSP-Tree requires restructuring of the tree based on the support of each item in the transaction before inserting the transaction into the tree adding to the tree construction cost. In BIN-Tree the additional step of comparing the parent and child node and removal of common factors from child node, similar to PWC-Tree, makes the algorithm less time efficient than WBIN-Tree even though both use the same data structure to store the data.

TABLE 4. Time efficiency comparison with WBIN-Tree algorithm to construct the tree based on varying data dimension (in percentage).

	WBIN			
Algorithms	**Chess**	**Mushroom**	**Connect**	**KDDCup99**
BIN	71 ↑	84 ↑	77 ↑	79 ↑
WC	73 ↑	90 ↑	93 ↑	89 ↑
PWC	97 ↑	98 ↑	98 ↑	98 ↑
SSP	99 ↑	99 ↑	99 ↑	99 ↑

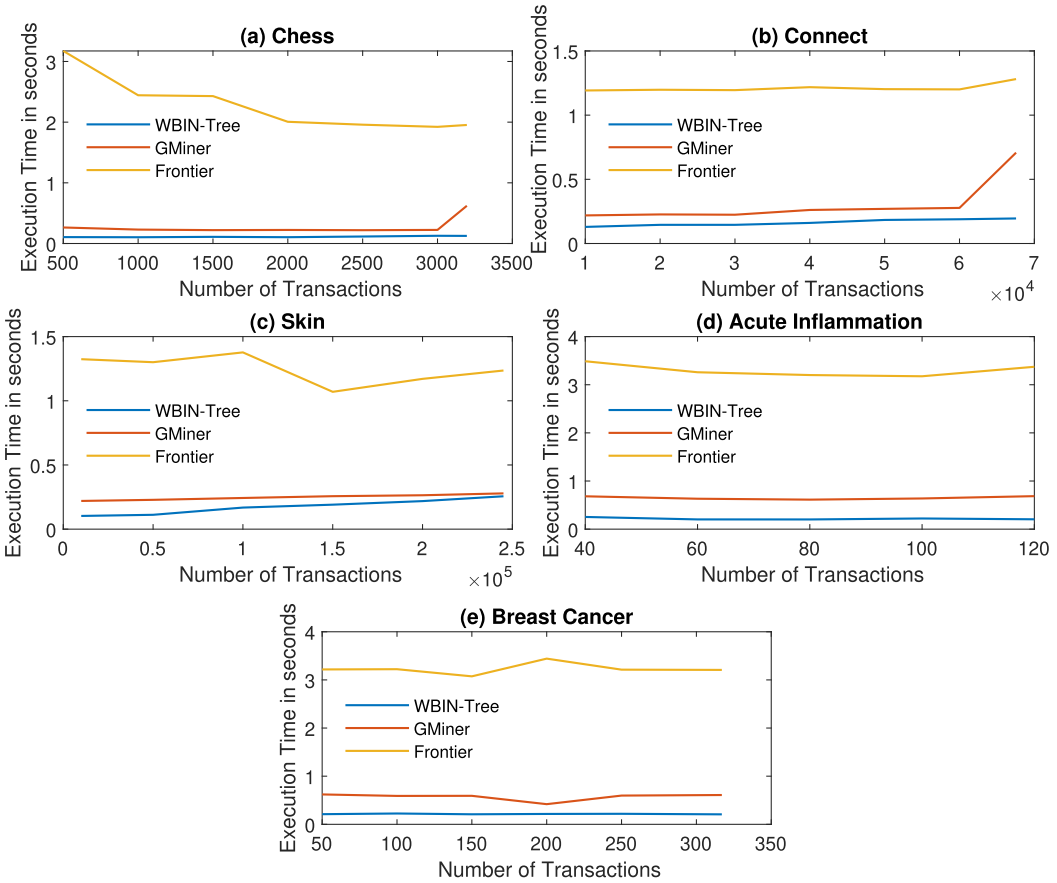

FIGURE 8. Execution time taken to discover frequent and rare itemsets from the datasets using WBIN-Tree, GMiner and Frontier Expansion algorithms.

TABLE 5. Space efficiency comparison with WBIN-Tree algorithm to store the entire dataset in the tree based on varying data dimension (in percentage).

	WBIN			
Algorithms	**Chess**	**Mushroom**	**Connect**	**KDDCup99**
BIN	0	0	0	0
WC	75 ↑	74 ↑	76 ↑	74 ↑
PWC	72 ↑	72 ↑	72 ↑	72 ↑
SSP	97 ↑	92 ↑	94 ↑	96 ↑

TABLE 6. Time efficiency comparison with WBIN-Tree algorithm to construct the tree based on varying data size (in percentage).

	WBIN				
Algorithms	**Chess**	**Mushroom**	**Connect**	**Skin**	**KDDCup99**
BIN	57↑	73↑	81↑	69↑	82↑
WC	78↑	85↑	96↑	14↑	90↑
PWC	92↑	98↑	98↑	88↑	98↑
SSP	98↑	99↑	99↑	97↑	99↑

The space efficiency of the algorithms are compared similarly by executing those on various datasets with different size as shown in Table 5. The maximum data dimension considered such that it is the maximum information stored in WC-Tree without information loss. BIN-Tree occupies the same amount of space as compared to WBIN-Tree to store the entire dataset as these two data structures use same representation of the data. WC-Tree and PWC-Tree occupy comparable amount of space in the main memory. Since common factors are stored in the parent node, PWC-Tree requires less number of REMNODEs and WC-Tree will require large nodes thus increasing the need for memory. SSP-Tree is based on prefix tree and hence leaves a large memory footprint.

The time efficiency of the proposed and existing algorithms based on varying data size are compared with respect to WBIN-Tree and the percentage wise average run time on different datasets is shown in Table 6. The performance of WBIN-Tree against BIN-Tree, WC-Tree, PWC-Tree and SSP-Tree is found to 50% more efficient on any size of dataset. There is a slight difference in the performance of WC-Tree and WBIN-Tree when executed on the Skin dataset where the number of attributes are less in the dataset. The common factor removal procedure in the BIN-Tree and PWC-Tree algorithms add to tree construction cost and hence with the increase in the number of transactions in the dataset the efficiency of the algorithms decrease. The performance of the SSP-Tree depends on the number of transactions that require restructuring of the tree before inserting it into the tree. The restructuring of the tree before insertion of every transaction decreases the efficiency of the algorithm.

The average space efficiency of the WC-Tree, PWC-Tree, BIN-Tree and SSP-Tree with respect to WBIN-Tree based

TABLE 7. Space efficiency comparison with WBIN-Tree algorithm to store the entire dataset in the tree based on varying data size (in percentage).

	WBIN				
Algorithm	Chess	Mushroom	Connect	Skin	KDDCup99
BIN	0	0	0	0	0
WC	76↑	74↑	75↑	71↑	74↑
PWC	72↑	72↑	72↑	68↑	72↑
SSP	97↑	94↑	95↑	90↑	97↑

TABLE 8. Speed up comparison with WBIN-Tree to discover rare and frequent itemsets.

Dataset	BIN
Chess	2.2
Mushroom	3.62
Connect	2.33
Skin	0.764
KDDCup99	2.9

on varying data size is given in Table 7. The table contains the average value of space utilised by different datasets considered during the experiments conducted. BIN-Tree and WBIN-Tree occupy the same amount of space in the main memory whereas the efficiency of WBIN-Tree against WC-Tree vary between 71% to 76% based on the size of the dataset and length of the transaction. WBIN-Tree is 72% more efficient than PWC-Tree on datasets with bigger transaction length and more than 90% space efficient when compare to SSP-Tree. The compact and abstract representation of the dataset helps is efficient memory space utilization whereas, prefix trees like SSP-Tree occupies larger amount of space in the main memory to store the dataset.

The speed up of parallel mining algorithm, WBIN-Tree, against sequential mining algorithm, BIN-Tree is recorded to identify the most suitable algorithm on different types of datasets. It is the ratio of the execution time taken by a sequential algorithm over a corresponding parallel algorithm as shown in the equation 7. Parallel algorithms are not suitable in all cases as the time taken to transfer the data from CPU to GPU adds to the execution cost. Hence an analysis on the speed up of the WBIN-Tree algorithm against BIN-Tree is carried out to determine the most efficient algorithm for different varieties of datasets.

$$Speedup = \frac{Sequential\ execution\ time}{Parallel\ execution\ time} \tag{7}$$

A speed up comparison to discover RFI is given in Table 8. WBIN-Tree has a speed up factor of 2.2 when compared to BIN-Tree for mining RFI from Chess dataset. The speed up factor of WBIN-Tree against BIN-Tree is fairly good on Mushroom, Connect and KDDCup99 datasets. However, on Skin dataset where the number of items per transaction is very less BIN-Tree shows a better performance. Hence, it can be deduced that parallel mining is not suitable on datasets with smaller transaction length.

A comparison of average time efficiency of GMiner and Frontier Expansion algorithm with respect to WBIN-Tree when executed on different datasets is given in Table 9. It is found that WBIN-Tree is 93% and 95% more efficient than Frontier Expansion algorithm when executed on Acute Inflammation, Breast Cancer and Chess datasets with less than 4000 transactions. As the number of transactions increases, the efficiency of Frontier Expansion also increase. However, WBIN-Tree is 86% and 85% faster than Frontier Expansion when executed on Connect and Skin datasets respectively. WBIN-Tree is 29% more efficient than GMiner algorithm when executed on Skin dataset, a large dataset with smaller transaction length. As the transaction length and the number of transactions increase in the dataset, the efficiency of WBIN-Tree increases when compared to GMiner algorithm.

TABLE 9. Time efficiency comparison with WBIN-Tree algorithm to discover RFI based on varying data size (in percentage).

Dataset	GMiner	Frontier
Chess	60	95
Connect	47	86
Skin	29	85
Acute Inflammation	43	93.33
Breast Cancer	65	93.43

WBIN-Tree is the most time and space efficient algorithm but the number of attributes in the dataset is limited by the size of the GPU. The subset generation is a procedure used in the discovery of association rules and the size of the subset must be less than or equal to the size of the GPU. This limitation must be addressed to cater the datasets with large dimensions. Further, if the allocated memory for bitset is small, then the data will not fit and the algorithm will throw error. If the allocated memory is large and number of attributes are less, then the memory is wasted. Hence, a dynamic method must be employed to allocate the memory dynamically.

VII. CASE STUDY: DETECTION OF BREAST CANCER

Breast cancer is a collection abnormal cell growth found in the breast and most commonly seen in women. An early detection of this abnormality and a prompt diagnosis can save several lives from further complications. ARM is one of the techniques used to analyse the symptoms for early detection of cancer and rare correlation among breast cancer related micro-array data in pregnant women [45], [46]. Research works are also carried out in various directions such as analysis of association between overall survival of the patient and the duration in weeks from the diagnosis of breast cancer to surgery. These analysis help in reducing the mortality and morbidity of the patients [47].

The analysis of association rules discovered is a technique to uncover the association among the itemsets. Several evaluation measures such as Confidence, Lift, Cosine are used to discover strong and interesting association rules among itemsets. Though Support, Confidence and Lift are the most prominent evaluation measures used in various fields for analysis, they have a few limitations when the association rules contain rare itemsets in the antecedent or consequent of the rule. These null-variant measures have a direct influence on the datasets containing null transactions. Null-invariant evaluation measures are used to assess the significance of

discovered strong rules that contain rare itemsets. Cosine and Kulczynski are null-invariant, symmetric evaluation measures to discover strong interesting rules. All null-invariant measures are symmetric in nature which identifies (X→ Y) is same as (Y→ X). This implication is not true in every situation. Hence, in addition to null-invariant measures, asymmetric and null-variant measures such as Confidence, Lift and Conviction are considered in this research work for the analysis of strong and interesting rules. Interesting association rules are discovered from Breast Cancer [40] dataset among strong association rules that are discovered based on the five evaluation measures mentioned earlier.

A. DATASET COLLECTION

The Breast Cancer dataset is collected from Kaggle website [40] and it contains 317 records of patients who have undergone surgery to remove the tumour. In a few rows the data related to patient's mortality were missing and hence ignored. Six attributes having categorical values, gender of the patient (Male-1, Female-2), tumor stage(I-3, II-4, III-5), Histology (Infiltrating Ductal Carcinoma-6, Infiltrating Lobular Carcinoma-7, Mucinous Carcinoma-8), HER2 status (Positive-11, Negative-12), Surgery_type (Lumpectomy-13, Simple Mastectomy-14, Modified Radical Mastectomy-15, Other-16), Patient_Status (Alive-9, Dead-10) found in the dataset are considered. The Patient_status is NULL there is no information available whether the patient is alive or deceased or if the patient didn't visit after the surgery. These type of records in the dataset are considered as transactions with missing values and such records are not considered.

B. PERFORMANCE EVALUATION

Rare and frequent itemsets are discovered by executing WBIN-Tree algorithm on the Breast Cancer dataset by setting frequent threshold to 40% and rare threshold to 10%. Lift, Conviction, Kulczynski and Cosine evaluation measures are applied on 72 association rules containing rare antecedents and frequent consequent. The distribution of rules is shown in Figure 9. Out of the 72 rules discovered, Lift and Conviction measures identified 21 rules as strong and interesting, whereas 51 rules are identified as poorly correlated rules. However, none of the rules are identified as strongly correlated and all 72 are found to be negatively correlated by Cosine measure. Furthermore, Kulczynski identified 59 poorly correlated rules and 13 strong and interesting rules from the association rules discovered.

The null-variant measures discovered 21 interesting rules with values slightly higher than 1. There are no rules discovered by Conviction with ∞, indicating that the rule holds 100% of the times. The null-invariant Cosine measure value is below 0.5, indicating that there are no strongly correlated rules with rare antecedent and frequent consequent in the Breast Cancer dataset. Kulczynski discovered 13 strongly correlated rules from the discovered association rules that are identified as poorly correlated as per Lift and Conviction measures. Hence, there are no interesting rules

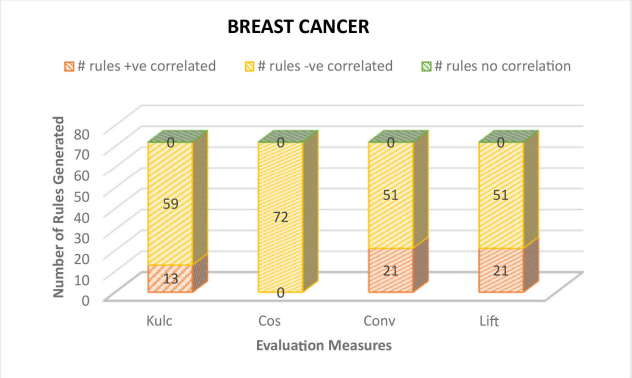

FIGURE 9. Number of rules discovered using evaluation measures: Lift, Conviction, cosine for breast cancer dataset.

TABLE 10. Sample association rules discovered and its values based in evaluation measures: Lift, Conviction (Conv), Cosine (Cos) and Kulczynski (Kulc) for breast cancer dataset.

Sl. No.	Rules	Lift	Conv	Cos	Kulc
1	5,12,15 → 6,9	1.0938	1.1386	0.2692	0.3675
2	5,12 → 6,9	1.1068	1.1609	0.3737	0.4242
3	5,6,12→ 9	0.9945	0.9779	0.3542	0.4784
4	6,12,15 → 4	0.8005	0.7923	0.2513	0.2967

with rare antecedent and frequent consequent as per the values calculated using the evaluation measures.

Four association rules discovered with their evaluation measures values are shown in Table 10. The first rule discovered shows that based on Lift and Conviction values among the patient records under consideration, women who were in third stage tumor of breast cancer(5), negative HER2 status (12) and have undergone Modified Radical Mastectomy(15) are strongly correlated with Infiltrating Ductal Carcinoma (6) type of cancer and alive post surgery (9). Similarly, second rule shows that women who were in third stage tumor of breast cancer(5) and negative HER2 status (12) are strongly correlated with Infiltrating Ductal Carcinoma (6) type of cancer and alive post surgery (9). However, Cosine and Kulczynski values show that the rare itemsets in antecedents and frequent itemsets in the consequent of these two rules are poorly correlated. Further, third and fourth rules are poorly correlated as per all four evaluation measures.

The analysis on rules given in Table 10 summarizes that even though Cosine measure is showing a negative correlation among the rare and frequent itemsets, it can be concluded that based on the Lift and Conviction values the patient in third stage of tumor can undergo Modified Radical Mastectomy which shows a high survival rate post surgery. This kind of association among rare symptoms and frequent disease shows interesting hidden information that will be useful to medical practitioners for early diagnosis of the disease.

VIII. CONCLUSION

WBIN-Tree is a compact, complete and abstract tree data structure designed and developed to read the dataset from the secondary memory using a single database scan. The performance of WBIN-Tree to read and store the dataset

is compared with existing PWC-Tree, BIN-Tree, WC-Tree and SSP-Tree algorithms. The results revealed that WBIN-Tree more space efficient than WC-Tree, PWC-Tree, SSP-Tree algorithms and equally space efficient when compare to BIN-Tree algorithm. Nevertheless, WBIN-Tree is the most time efficient during tree construction when executed on large datasets. The performance of WBIN-Tree for mining is compared with sequential and parallel algorithms. The experimentation results revealed that for a dataset with average transaction length less than 5 the sequential mining methods utilised in SSP-Tree and BIN-Tree are more efficient than WBIN-Tree. However, for a larger dataset WBIN-Tree outperforms existing sequential and parallel mining algorithms. Since most of the real time data sets have larger data dimensions, WBIN-Tree is the most efficient algorithm to discover the association rules among the itemsets.

Strong and interesting association rules among the itemsets with rare antecedents and frequent consequent are useful especially in the medical field. A combination of rare symptoms causing a frequent disease is usually ignored and it may lead to serious problems. Hence, the association rules are discovered using Lift, Confidence, Conviction, Cosine and Kulczynski evaluation measures. These evaluation measures are employed based on its null-in-variance and symmetric properties. At present, WBIN algorithm is executed on a small Breast Cancer dataset. It can be executed on a large primary or secondary dataset, if available, for discovering more interesting rules.

WBIN-Tree can be used to discover the association rules that contain rare antecedents and consequent, frequent antecedents and consequent, rare antecedents and frequent consequent or frequent antecedents and rare consequent from datasets of various sizes in different fields of research such as medicine, security and business. However, the number of attributes in the dataset that can be stored in WBIN-Tree is limited by the size of the GPU during mining. This limitation can be handled by dividing the attributes into sub-attributes. The memory for the bitset data structure used in WBIN-Tree is reserved using static allocation. Static allocation leads to several concerns during run time. Hence, a dynamic allocation of the bitset memory may be employed to utilize the memory suitably.

REFERENCES

[1] A. Methaila, P. Kansal, H. Arya, and P. Kumar, "Early heart disease prediction using data mining techniques," *Comput. Sci. Inf. Technol. J.*, vol. 28, pp. 53–59, Aug. 2014.

[2] P. Jia, J. Zhang, B. Zhao, H. Li, and X. Liu, "Privacy-preserving association rule mining via multi-key fully homomorphic encryption," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 2, pp. 641–650, Feb. 2023.

[3] C.-H. Chee, J. Jaafar, I. A. Aziz, M. H. Hasan, and W. Yeoh, "Algorithms for frequent itemset mining: A literature review," *Artif. Intell. Rev.*, vol. 52, no. 4, pp. 2603–2621, Dec. 2019.

[4] S. Darrab, D. Broneske, and G. Saake, "Modern applications and challenges for rare itemset mining," *Int. J. Mach. Learn. Comput.*, vol. 11, no. 3, pp. 208–218, May 2021.

[5] A. Borah and B. Nath, "Incremental rare pattern based approach for identifying outliers in medical data," *Appl. Soft Comput.*, vol. 85, Dec. 2019, Art. no. 105824.

[6] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. Int. Conf. Very Large Data Bases*, vol. 1215, 1994, pp. 487–499.

[7] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, Jun. 2000.

[8] S. Rai, M. Geetha, P. Kumar, and B. Giridhar, "Partial weighted count tree for discovery of rare and frequent itemsets," *Engineered Sci.*, vol. 20, pp. 284–295, Jun. 2022.

[9] S. Rai, M. Geetha, P. Kumar, and B. Giridhar, "Binary count tree: An efficient and compact structure for mining rare and frequent itemsets," *Engineered Sci.*, vol. 17, pp. 185–194, Jan. 2022.

[10] F. Liu and X. Zhang, "Hypertension and obesity: Risk factors for thyroid disease," *Frontiers Endocrinol.*, vol. 13, Jul. 2022, Art. no. 939367.

[11] B. Xiao and G. Piao, "Analysis of influencing factors and enterprise strategy of online consumer behavior decision based on association rules and mobile computing," *Wireless Commun. Mobile Comput.*, vol. 2022, pp. 1–9, Mar. 2022.

[12] N. Jia and Z. Madina, "An association rule-based multiresource mining method for MOOC teaching," *Comput. Math. Methods Med.*, vol. 2022, pp. 1–7, Feb. 2022.

[13] A. Borah and B. Nath, "Identifying risk factors for adverse diseases using dynamic rare association rule mining," *Exp. Syst. Appl.*, vol. 113, pp. 233–263, Dec. 2018.

[14] K.-W. Chon and M.-S. Kim, "BIGMiner: A fast and scalable distributed frequent pattern miner for big data," *Cluster Comput.*, vol. 21, no. 3, pp. 1507–1520, Sep. 2018.

[15] K.-W. Chon, S.-H. Hwang, and M.-S. Kim, "GMiner: A fast GPU-based frequent itemset mining method for large-scale data," *Inf. Sci.*, vols. 439–440, pp. 19–38, May 2018.

[16] M. A. Benatia, D. Baudry, and A. Louis, "Detecting counterfeit products by means of frequent pattern mining," *J. Ambient Intell. Humanized Comput.*, vol. 13, no. 7, pp. 3683–3692, Jul. 2022.

[17] P. Lenca, B. Vaillant, P. Meyer, and S. Lallich, "Association rule interestingness measures: Experimental and theoretical studies," in *Quality Measures in Data Mining*. Berlin, Germany: Springer, 2007, pp. 51–76.

[18] A. Verma, K. Dhalmahapatra, and J. Maiti, "Forecasting occupational safety performance and mining text-based association rules for incident occurrences," *Saf. Sci.*, vol. 159, Mar. 2023, Art. no. 106014.

[19] M. Hahsler. *A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules*. Accessed: Feb. 27, 2023. [Online]. Available: https://mhahsler.github.io/arules/docs/measures

[20] H. Najadat, A. Shatnawi, and G. Obiedat, "A new perfect hashing and pruning algorithm for mining association rule," *Commun. IBIMA*, vol. 4, pp. 2524–2531, Jan. 2011.

[21] M. Geetha and R. D'Souza, "An efficient discovery of frequent concepts using weighted count tree," *Inst. Comput. Sci., Social Informat. Telecommun. Eng.*, vol. 539, pp. 367–370, Jan. 2012.

[22] N. Shahbazi, R. Soltani, J. Gryz, and A. An, "Building FP-tree on the fly: Single-pass frequent itemset mining," in *Machine Learning and Data Mining in Pattern Recognition*. Berlin, Germany: Springer, 2016, pp. 387–400.

[23] Y. Djenouri, D. Djenouri, J. C. Lin, and A. Belhadi, "Frequent itemset mining in big data with effective single scan algorithms," *IEEE Access*, vol. 6, pp. 68013–68026, 2018.

[24] V. Kadappa and S. Nagesh, "Local support-based partition algorithm for frequent pattern mining," *Pattern Anal. Appl.*, vol. 22, no. 3, pp. 1137–1147, Aug. 2019.

[25] J. Sun, Y. Xun, J. Zhang, and J. Li, "Incremental frequent itemsets mining with FCFP tree," *IEEE Access*, vol. 7, pp. 136511–136524, 2019.

[26] L.-N. Sun, "An improved apriori algorithm based on support weight matrix for data mining in transaction database," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 2, pp. 495–501, Feb. 2020.

[27] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2003, pp. 326–335.

[28] M. Man, W. A. W. Abu Bakar, M. M. A. Jalil, and J. A. Jusoh, "Postdiffset algorithm in rare pattern: An implementation via benchmark case study," *Int. J. Electr. Comput. Eng.*, vol. 8, pp. 4477–4485, Dec. 2018.

[29] M. Man, J. A. Jusoh, S. I. A. Saany, W. A. W. A. Bakar, and M. H. Ibrahim, "Analysis study on R-Eclat algorithm in infrequent itemsets mining," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 9, no. 6, p. 5446, Dec. 2019.

[30] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *Proc. 19th IEEE Int. Conf. Tools Artif. Intelligence*, Oct. 2007, pp. 305–312.

[31] S. A. Ahmed and B. Nath, "ISSP-tree: An improved fast algorithm for constructing a complete prefix tree using single database scan," *Exp. Syst. Appl.*, vol. 185, Dec. 2021, Art. no. 115603.
[32] M. A. Mahdi, K. M. Hosny, and I. Elhenawy, "FR-tree: A novel rare association rule for big data problem," *Exp. Syst. Appl.*, vol. 187, Jan. 2022, Art. no. 115898.
[33] M. Mahdi, S. Abdelrahman, R. Bahgat, and I. Ismail, "F-tree: An algorithm for clustering transactional data using frequency tree," 2017, *arXiv:1705.00761*.
[34] A. Borah and B. Nath, "Tree based frequent and rare pattern mining techniques: A comprehensive structural and empirical analysis," *Social Netw. Appl. Sci.*, vol. 1, no. 9, pp. 1–18, Sep. 2019.
[35] K.-W. Chon, E. Yi, and M.-S. Kim, "SGMiner: A fast and scalable GPU-based frequent pattern miner on SSDs," *IEEE Access*, vol. 10, pp. 62502–62519, 2022.
[36] Y. Djenouri, D. Djenouri, A. Belhadi, and A. Cano, "Exploiting GPU and cluster parallelism in single scan frequent itemset mining," *Inf. Sci.*, vol. 496, pp. 363–377, Sep. 2019.
[37] P. D. McNicholas, T. B. Murphy, and M. O'Regan, "Standardising the lift of an association rule," *Comput. Statist. Data Anal.*, vol. 52, no. 10, pp. 4712–4721, Jun. 2008.
[38] P. Fournier-Viger, J. C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The spmf open-source data mining library version 2," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, 2016, pp. 36–40.
[39] J. Czerniak and H. Zarzycki, "Application of rough sets in the presumptive diagnosis of urinary system diseases," in *Artificial Intelligence and Security in Computing Systems*. Berlin, Germany: Springer, 2003, pp. 41–51.
[40] QUBC. Research. *Real Breast Cancer Data*. Accessed: Jan. 23, 2023. [Online]. Available: https://www.kaggle.com/datasets/amandam1/breastcancerdataset
[41] A. Borah and B. Nath, "Rare association rule mining from incremental databases," *Pattern Anal. Appl.*, vol. 23, no. 1, pp. 113–134, Feb. 2020.
[42] F. Zhang, Y. Zhang, and J. D. Bakos, "Accelerating frequent itemset mining on graphics processing units," *J. Supercomput.*, vol. 66, no. 1, pp. 94–117, Oct. 2013.
[43] P. Agrawal. *Gminer*. Accessed: Jan. 23, 2023. [Online]. Available: https://github.com/coderbond007/GMiner
[44] Zhangfan. *Frontier Expansion*. Accessed: Jan. 23, 2023. [Online]. Available: https://github.com/zhangfan0726/fim_gpu
[45] M. Jajroudi, M. Enferadi, Z. Bagherpour, and R. Reiazi, "Association rule mining-based radiomics in breast cancer diagnosis," *Iranian J. Med. Phys.*, Jan. 2023.
[46] S. Bouasker, W. Inoubli, S. B. Yahia, and G. Diallo, "Pregnancy associated breast cancer gene expressions : New insights on their regulation based on rare correlated patterns," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 18, no. 3, pp. 1035–1048, May 2021.
[47] A. A. Wiener, B. M. Hanlon, J. R. Schumacher, K. A. Vande Walle, L. G. Wilke, and H. B. Neuman, "Reexamining time from breast cancer diagnosis to primary breast surgery," *JAMA Surg.*, vol. 158, no. 5, p. 485, May 2023.

SHWETHA RAI received the M.Tech. degree in computer science from the Manipal Institute of Technology, Manipal, India, and the Ph.D. degree in the area of data mining from the Manipal Academy of Higher Education, Manipal. She is currently an Assistant Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education. Her current research interests include data mining and parallel computing. She has presented several papers in national and international conferences and her work has been published in various international journals.

PREETHAM KUMAR received the Ph.D. degree in data mining from the National Institute of Technology, Karnataka. He is currently the Deputy Registrar-Academics (Technical) with the Manipal Academy of Higher Education, Manipal, India, and a Professor with the Department of Information and Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education. His research interests include data mining, image processing, bioinformatics, advanced database management systems, operating systems, software architecture, and software engineering.

K. NAKUL SHETTY received the M.Tech. degree in digital electronics and communication from the Manipal Institute of Technology, Manipal, India, in 2008. He is currently pursuing the Ph.D. degree in the area of microfluidics with the Manipal Academy of Higher Education. He is also an Assistant Professor with the Department of Electronics and Communication Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education. His research interests include data mining and microfluidics.

M. GEETHA received the Ph.D. degree from the National Institute of Technology, Karnataka, Surathkal. She is currently the Director (Student Affairs) with the Manipal Academy of Higher Education, Manipal, India, and a Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education. Her current research interests include text mining in healthcare and financial sectors, and data mining. She has attended and presented several papers in national and international conferences and her work has been published in various international journals.

B. GIRIDHAR received the B.Tech. degree from the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal. His current research interests include data mining and data protection. His work has been published in international journals.