INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

A Scalable High Utilization Itemset Mining Technique for Large Datasets Using A Bit-Based Model

Aditya Nellutla[1]*, **N Srinivasan**[2]

1 Research Scholar, Sathyabama University, Chennai, Tamil Nadu, India
2 Professor, Department of Computer Science and Engineering, Rajalakshmi Engineering College, Chennai, Tamil Nadu, India

OPEN ACCESS

Received: 06-10-2022

Accepted: 01-12-2022

Published: 23-01-2023

Citation: Nellutla A, Srinivasan N (2023) A Scalable High Utilization Itemset Mining Technique for Large Datasets Using A Bit-Based Model. Indian Journal of Science and Technology 16(3): 204-213. https://doi.org/10.17485/IJST/v16i3.1972

* **Corresponding author**.

adityaresearch3@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: Utility-list based algorithms have gained a lot of traction due to their efficiency and the ease with which they may be modified. While there have been some enhancements, the problem of inefficiency persists. This research presents a solution to this issue by enhancing the utility-list building process, a crucial function that has received little attention in previous studies. Also, the research aims at reducing memory complexity and better performance than existing approaches. **Methods:** To expedite building, a new set of bitwise operations termed Bit combine construction (BCC) is proposed. In addition, BCC is supported by a unique data format called EBP (Efficiency Bit Partition). An innovative EBP-Miner algorithm is developed with this framework in mind, and it uses many techniques to narrow the search field. **Findings:** On widely used baseline methods, experimental findings reveal that EBP-Miner outperforms numerous state-of-the-art techniques, including FEACP as well as CLH-Miner approaches. The experiments were conducted with utilization value ranging from 20% to 100% of the nodes. The proposed system achieves an average of 390s runtime and utilization value of 90.25% which are outperformed the existing methods. Also, the approach has proven 20% lesser memory complexity than of the existing algorithms. **Novelty:** In the field of data mining, high utility itemset mining (HUIM) is an important challenge. The idea is to discover groups of data in a database that are particularly significant or profitable in order to unearth information that can aid in making decisions. The novelty of this study is on developing a better method for building algorithms for HUIM that make use of a bitwise data structure, and on suggesting a more time- and effort-effective strategy for building utility-lists.

Keywords: High Utility Itemsets; Data Mining; Optimization Model; Bitwise Operations; Pattern Mining

1 Introduction

Consumer research, banking transactions, and even biological studies all make use of data mining, making it a crucial area of study. Data mining relies heavily on frequent itemset mining (FIM), which provides a complete list of all the transactional sets that occur frequently. Although it considers whether or not an itemset appears in a transaction, it disregards the quantity, size, profitability, and significance of the transaction information. As a result, FIM may come into dull collections that don't bring in much money(1). High Utility Item Set Mining (HUIM) is a refinement of FIM in which the utility of each item is quantified.

This metric deliberates the significance and quantity of item sets, and it may be used to mine profitable item sets from a transaction database. In the retail industry, for instance, HUIM can use market basket to suggest popular bundles of products. They may only available sometimes, however HUIM disregards this property of itemsets, hence it may be unsuitable to utilise these itemsets as sales choices. As a result, it's important to evaluate each set of goods not just based on how often they're used but also in terms of how much UO they require. Most research has focused on either profitability as key metric. However, there are restrictions associated with relying just on these two metrics. Our goal in this work is to discover sets of objects with a high utility occupancy (HUOI) that pass both the support and the occupancy tests.

With the help of the UO metric, we can make sure that HUOI appears frequently in transactions and consumes a sizable portion of their utility when it does. By considering the itemset's contribution, this provides higher practical value and facilitates better optimising judgments. Whereas FIM and HUIM have received considerable attention, relatively few methods exist while considering of their usefulness. The fundamental difficulty in this study comes from the fact that utility utilization would be neither monotonous nor anti-monotonic; the UO of a superset of itemset may be more than, less than, or the same as the UO of the itemset itself(2).

Furthermore, as HUOIM requires the calculation of supporting and for this, it is often costlier of such activities. This work is focused on HUOIM on large data since, in practise, the size of the database that has to be mined grows exponentially. Current methods like the HUOPM algorithm and the OCEAN algorithm are examined, and it is shown that they need to load the complete dataset simultaneously. Consequently, these are limited in the size of the datasets they can handle, and their performance suffers greatly when dealing with huge amounts of information. Knowledge discovery is greatly aided by data mining.

Data mining algorithms are able to sift through vast amounts of information and unearth nuggets of value for their end consumers. Itemsets, a type of knowledge uncovered via this process, can aid in comprehension of the facts and decision-making in a wide variety of practical contexts. Sets of things may be found in a wide variety of data structures. When it comes to finding new sets, regular itemset mining is a common assignment FIM, also known as frequent pattern mining. Algorithms addressing this problem discover all groups of items in a transaction database that meet a minimal support level chosen by the user. Specifically, for this purpose, the well-known Apriori algorithm was created. Apriori searches the database several times and can merge various itemsets to produce larger itemsets, thus while it does use search space trimming qualities to speed up the itemset mining process, it performs badly in many cases(3).

Several algorithms, including such FP-Growth, Eclat, and LCM, were later developed to increase performance while searching for often occurring item sets. Association rule mining (ARM) is a methodology that use the aforementioned techniques to infer robust relationships between items after mining frequent itemsets. Despite FIM and ARM's widespread implementation, they only locate itemsets according to their occurrence frequencies and don't consider other metrics like weight and profit when determining an itemset's significance. To account for this, the FIM issue was recast as one of mining high-utility sets (HUIM). A transaction's items are assigned two positive numbers in HUIM, one for the internal utility (or buy quantity) and one for the exterior utility (or sales price) (or unit profit). The primary objective of HUIM is to discover the groups of products that are the most profitable in a statistical dataframe(4).

Like FIM, HUIM employs a threshold determined by the user to exclude irrelevant data sets. The necessary utility threshold is the level of usefulness an item set must have in order to be labelled as "high utility" (HUI). Despite being an extension of FIM, HUIM is more difficult than FIM because the utility function included to evaluate the relevance of an itemset does not meet the downward closure property, also known as the anti-monotonicity property. This indicates that the utility of a set of items may be less than, equal to, or higher than the utility of any of the sets that make it up. Efficient FIM algorithms could be directly applied to HUIM due to the fact that they rely on the descending closure property to minimise the search space of itemsets. Because of this need, a number of specialised HUIM algorithms were developed. Most of these algorithms are effective, but they don't consider the fact that objects in transaction databases are typically arranged in a taxonomy with nested categories and subcategories.

Peripherals include things like printers and scanners, whereas computers as a whole include things like notebooks and desktops. Furthermore, Computers and Peripherals are specialised subsets of the larger Hardware category. When mining databases for itemsets, a taxonomy's ability to organise items into semantically related, generalised itemsets—itemsets that include generalised objects—can be invaluable. Hardware, Computers, and Peripherals are the catch-all categories. Because conventional HUIM algorithms don't take taxonomy into account, they can only find itemsets that contain items that occur at the leaf node of a transaction database's taxonomy tree. Consequently, even if the results of these algorithms are HUIs, you won't find anything like Peripherals, Computers, or Hardware among them. Based on FIM and ARM, numerous algorithms have been devised that take category data into consideration and cover a broad range of uses[5].

Because the foundation of likelihood of occurrence of frequent itemset is anti-monotonic, the suggested procedures may rapidly identify generalised itemsets. However, due to the utility function's disregard for the downward-closure feature, mining HUIs at varying levels of abstraction, including taxonomies, is far more difficult in HUIM. A new approach, ML-HUI-Miner, was recently introduced by Cagliero et al. to extract Generalized HUIs (GHUIs) in analysing data. GHUIs are itemsets that generate significant profits and can be at varying abstraction levels. A broader challenge of extracting cross-level HUIs and an algorithm by that name (CLH-Miner) were developed in 2020 by Fournier-Viger et al. When it comes to permitting an itemset to combine items from multiple abstraction levels, CLH-Miner overcomes some of the constraints of prior research by offering new upper bounds on the usefulness and effective pruning algorithms. The result is the ability to unearth unique collections of things that would have otherwise eluded discovery by ML-HUI Miner. While ML-HUI Miner as well as HUI-Miner execute quickly and use little memory, CLH-Miner takes significantly more time[6].

Many methods have been proposed since the invention of FIM in transaction databases; these algorithms use the downhill convergence characteristic of the stimulate the development to efficiently tackle the problem at hand. However, a key limitation of FIM is that it does not consider the significance or the quantity of things purchased during a transaction. So, FIM can unearth several groups of products that are often transacted but provide a small profit while ignoring many sets of items that are rarely transacted but generate a huge one. By applying FIM to the more systemic problem of HUIM, this key limitation of FIM has been removed. The process involves mining transaction databases for sets of goods that are particularly significant on the basis of criteria like weight or profit.

Unlike FIM, where the utility measure satisfies the downward closure property, HUIM does not. This makes it a far more difficult problem to solve. That's why it's so important to develop a solid foundation for mining HUIs, which will drastically cut down on the amount of time spent searching. The HUIM problem has been addressed by a number of algorithms. TwoPhase was the first fully-fledged HUIM algorithm, and it was this method that first proposed a utility upper bound (called the Transaction Weighted Utility Upper Bound) (TWU). In this way, the mining process may be sped up by excluding groups of goods that have little to no chance of becoming profitable[7].

Because it performs numerous database searches and can provide unpromising candidate itemsets, Two-Phase can still have a lengthy runtime. Various algorithms, such IHUP and UPGrowth, expanded the pattern growth model provided by Han et al. to further enhance HUIM's efficacy. These algorithms rely heavily on the TWU measurement for solution space reduction, but they do not suffer from the issue of producing unpromising candidates. One drawback of this metric is that it is only an approximation of the utility, therefore it cannot always be used to narrow down the search space. Several improved pruning algorithms and constraints on the size of the issue space were offered. Remaining utility upper boundaries and the utility-list format were first presented in 2012 with the HUI-Miner method.

The EFIM method introduced two additional upper limits based on utility: local utility and sub-tree utility. These upper boundaries are similar to the uBitem constraint and the uBfpe bound. When compared to the TWU and the upper limits of EFIM and other studies, the local utility confidence interval is tighter in most cases, although it is the same as the TWU for sets with only one item[8]. Sub-tree utility upper bound is equal to remaining utility upper bound and uBfpe bound, which were suggested in HUI-Miner and d2HUP, respectively. Local utility bound is comparable to uBitem upper bound of d2HUP. However, these algorithms' actual implementations of these upper limitations throughout the high utility itemset mining phase vary. It was demonstrated, for instance, that EFIM's sub-tree utility is applied sooner than HUI-remaining Miner's function, allowing it to prune a larger number of itemsets. If you're interested in learning more about the limits of EFIM, you may check out.

There are two unique and successful strategies that EFIM created to lessen the expense of database scans and boost the performance of HUIM (HTM). Results showed that EFIM uses less memory and scales practically linearly in complexity with the volume of transactions. In order to further lower the cost of database scans, Nguyen et al. introduced a technique named iMEFIM, which extends EFIM. Additionally, a novel utility structure was provided by the authors to manage transaction databases with movable profit value components in a more suitable manner. There have been recent extensions to the HUIM issue that consider new types of limitations.

Due to the need for a separate phase in which candidates are generated and their usefulness is assessed by a second database search, the existing Apriori-based as well as FP-Growth-based procedures for HUIM are considered to be two-phased. Several early HUIM algorithms employ a two-stage technique, which is inefficient since it necessitates doing repeated database scans and keeping a potentially extremely large number of contenders in memory at once. The HUI-Miner algorithm was developed to deal with this problem. The high utility sets are discovered in a single step using an ECLAT-based strategy. HUI-Miner takes its cue from ECLAT by vertically representing the database in a form known as a utility-list. It just takes one pass over the database to get the useful lists of individual objects. In this way, the utility-lists of every other itemset may be constructed by combining the utility-lists of a number of its constituent subsets. If all the information needed to compute an item's utility is already included in the utility-list of the itemset, then a second database scan is not necessary. As a result, a slew of alternative optimised utility-list-based algorithms were presented[9].

In order to effectively narrow down the search space, FHM includes a two-dimensional matrix. HUP-Miner uses two pruning techniques, called LA-Prune and PU-Prune, to divide each utility-list into many segments. To improve search space pruning, HUI-Miner combines the ideas of a prefix tree and a utility-list to streamline the construction of a prefix tree's conditional subtrees via the join operation of utility-lists. ULB-Miner suggests a merge-like method to enhance the join operation in an array-based implementation of utility-lists. It was demonstrated that this join operation outperformed earlier techniques. HUI-Miner is the most up-to-date utility-list-based method, and it optimises the join process of utility-lists by making use of pointers. Other single-stage algorithms, based on other concepts, were also developed alongside the aforementioned methods. There are algorithms based on hypertext links, in which a series of arrows is used to go from one transaction to the next. The EFIM method uses a horizontal database and transaction merging to make database projections more economically viable[10].

The utility-list format was widely used in these analyses because it is straightforward, effective, and versatile. Many algorithms are built on top of utility lists. While the search itself has been vastly improved by previous algorithms, the cost of each join has seen relatively little innovation. Optimizing the join operation, the most expensive action performed by utility-list based algorithms, has the potential to yield a significant speed boost. In conclusion, utility-list-based algorithms would benefit from the introduction of a linear join operation.

The focus of this study is on developing a better method for building algorithms for HUIM that make use of a bitwise data structure, and on suggesting a more time- and effort-effective strategy for building utility-lists. In addition, a number of techniques for narrowing the search area are used. The four main contributions of this paper are as follows:

- In order to boost the performance of the building operation for utility-list based algorithms, we suggest a set of bitwise operations termed Bit Combine Construction (BCC). This decreases the state-of-the-art algorithm's computational complexity from O (log n) to O (n).
- We propose a new data structure called Efficiency Bit Partition (EBP), which is based on the recently introduced utility list buffer structure and optimises the efficiency of traversing elements for the extensions of two itemsets.
- Using the utility bit partition structure and a number of search space pruning algorithms, a unique high utility itemset mining algorithm is developed; we name it Efficiency Bit Partition Miner (EBP-Miner).
- The effectiveness of the suggested EBP-Miner algorithm is investigated through experimental analysis. The obtained results demonstrate its superiority over other state-of-the-art algorithms on a variety of standard-setting data sets.

2 Methodology

Consider every transaction in a dataset with n transactions $(T = t1, t2, \ldots, tn)$ has a unique id (t_{id}) and belongs to a set $(I = i1, i2, \ldots, im)$. An itemset that contains data that is also a subsection of the itemset I. $eu(i)$ is the unit profit or significance associated with item i in set I. Because the quantity of each item I in t is represented by the internal utility function $iu(i, t)$. All the purchases that have included the itemset X form a set, $X(T)$.

Databases used in real-world applications are typically too big to fit entirely in memory, necessitating partitioning. Conventional algorithms often use horizontal division. In order to determine HUOIs, we must first partition each transaction into itemsets, then compute the performance of each itemset wherever it appears, and lastly combine the results from each partition. A significant amount of extra time is needed for these procedures. This study introduces a suffix-partition-based algorithm for mining high utility occupancy itemsets in large datasets, which is shown to be effective in solving the problem at hand. Two stages make up BA: the first is the partitioning of itemsets, and the second is the mining of HUOIs.

In the first step, all purchases are broken down into smaller groups called "itemsets," and those groups are then assigned to various categories. The second step is to build a set-enumeration tree for each partition, which allows us to mine itemsets in DFS order inside each partition. The itemset partitioning step involves reading the transaction database T into memory, parsing each transaction into a number of itemsets, and then separating those itemsets into distinct partitions based on the suffix items they include. Then, collections with the same final-item-suffix are kept together. The unique item in a 1-itemset is itself, making it the set's suffix item. Then, the itemsets for the following database transaction, t2, are partitioned in the same way.

The database undergoes the same processing, and the transactions are split apart till all of the database's transactions are repeated. Segregating yields non-overlapping subsets of the database, and the whole transaction database T is the union of all partitions. If the preceding is correct, then all partition-generated itemsets must include the suffix item. Support (sup) and

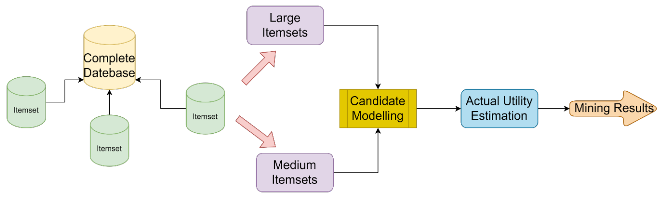

Fig 1. Proposed System architecture

utility occupancy (uo) of itemset may be computed by simply searching and mining the local sup and local uo in the suffix-partition. An itemset's partition sup and uo are identical to the database's global sup and uo. Figure 1 shows the proposed system architecture.

For instance, just the local sup and uo in partition $P1$ need to be computed as the global sup and uo in T while mining itemset $i1$. This greatly improves the efficiency by limiting the calculation of itemsets to the suffix-partition rather than the entire database. At this stage (itemsets mining), the partitions are handled in lexicographic order. For each partition, we build a set-enumeration tree; then, using the DFS search order, we mine itemsets and determine their sup and uo. An itemset is a HUOI if its sup is more than or equal to min sup and its uo is greater than or equal to min uo. Let's pretend t' is a partition transaction; below we'll see an example that shows how utility occupancy pat $uo(X)$ of itemset X in t' relates to the transaction utility part $tu(t')$.

Algorithms for the itemset mining job are generally constrained by the capacity of available memory since databases tend to be too huge to fit in memory in their whole. In this research, we suggest a suffix-based partitioning (SP) technique for conceptually splitting the horizontal database up into many separate, non-overlapping sections. The partition Pi stores the communications for i, also they are formed in Pi need to have the associated value i. We also establish an imposed order for the items of $I = i1,\ i2, \ldots,\ im$, where the components of datasets are ordered in the order. We run a consecutive search on the provided dataset T, where we consider as each operation t includes w unique items, where $t = (t[1],\ t[2], \ldots,\ t[w])$, where $t[j] = (itemj,\ utilj)$. The subsequence $t' = (t[1],\ t[2], \ldots,\ t[j])$ will be printed into partition Pi, where $itemj = I$ is the jth item in t and $util\ j$ is the core efficacy of item I in t. If $\forall j\ 1 \leq j \leq w$, then w is the width of the partition. When we're done with t, we'll divide it up into w subsequences.

After the suffix-based partitions of T have been obtained, the sequential scan will end. All of database T may be thought of as a collection of itemsets, the union of which is the whole. Such partitions do not overlap with one another. Together in definite suffix-based partition, this makes it simple to extract the set of objects and determine their sup and uo. Vertical utility-lists, or a list of all the tid in which an item appears and the utilities it corresponds to, are produced for each item in each partition. The utility lists will be sorted by their support counts, which is a measure of their relative size. The item with the most endorsements appears first in the partition, while the item with the fewest is placed last. The miner construction process Algorithm 1 is shown below.

Algorithm 1: Miner Construction Process

```
Input: Itemutility list (P), the list of objects in the set P for which they are useful.
utility-list of set P(x), abbreviated as ul($P_x$);
the utility-list of set P(y), abbreviated as ul($P_y$).
Output: The result is a utility-list of elements in the set P that have the x and y coordinates.
Steps:
1. Null ul($P_{xy}$) = 1
2. do ul($P_x$) for $E_x$ if and only if $E_x$ is in ul($P_x$)
3. Check whether ul(P) is not null.
4. Lookup in binary for E ul(P) where $E.T_{ID} == E_x.T_{ID}$
5. $E_{xy} = (E_x.T_{ID}, E_x.I_{util} + Ey.I_{util} - E.I_{util}, Ey.R_{util})$. else
6. $E_{xy} = (E_x.T_{ID}, E_x.I_{util} + E_y.I_{util}, E_y.R_{util})$.
7. Insert $E_{xy}$ after ul ($P_{xy}$)
```

As long as the item's support count is below a user-defined threshold, the item and its utility list will continue to be read into memory as long as the support counts of any things ahead of it are not more than the support threshold. Items that do not meet the min sup criteria are not HUOIs, thus it is safe to skip them. In a nutshell, SP employs a vertical layout and sorts support counts-based utility lists in decreasing order. One major benefit is a decrease in the amount of time needed for calculation and input/output operations. It is only necessary to traverse a single partition of the database to compute *sup* and *uo* for each itemset, rather than the full database. The second benefit is that it takes advantage of the closing-down nature of support. It is sufficient for the algorithm to read just those items from the partition whose supports are greater than or equal to min sup while loading the partition.

Partition scanning is concluded and the remaining items in this segment are skipped. As a consequence, the division can be utilised recurrently. The suggested approach is quite similar to the projection method based on barrier technique and loading information edifice. Common methods now include ”branch” or ”initial” projections, ”self interested” projections, ”index-based” projections, and ”Depth” projections.

The key applications of the developed approach specifies the algorithm and the methods of pruning, despite the fact that it uses a partition technique and generates a storage structure that is comparable to that of the projection method. Here are some of the key distinctions between the proposed and the projection model: In contrast to the projection approach, which reads the whole database into memory before processing it, the algorithm processes only one partition at a time, significantly lowering the memory overhead required. Next, the divisions created by the suffix-based method do not overlap with one another. The miner construction process Algorithm 2 is shown below.

Algorithm 2: Mining Construction process

```
Utilities list buffer (ulb), summary of itemset (P), analysis of itemset ($P_x$), and summary of utilities ($P_y$) are all inputs.
Summarize a set of elements, P, using the format su($P_{xy}$).
1. su($P_{xy}$) = 0
2. su = $P_{Pos}$(P).
3.$P_{xPos}$ = su Start P ($P_x$).
4. Assume that Start P, $P_{yPos}$ = su ($P_y$). While $P_{xPos}$ >= su in ($P_x$).
5. $P_{yPos}$ = su ($P_y$).
6. $E_{xp}$ = ulb[$P_{xPos}$], $E_i$ = ulb[$P_{yPos}$], $W_p$ = ulb[$P_{Pos}$];
7. If $E_x.T_{ID}$ is less than $E_y.T_{ID}$, set $P_{xPos}$ = $P_{xPos}$ + 1; otherwise, set it to 7.
8. if $E_x.T_{ID}$ > $E_y.T_{ID}$ then 9: $P_{yPos}$ = $P_{yPos}$ + 1 10: otherwise
9. If su(P) is not null, then 11. while PPos su
10. $E_{xy}$=($E_x.T_{ID}$, $E_x.I_{util}$ + $E_y.I_{util}$ EP.$I_{util}$, $E_y.R_{util}$)
11. while P and $E_P.T_{ID}$ = $E_x.T_{ID}$
12. $P_{Pos}$= $P_{Pos}$ + 1
13. $E_{xy}$=($E_x.T_{ID}$, $E_x.I_{util}$ + $E_y.I_{util}$ EP
14. Update su($P_{xy}$) from ulb; $P_{xPos}$ = $P_{xPos}$ + 1 Return
```

This allows the item set to be mined for each partition independently or simultaneously. Candidates are generated by a node's pathfinding as the projection method. For good measure, the proposed suffix-based segmentation is an instance of prediction, a method of dynamic construction. Our plan is meant to help us spend less time reading meaningless content. Due to the restrictions of *sup* and *uo*, it is possible to immediately pass over a large number of useless objects that don't need to take part in the computation, or even full elements of a subdivision. Fewer resources are needed to store temporary data, and computations are more efficient. As an example, while the projection method reads the entire dataset into memory, the suggested method loads only a tiny subset at a time.

Therefore, it drastically reduces the amount of RAM a mining operation needs. When importing partitions, for instance, this strategy ignores data whose support is less than the limit. This reduces the amount of data that must be moved into memory for processing, which in turn reduces the amount of computational load.As an example, the SHO algorithm makes use of the concept of SP strategy since the RAM cannot retain the transaction database in its whole. We may acquire *sup* and *uo* of an itemset just in a given partition as opposed to accessing the full database by using suffix-based partitioning, as we assume that each partition containing the same suffix-item is too tiny to be retained in the memory. Second, a more space-effective vertical structure is developed to keep track of the utility-list of each object in partition.

As a result, both the input/output cost and computational overhead can be drastically cut down. Therefore, we believe the following to be the key reasons why the performance of the HUOPM algorithm suffers significantly when the size of UO-List is quite huge. It is necessary to continually construct elaborate data structures for use in the HUOPM algorithm and then recover those structures for further processing. New itemsets are mined by performing intersection counting on two tid sets; at the very same time, the UO-List and FU-table are built to store itemset data.

The calculation cost of intersection is particularly high when calculating itemsets with big tid set, especially to 2-candidate-itemsets and the 3-candidate - itemsets, since the efficiency of intersection degrades horribly as the length of the tid set rises. As a result, when tid set is big, the algorithm is much less effective than the direct counting approach. As a result, HUOPM's pruning search is ineffective. In order to create a (k + 1)-itemset from two K-itemsets, it is necessary to do an intersection-counting comparison on each member of the tid sets of the two itemsets. It will take a substantial amount of time to determine the connection between the two *tid sets*.

The cross-intersection method is quite inefficient when itemset has a very big tid set but is not HUOI. It produces several potential subset objects, where each subset consists of k items. In this way, HUOPM produces many subsets, severely diminishing the algorithm's performance. In this work, we think about how to minimise the time spent scanning the entire database and coming up with potential candidates. As a result, the SHO algorithm can only store the most frequently used 1-itemsets in RAM.

As an added bonus, SHO offers a comprehensive set-enumeration tree-based pruning approach for 3-itemsets. Because when number of objects in a database grows, the size of the search space grows exponentially. Using n as the number of records in the database, we can say that the search space is 2n. To further increase the efficiency of method, we offer a unique data structure *Item-list*. The enlarged items' data is stored in an item-list, a two-way stack data structure. The enlarged items, which are 1-itemsets, will be used in a breadth-first search to produce itemsets of size 2 or greater. As a means of narrowing the search space, we build an Item-list for HUOI mining within a partition. The cardinality sequence and the operand sequence make up the whole thing.

The steps involved in creating an Item-list are as follows. Items in a division start off often occurring and sorted in descending support order. The first item is placed in cardinality order on the left, accompanied by the other 1-itemsets, which are placed in operand order on the right. The operand sequence stores items from lowest support to greatest support, with the lowest supported items at the end of the list (the tail). After that, we sequentially insert items from the operand sequence into the Item-list, followed by an item from the cardinality sequence. In conclusion, we reverse-engineer the Item-list and use it to mine itemsets and construct a set-enumeration tree.

3 Results and Discussion

EBPMiner's efficacy was evaluated experimentally on a number of benchmark datasets. The section examines the execution time, memory use, and scalability. Extensive experiments are implemented in JAVA using Open JDK-15.0.2-1 to test the performance of the proposed SHO method. The tests are performed on a Lenovo Vostro square patch with a 3.07 GHz Core i7-10700 processor, 8GB of RAM, and a 64-bit version of Windows 10. The project makes use of a variety of actual and synthetic datasets of varying sorts. We use the freely available Retail and Kosarak datasets included in the SFMP Data Mining Library for our practical examples.

Here are some explanations: When compared to other algorithms, HUOPM, the newest one for high utility saturation itemsets mining, performs better. Vertical mining method BA and there are many different kinds of representative algorithms, but two of them are the HUOPM and the BA. The suffix-based partitioning pre-computation technique has two steps: itemset segmentation and vertical-format transformation. This takes 6 seconds to run when the quantity of communications is 100K. With an increase of 100,000,000 transactions, suffix-based partitioning takes 1610,282 seconds to complete. Number of node based performance has been analysed as inFigure 2 .

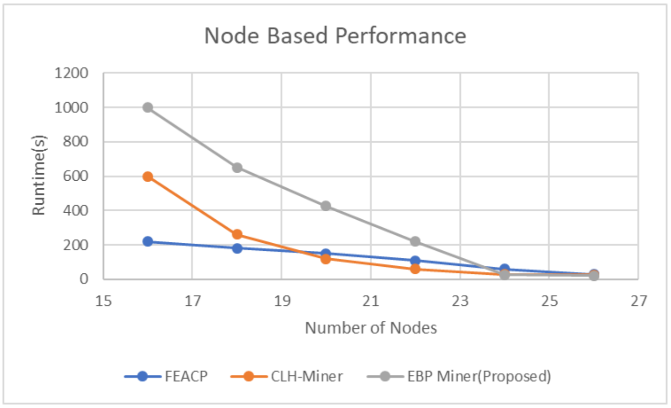

Fig 2. Node based performance

Pre-computation takes a while to execute when there are several transactions. SHO and BA have higher I/O costs than HUOPM. The numbers of bytes obtained from in order, 20KB, 60KB, and 40KB when *n* is small. However, as the dataset expands, the discrepancies in the total amount of bytes recovered become more pronounced. To retrieve 335 745 277 bytes from SHO, 704 801 236 bytes from BA, and 207 466 095 bytes from HOUPM when the number of transactions is 107 is as follows. But suffix-based partitioning is done just once, at the start of the algorithm, so keep that in mind. After that, the partitioning result can be used in the subsequent mining without any additional processing. Minimum Utility based performance results are listed in Table 1.

Table 1. Minimum Utility based performance

	Time Duration(s)		
Minutil (%)	FEACP	CLH-Miner	EBP Miner (Proposed)
20	0.257	0.635	0.115
40	0.254	0.398	0.096
60	0.524	0.486	0.188
80	0.685	0.547	0.204
100	0.788	0.657	0.274

While the SHO algorithm's efficiency is superior to that of HUOPM and BA, it does require more time up front due to the need for pre-computation. With SHO, you may expect execution times that are up to three times as rapid as those of HUOPM and BA for n = 500U t, 1000U t, 5000U t, 10,000U t, respectively, whereas BA is just 30.1 times, 279.6 occasions, 434.3 times, and 810.8 times faster in these same scenarios. There is empirical evidence that pre-computation may boost algorithmic efficiency; hence, we believe it is highly beneficial to perform pre-computation in real-world mining tasks. It's correct that SHO and BA need more room than HUOPM does when n gets really big. The memory complexity values of various approaches with the proposed approach has been shown in Figure 3.

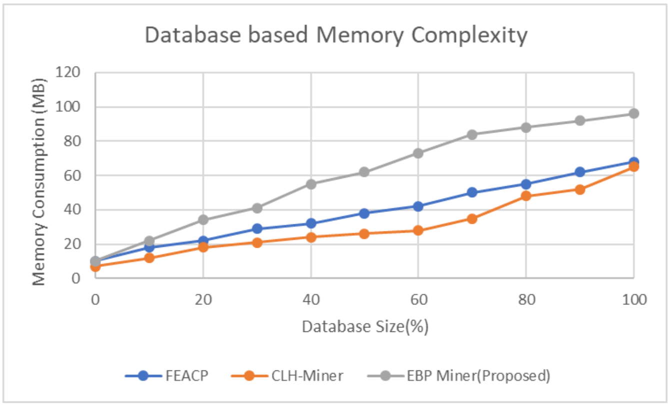

Fig 3. Database based Memory complexity

Partitioning by suffixes retains more occurrences than the initial transaction database did since a single suffix-based partition might encompass items. The suffix-based partitions, on the other hand, include significantly fewer but shorter transactions. The limitations of physical memory are no longer an issue. Since discs are increasing in size and decreasing in price, most modern computers have enough of space. Because of this, we prioritise temporal considerations over spatial ones when assessing algorithms. Here, SHO travels at a suitable bigger space overhead for a significantly faster radio. The following studies show that SHO can significantly outperform BA and SHO by several orders of magnitude.

Profoundly, both SHO and BA are based on the idea that separating the large dataset up into multiple compartments in accordance with the suffix-items. Based on a fixed volume of transactions as well as a fixed total transaction size, a significantly larger value of *d* appears to mean more fragments with fewer and shorter transactions overall. When there are too many things to sort, the resulting partitions will be chaotic. Consequently, it will cause extra work and storage that isn't essential, leading to wasted resources. The effectiveness of combination of two segmentation is low, and the approach cannot partition the dataset across divisions independently sufficient, thus the computational complexity in the frequent items mining stage stays extremely significant whenever the number of objects is too small. Utilization oriented memory occupancy has been shown in Figure 4.

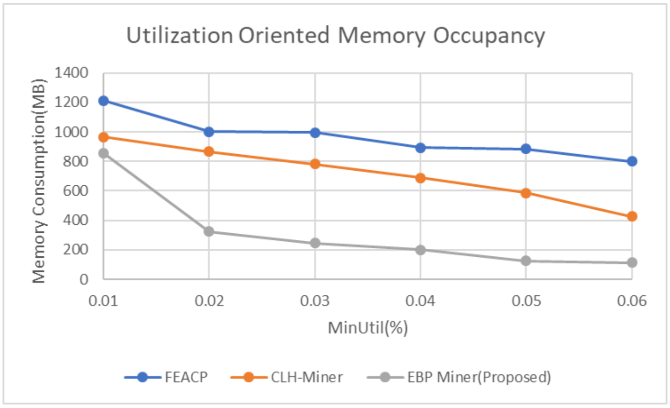

Fig 4. Utilization oriented memory occupancy

Consequently, the majority of things affects SHO's effectiveness; keeping the number of components low to medium can improve the computation performance. As stated above, a large number of studies showed that SHO was superior to BA and HOUPM, but we still needed statistical proof of our findings, therefore we present it here. Both the Friedman test and the Nemenyi post-hoc test is used for the verification. First, we use the Friedman test to see if there is a statistically significant difference in performance between SHO, BA, and HUOPM. Because Java is an object - oriented language executed in a virtual machine, it is not uncommon for memory usage to fluctuate slightly.

Like ULB-Miner, EBP-Miner uses around the average amount of RAM for a mining algorithm. It's because the utility-list buffer is foundational to both methods. The memory usage of EBP-Miner is slightly higher than that of ULB-Miner because it stores extra partition information. As a whole, EBP-memory Miner's footprint is manageable. We performed the algorithms using 10% increments of the Retail and Accidents databases' transaction volumes, ranging from 10% to 100% of each database, to gauge their scalability. For these datasets, the minutil parameter was set as 0.01% and 10%. Memory consumption is further decreased by values.

Because of the need to keep track of partitions and a tree structure. It has been found that EBP combined with pruning methods is the most recollection version, while BUL does not do well in some cases. Due to the extensive duplication of bitsets and hash maps required by BUL, its performance is inferior to that of the original technique. It's important to remember that in EBP, instead of making copies, Support and Tidset are calculated with the long primitive type. Partition based mining memory utilization has been depicted inFigure 5.

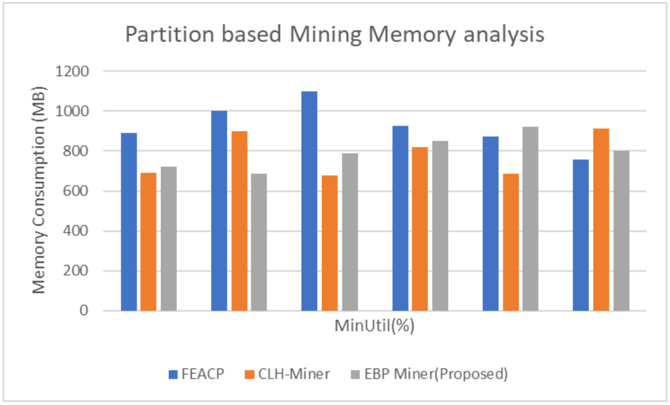

Fig 5. Partition based mining memory utilization

EBP-Miner, which is based on utility lists, offers the most significant speedup. The performance of EBP-Miner, which is based on partitioned utility lists, and EBPMiner, which is based on IHUP trees, are virtually identical. It is observed that EBP and the pruning techniques are highly powerful in some circumstances. Potentially ten times faster. As a result, the suggested approach can be as quick as 10 times more efficient than the standard methodology. Pruning techniques can be beneficial in enhancing memory consumption, albeit occasionally reducing runtime.

4 Conclusion

In this article, we presented BCC, a new approach to building. As a result, the join operation of common utility-list based algorithms may be enhanced by using bitwise operations. The intricacy of BCC was compared to that of cutting-edge building methods, and found to be superior. Previously, a bit division utilitylist architecture known as BCC was offered, but now, a new structure known as EBP has been presented to enhance BCC's performance. Multiple "partitions" hold various subsets of the database. These concepts and many pruning techniques were then combined to create a novel high utility itemset mining algorithm dubbed EBPMiner. It was proven that its join operation has constant time complexity regardless of the size of the database. The experimental findings support the recommended approach performs better than both the original approach and the current state-of-the-art methods. The proposed system achieves 390s runtime and utilization value of 90.25% which are outperformed the existing methods. Also, the approach has proven 20% lesser memory complexity than of the existing algorithms. We hope to develop the suggested approach further so that it may be used in a decentralized system and to handle data streams in the future. Other improvements for runtime and memory will also be designed.

References

1) Reddy A, Murali MH, Prasad K. High utility item-set mining from retail market data stream with various discount strategies using EGUI-tree. *Journal of Ambient Intelligence and Humanized Computing*. 2021;p. 1–12. Available from: https://doi.org/10.1007/s12652-021-03341-3.
2) Baek Y, Yun U, Kim H, Kim J, Vo B, Truong T, et al. Approximate high utility itemset mining in noisy environments. *Knowledge-Based Systems*. 2021;212:106596. Available from: https://doi.org/10.1016/j.knosys.2020.106596.
3) Han X, Liu X, Li J, Gao H. Efficient top-k high utility itemset mining on massive data. *Information Sciences*. 2021;557:382–406. Available from: https://doi.org/10.1016/j.ins.2020.08.028.
4) Qu JF, Fournier-Viger P, Liu M, Hang B, Wang F. Mining high utility itemsets using extended chain structure and utility machine. *Knowledge-Based Systems*. 2020;208:106457. Available from: https://doi.org/10.1016/j.knosys.2020.106457.
5) Sathyavani D, Sharmila D. Retraction Note to: An improved memory adaptive up-growth to mine high utility itemsets from large transaction databases. *Journal of Ambient Intelligence and Humanized Computing*. 2021;12(3):3841–3850. Available from: https://doi.org/10.1007/s12652-022-04039-w.
6) Sethi KK, Ramesh DC, Trivedi MC. A Spark-based high utility itemset mining with multiple external utilities. *Cluster Computing*. 2022;25(2):889–909. Available from: https://doi.org/10.1007/s10586-021-03442-w.
7) Song W, Li J. Discovering High Utility Itemsets Using Set-Based Particle Swarm Optimization. *Advanced Data Mining and Applications*. 2020;p. 38–53. Available from: https://doi.org/10.1007/978-3-030-65390-3_4.
8) Shen W, Zhang C, Fang W, Zhang X, Zhan ZHCW, Lin JCW. Efficient High-utility Itemset Mining Based on a Novel Data Structure. *2021 IEEE International Smart Cities Conference (ISC2)*. 2021;2:1–6. Available from: https://doi.org/10.1109/ISC253183.2021.9562788.
9) Lin JCWCW, Djenouri Y, Srivastava G, Yun U, Fournier-Viger P. A predictive GA-based model for closed high-utility itemset mining. *Applied Soft Computing*. 2021;108:107422. Available from: https://doi.org/10.1016/j.asoc.2021.107422.
10) Krishna GJ, Ravi V. High utility itemset mining using binary differential evolution: An application to customer segmentation. *Expert Systems with Applications*. 2021;181:115122. Available from: https://doi.org/10.1016/j.eswa.2021.115122.