Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 21(2): 117-129, Jun. 2023 Regular paper

Multi-Sized cumulative Summary Structure Driven Light Weight in Frequent Closed Itemset Mining to Increase High Utility

Siva S[1]* and Shilpa Chaudhari[2]

[1]Department of Computer Science and Applications, Reva University, Bangalore 560064, India
[2]Department of Computer Science and Engineering, MS Ramaiah Institute of Technology, Bangalore 560054, India

Abstract

High-utility itemset mining (HIUM) has emerged as a key data-mining paradigm for object-of-interest identification and recommendation systems that serve as frequent itemset identification tools, product or service recommendation systems, etc. Recently, it has gained widespread attention owing to its increasing role in business intelligence, top-N recommendation, and other enterprise solutions. Despite the increasing significance and the inability to provide swift and more accurate predictions, most at-hand solutions, including frequent itemset mining, HUIM, and high average- and fast high-utility itemset mining, are limited to coping with real-time enterprise demands. Moreover, complex computations and high memory exhaustion limit their scalability as enterprise solutions. To address these limitations, this study proposes a model to extract high-utility frequent closed itemsets based on an improved cumulative summary list structure (CSLFC-HUIM) to reduce an optimal set of candidate items in the search space. Moreover, it employs the lift score as the minimum threshold, called the cumulative utility threshold, to prune the search space optimal set of itemsets in a nested-list structure that improves computational time, costs, and memory exhaustion. Simulations over different datasets revealed that the proposed CSLFC-HUIM model outperforms other existing methods, such as closed- and frequent closed-HUIM variants, in terms of execution time and memory consumption, making it suitable for different mined items and allied intelligence of business goals.

Index Terms: Business Intelligence, Cumulative Summary List-Structure, Frequent Closed High-Utility Itemset Mining

I. INTRODUCTION

The rapid growth in software technologies, the Internet, decentralized computing, and low-cost hardware Fhas broadened the horizon of analytics solutions for real-time decision-making. Advanced analytics solutions have become central entities for business intelligence and enterprise solutions, particularly for product recommendations, market segmentation, and supply chain management. It also helps in future demand analysis and prediction, which shapes businesses for proactive decision-making [1-4]. Pattern mining or high-utility itemset mining (HIUM) methods are commonly used in market basket analysis, business intelligence, and recommendation systems [5]. Functionally, HIUM and related pattern-mining approaches involve the identification of vital features, cues, or data objects, thus enabling target segmentation and prediction [6]. In the real world, the patterns can be periodic patterns, sequential patterns, and frequent itemsets. However, most business intelligence solutions employ frequent itemset patterns that enable the segmentation of the top demanding products, probable demands and items, inter-item associations, and so on for further prediction and decision-making. Frequent itemset mining methods have been extensively applied for frequently occurring item segmenta-

Received 01 October 2022, **Revised** 21 February 2023, **Accepted** 01 March 2023
***Corresponding Author** Siva S (E-mail: sivaraju.reva@gmail.com)
Department of Computer Science and Applications, Reva University, Bangalore 560064, India

Open Access **https://doi.org/10.56977/jicce.2023.21.2.117** print ISSN: 2234-8255 online ISSN: 2234-8883

tion over a large transaction dataset [6-8], thus, identifying frequent items with frequencies greater than a defined threshold (e.g., support-value) where prediction decisions are based [9]. Among the major pattern-mining approaches, methods such as Apriori [6-7] were developed for frequent itemset identification by performing (iterative) level-wise searching. These approaches apply the downward bottom closure (DBC) function, also known as the Apriori property, to prune the insignificant itemsets; therefore, it identifies only those items with high frequency to be labeled as high-frequency items. However, the iterative search method and large computation force them to undergo delays and memory exhaustion, making them unsuitable for real-time enterprise solutions. However, in later efforts, the authors proposed different improved solutions, including ECLAT [2], FP-Growth [3-10], and HMine [11], where the focus was on improving the data structure and pruning mechanism to achieve a higher accuracy and lower computation time. However, in these methods, factors such as transaction and its volume, copurchased items and their frequency, and items with high-profit value can affect value-oriented pattern mining and, hence, can confine their efficacy [12]. To address these limitations, high-utility itemset mining (HIUM) [13-14] methods have been developed, which use both volume and its unit profit to improve accuracy, scalability, and time efficacy. The HIUM method applies a utility factor that signifies the total profit of an itemset to identify a set of high-utility items. Numerous state-of-the-art HIUM algorithms [12-15] generate a large volume of candidate itemsets and impose computational costs and delays. However, pruning insignificant itemsets (i.e., items with a low frequency) can reduce the search space and improve performance [20-24]. Unlike the aforementioned DBC-based pruning methods, recent authors [20-21] have found that transaction-weighted utility (TWU) can improve the search space without added computation. In this function, TWU measures the value of the threshold of an upper (utmost) itemset to perform pruning. Numerous efforts have been made towards two-phase implementation [16-19] and single-phase models [20-21] for HIUM, where the first step identifies highly correlated itemsets, followed by a suitability analysis as the HUI in the second step. Unfortunately, owing to its iterative data-scanning nature, Apriori suffers from deficiency, thereby imposing a cost of computation [20]. To improve this, one-phase methods were developed [20] using a list structure, often called a utility list, to identify high-utility items. However, the utility value being proportional to the length of the itemset increases the computational costs for large data. Subsequently, the average-utility method was applied to minimize the influence of item length on high-utility itemset estimation [22-24]. Unlike support value-based mining, using a probability factor can improve HUIM performance [25]. Thus, the list of high-utility items can be improved, and items that are more profitable and in demand can be proposed.

This study proposes a reliable, lightweight frequent closed high-utility itemset mining model driven by cumulative summary list structures (CSLFC-HUIM). The proposed model aims to determine an improved utility list structure with the most significant and frequent itemsets throughout the search space. It uses a stringent threshold measure to limit the number of candidate itemsets in the search space, thereby reducing computing time and memory fatigue. Unlike classical HUIM methods, in which the authors use the minimum support threshold to filter insignificant items, the proposed CSLFC-HUIM model employs the lift value as a support threshold to perform pruning in a nested-list structure. This value is derived from the support value, conviction, and confidence, signifying the likelihood of an item frequently occuring over transactions. The structure of the nested list was improved using the abovementioned lifting measure as a support threshold, called cumulative support, and the corresponding cumulative utility. (e.g., cumulative summary list structure or simply, CULS) to retain only the highly significant items. This helped reduce computational time and memory exhaustion. Simulation results over different benchmark datasets under various threshold conditions revealed that the proposed CSLFC-HUIM model outperformed other methods, including closed-HUIM and frequent closed-HUIM variants. CSLFC-HUIM was found to be more efficient in terms of execution time and memory consumption and is a potential solution for many types of pattern mining or business intelligence tasks.

II. SYSTEM MODEL AND METHODS

Frequent itemset-mining methods include ECLAT [2], FP-Growth [3], Apriori [4], and HMine [11]. However, these methods fail to guarantee time efficiency and accuracy. The authors [6] applied a level-wise search method to design Apriori for frequent itemset estimation. The involved processes, such as iterative database scanning, are time-consuming and exhaustive. In [3], FP-growth was proposed to inculcate tree structure-based itemset identification. First, it generates an FP-tree structure by traversing the search (data) space and identifying frequent itemsets. However, exceedingly high reliance on the search space size and iterative scanning confine for real-time enterprise solutions. The authors in [11] designed an HMine model similar to FP-Growth but with a pointer-based hyperlink to represent items. By contrast, ECLAT [2] was designed as an improved model constituting a vertical layout of the database called "Transaction ID list" to alleviate iterative database scans. In this previous study, the authors applied the support count information for each itemset for comparison with the Transaction ID list to find the frequent itemset(s). Unfortunately, merely applying a support count cannot yield an optimal solution because of the increased search space over a large input dataset. Similar limitations have been observed in other frequent itemset mining methods [6-26-29] that failed to guarantee optimality, particularly in terms of accuracy with minimum time and memory utilization. HIUM methods [13-14] were later designed to identify significant patterns by exploiting both itemset volume and its profit information. HIUM methods were designed as two-phase [16] and one-phase methods. In the two-phase method, the first phase generates a set of significant itemsets possessing a higher frequency, whereas the later phase estimates the candidate's utility to identify high-utility itemsets. To avoid dependence on the utility value, a transaction-weighted utility (TWU) function was designed [16]. The authors [17] designed a TWU with a flexible upper threshold and high-utility itemset skipping ability to improve performance. They used a two-phase model with a pruning algorithm called the isolated itemset discarding strategy (IIDS) to perform swift data scanning and pruning towards HIUM. Subsequently, various tree-based models, such as IHUP [18], HUP-tree [30], UP-Growth [19], UP-Growth+ [31], MU-Growth [32], and PB [33], were developed. These methods apply FP-Growth [3] to reduce the scanning cost using a tree structure. Unlike two-phase methods, the single-phase method [20] focuses on identifying candidate itemsets with high utility values. HUI-Miner [20] was designed as a one-phase model with a list structure called a utility list (UL) to store the information of high-frequency itemsets. With the improved search space, the pruning efficacy became better. To improve HUI-Miner, the authors [21] reduced the number of joint operations between the utility value and frequency using a pruning method named estimated-utility cooccurrence pruning (EUCP), which mainly relies on a matrix structure called estimated-utility cooccurrence structure (EUCS). EUCP stores the TWU values of two item sets in the EUCS matrix, which are used to prune insignificant items without estimating their utility value. To improve [21], the U-pruning, PU-pruning, and LA-pruning methods were developed [34]. The authors [35] designed the EFIM by applying upper bounds named subtree utility and local utility. To minimize the scanning cost, a transaction merging model was designed in [35]. In [36], HMiner was improved by using utility information storage. Methods such as BAHUI [37], the HUIM-BPSO sign [38], MinHUIs [39], and FHM+ [40] were designed for HIUM; however, they were found to be susceptible to large real-time continuous data. Unlike HIUM methods, high average-utility itemset mining methods use average-utility values to reduce reliance on length constraints. As an evolution from TPAU [22], a two-phase mining model applied the upper bound as average utility upper bound (AUUB) criteria. If the AUUB value of an itemset did not meet the aforementioned criteria, it was discarded as a high-utility item, thereby reducing the search space. However, as a level-wise search method, TPAU [22] can experience higher delays and costs. To improve time efficiency, PBAU [23] was designed as a projection-based method with an indexing structure. With PBAU, an upper-limit-driven prefix is applied to reduce the search space. A tree-based high-average-utility itemset mining method was designed in [41], in which the itemset structure was applied to improve performance. The authors [42] designed HAUI-Growth with a tree structure to minimize unwanted iterative scans. A single-phase HAUI was proposed in [43] using an average utility-based list structure. AUUB is used to prune weak candidates from the search space. The authors [44] designed a HAUI miner called EHAUPM by merging two strictly defined upper thresholds called looser upper-bound utility (LUB) and revised tighter upper bound (RTUB). Despite its ability to reduce the search space, it failed to address the association between different items and the probability of becoming a HUI. In [45], MHAI was designed using the HAI list structure to identify suitable high-utility itemsets. A closed HUI (CHUI) was proposed in [51] to reduce the number of candidate itemsets. To improve the time efficiency, CHUI-Miner [52] was designed as a single-phase solution using the EU-List structure. CHIU-Miner, also known as EFIM-Closed [53], was proposed with two upper thresholds with a forward-backward check facility. It applies local utility and subtree utility thresholds to prune the search space to improve performance [54]. Unfortunately, none of these methods [46-50] can assess the probability of coexisting items for profit.

A. Problem Formulation

A. An itemset X with threshold $MinSup$ can be a frequent itemset if $CumSup > MinSup$, where the cumulative support is an upper threshold signifying the corresponding support value, conviction, confidence, and lift scores. For a high-utility itemset, let $I = \{i_1, i_2, ..., i_m\}$ represent the items (e.g., a transaction). The transaction list $D = \{T_1, T_2, ..., T_n\}$ is the set of transactions. For each transaction $TtID \in D$, let tID be a distinct transaction identifier. Let each item $i \in I$ be connected to two nonnegative numbers and $p(i)$ and $q(i; TtID)$ be the external and internal utilities of a transaction, respectively. Here, the internal utility signifies the volume of each item, whereas the external utility represents the profit for item i. The cumulative utility value of an item is derived using (1).

$$u(i) = p(i)q(i; T_{tID}) \quad (1)$$

B. In sync with (1) and the allied HUIM problem, several definitions are derived. These are expressed as follows:

C. ***Definition I*** *(Utility Value of an Itemset in a Transaction)*

D. For a transaction set D encompasses X itemsets, the utility value of X throughout T_{tID} is estimated according to (2).

$$u(X, T_{tID}) = \sum_{i \in X} u(X, T_{tID}) \tag{2}$$

E. ***Definition 2*** *(Cumulative Utility of an Itemset in Database D)*

F. For X itemset in D, the cumulative utility value is given by (3).

$$u_D(X) = \sum_{T_{tID} \in D} u(X, T_{tID}) \tag{3}$$

G. ***Definition 3*** *(Utility Values of a Distinct Transaction and a Complete Database D)*

H. The utility of a transaction refers to the cumulative utilities of all items, as shown in (4).

$$u(T_{tID}) = \sum_{i \in T_{tID}} u(i, T_{tID}) \tag{4}$$

I. ***Definition 4*** *(Relative Utility of an Itemset)*

J. For database D, the relative utility of an itemset is given by (5).

$$\frac{u_D(X)}{U_D} \tag{5}$$

K. ***Definition 5*** *(High-Utility Itemset Mining)*

L. In the context of HUIM,X can be an HIU only when $u_D(X) \geq MinUtil$, where *MinUtil* refers to the minimum utility threshold.

M. ***Definition 6*** *(Closed High-Utility Itemset)*

N. An itemset X can be a closed HUI only when there exists no superset Z, with $X \subset Z$ and $Sup(X) = Sup(Z)$. X and Z can be HUIs provided that $u_D(X) \geq MinUtil$ and $u_D(Z) \geq minUtil$ [26-27]. $Sup(X)$ and $Sup(Z)$ values are the probabilities indicating that the likelihood of X to become an HUI is the same as that of Z.

O. ***Definition 7*** *(Frequent Closed High-Utility Itemset)*

P. If the support value of itemset X satisfies $Sup(X) \geq MinSup$ while maintaining a value near the utility value $u_D(X) \geq MinUtil$, it is called a CHUI. Unlike classical standalone *MinSup*-sensitive pruning methods, the use of *MinUtil* as the second criterion optimizes the search space and HUIM performance.

Q. ***Definition 8*** *(Probabilistic Frequent Closed High-Utility Itemset)*

This work extends the above definitions and derives a factor called a probabilistic frequent closed high-utility itemset to improve overall efficiency. If the cumulative support value of an itemset X in terms of support value, conviction, confidence, and lift values meets the criteria $CumSup(X) > MinSup$ while retaining a value close to the utility value $u_D(X) \geq MinUtil$, it is called a probabilistic CHUI itemset (P-CHUI). In P-CHUI, $CumSup(X) > MinSup$ can optimally filter the search space to achieve a better performance. In sync with Definition 8, the proposed work applies different probabilistic criteria, such as the support value, confidence, conviction, and lift to estimate $CumSup(X)$, which is later used for pruning the search space. Referring to the above definitions, for D as a transaction database, *MinCumSup* and *MinUtil* are the cumulative support threshold and minimum utility, respectively. Subsequently, it identifies the probabilistic frequent closed high-utility itemset (PFCHUI) in D while fulfilling the criteria of *MinCumSup* and *MinUtil*. To ensure the antimonotonic value of the utility, the cumulative TWU (CTWU) was applied as the upper threshold, which is derived using (6).

Table 1. Transaction utility and CTWU estimation

TtID	T_1	T_2	T_3	T_4	T_5	T_6
***u*(*TtID*)**	38	39	11	25	30	37
Item name	a	b	c	d	e	F
CTWU	143	144	77	118	141	123

Definition 9 *Cumulative Transaction-Weighted Utilization*

CTWU of an itemset X ($CTWU(X)$) can be defined as (6).

$$CTWU(X) = \sum_{T_d \in D \wedge X \subseteq T_d} u(T_d) \tag{6}$$

For the illustrations in Table 1, CTWU of item b can be derived using (7).

$$\begin{aligned} CTWU(b) &= u(T_1) + u(T_2) + u(T_5) + u(T_6) \\ &= 38 + 39 + 30 + 37 \\ &= 144 \end{aligned} \tag{7}$$

In (7), u refers to the cumulative utilization. The transaction utilities and CTWU for each item are listed in Table 1.

To apply PFCHUI, CULS is applied, which is followed by the storage of the residual utility values of the items in CULS for filtering. Here, CULS stores item information in the form of (t_{ID}; *iutil*; *rutil*) for each tuple. In CULS,t_{ID} is the transaction identifier, *iutil* is the utility value, and *rutil* is the residual utility value of an itemset in D. CULS is formed by performing the intersection of the utility lists and corresponding probabilistic subsets without scanning the original database. *CumUtil* of an itemset is measured by adding all utility values (i.e., *iutil*) of the itemset in each transaction. If *iutil* and *rutil* values of an itemset are lower than the *MinUtil* value, the itemset is labeled as insignificant. Four probabilistic measures, including support value, conviction, confidence, and lift, were used as the probabilistic utility measures, where the criteria *CumUtility* > *MinUtil* helped refine the search space to improve performance in comparison to the classical TWU-based methods. The addition of *iutil* and *rutil* can improve pruning compared with state-of-the-art TWU methods but at the cost of additional calculations (e.g., conviction, confidence, lift, and cumulative support estimation). To address this, TWU applies a utility list to prune the data space and therefore appends the remaining utility values of an itemset. This study applies *CumUtility* to further reduce insignificant itemsets in *D*. It estimates *CumUtility* as per Definition 10.

Definition 10 *Cumulative Utility*

The *CumUtility* of item *X* in transaction T_{tID} is given by (8), which is estimated using (9).

$$CumUtility(X, T_{tID}) = \begin{cases} u(X, T_{tID}) + ru(X, T_{tID}) & if\ ru(X, T_{tid}) \neq 0 \\ 0 & if\ ru(X, T_{tid}) = 0 \end{cases} \quad (8)$$

$$CumUtility(X) = \sum_{X \in T_d \in DCumUtility(X, T_{tid})} u(T_d) \quad (9)$$

Let *X* be the itemset; then the supersets can be identified as HUI, conditioned at *CumUtility*(*X*) > *MinUtil*. Otherwise, all superset items are categorized as low-utility items and, hence, are dropped. The proposed model exploits the remaining utility of the itemsets to prune low-utility items. It estimates the addition of *iutil* values in the utility list (for *X* itemset) and $u_D(X)$. If $u_D(X) > MinUtil$, then *X* is defined as an HUI. Subsequently, it applies the sum of $u_D(X)$ and $\Sigma_{tI\text{-}D} ru(X, TtID)$ to assess whether the probabilistically associated itemsets are of high utility.

1) System Model

The overall proposed model is depicted in Fig. 1.

a) Cumulative Utility Summary List (CULS):

Most existing HIUM models use a utility list structure to prune and define HUIs. The utility list of a set of entries in a transactional database can be measured by traversing the utility lists of its subsets without parsing the original database. In contrast, most existing methods perform repetitive intersection functions and contain noncritical elements in the memory. This exhausts the memory and time required by classical methods. Therefore, unlike this approach, classical utility-list models are not optimal for truncating infrequent and insignificant item sets in large transactional databases. Classical methods do not store the associations between itemsets or the probabilistic relationships between items. To address this problem, CULS is applied, which depends on a structure called a utility-list buffer [25]. CULS applies a utility list buffer structure (ULB), followed by its definition. In this study, ULB is designed as a memory-pipeline structure to store the information pertaining to each itemset in the form of (*tID*; *iutil*; *rutil*). In this list structure, *tID*, *iutil*, and *rutil* state the transaction ID, the internal utility value, and the remaining utility value, respectively, for an itemset in a transaction. The utility list for each item is stored in different ULB data segments. With the aforementioned ULB list structure, the proposed model applies CULS to swiftly access the itemset information. Here, CULS acts as an index segment to locate and access itemset information. In *D*, the index segment CULS of an itemset *X* (e.g., CULS(*X*)) can be defined as a tuple possessing a structure (10).

$$CULS(X) = \begin{pmatrix} X; StartLoc; EndLoc; \\ SumIUtil; SumRUtil; \\ CumSup(X) \end{pmatrix} \quad (10)$$

In (10), *StartLoc* and *EndLoc* refer to the start and end indices, respectively, of an index segment in the ULB structure. *SumIUtil* is the sum of the internal utility values *iUtil*, and the sum of the remaining utility values (in the utility profile of itemset *X*) is given as *SumRUtil*. The parameter *Sup*(*X*) is the support value of an itemset *X*. With a prespecified itemset *X* and the corresponding CULS structure, the allied support threshold is estimated using (11). This approach can help swiftly estimate the support values of each itemset without requiring repeated database scanning. Consequently, this can make the model more time and memory efficient.

$$Sup(X) = EndLoc - StartLoc \quad (11)$$

b) CULS Precheck Model:

The use of *CunUtil* can improve pruning; however, there can be several itemset candidates for further mining, which can impact performance. In CHUIM models [57-59], two distinct itemsets are assessed to determine whether they possess any subsumption association by performing a comparison among transaction *tID* sets. It is vital to estimate the close high-utility itemsets; however, executing the overall process can be time-consuming and complex. This can become more complex if it requires estimating the relationship between itemsets and previously mined close HUIs. To address these issues and reduce insignificant itemsets, this study improved the subsume check using a precheck. Let the itemsets be *X* and *Z*. The latter can be defined as an itemset *X* only when $X \subset Z$ and $Sup(X) = Sup(Z)$. When defining a high-utility itemset, the aggregate relationship between the

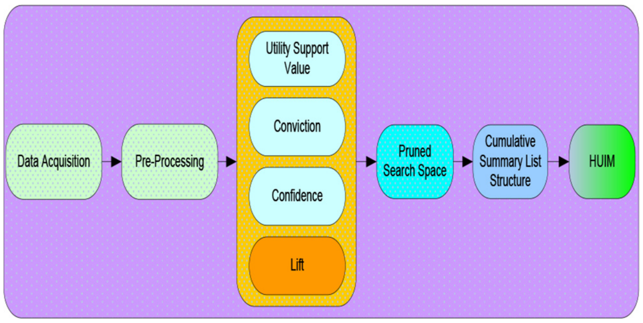


Fig. 1. Proposed model.

itemsets occurs in decreasing order of length**.** Item set *ab* is generated after item *a*, whereas itemset *abc* is generated as a subsequent itemset *ab*. Therefore, it is possible to examine whether an itemset possesses a possible subset during its generation. Therefore, only important itemsets are retained, with the infrequent items deleted. Only meaningful item sets are retained for HUI extraction.

c) Cumulative Summary List Structure (CSLS):

This work emphasizes performing frequent closed HUIM, signifying that not only is the itemset's cumulative utility supposed to be higher than *MinUtil* but also that its allied *CumSup* is supposed to be higher than the *MinSup* value. To achieve this, the itemset candidates obtained in the previous section are split into multiple data blocks with respect to their corresponding support values. Because the itemsets in a block possess equal support values, our proposed CSLFC-HUIM model requires estimating frequent CHUI itemsets in each data block by applying the abovementioned subsumption relationship. CSLFC-HUIM applied subsumption relationship information among itemsets in each block to identify probabilistic frequent CHUIs. The proposed model applied a nested list structure to store the itemset candidates. A targeted CSLS structure was designed using the modality described in (12), where *CumSupport* stores the cumulative support values of each itemset, and *List⟨List⟨Itemsetlength⟩⟩* stores the itemset candidates pertaining to each *CumSupport* value. The itemsets pertaining to each *CumSupport* are stored in ascending order of length.

$$Map2EachSupport = Map\left\langle \begin{matrix} CumSupport, \\ List\langle List\langle ItemsetLength\rangle\rangle \end{matrix} \right\rangle \quad (12)$$

d) Implementation:

To implement the model, function (12) (i.e., *Map2EachSupport*) estimates the set of itemsets from the CSLFC-HUIM candidate list, along with their corresponding length and cumulative support values. These values are appended to a new list structure to construct a targeted nested list structure, as defined in (12) (i.e., the *Map2EachSupport* structure). Storing these itemsets and their support (i.e., *CumSupport*) or length values, CSLFC-HUIM splits the significant itemsets according to their corresponding *CumSupport* values. Pseudocodes are given in Algorithms I-III.

Algorithm-1 Pseudocode for CSLFC-HUIM mining
Pseudocode-1 CSLFC-HUIM Mining Model
Input: Transaction database *D*, *MinSup*, and *MinUtil*
Output: Targeted probabilistic frequent CHUI
Algorithms:

```
1. Initiate the scanning of the input transaction database
   D, and estimate the CTWU for first itemset.
2. Consider X* as the set of each itemset with CTWU
   ≥ MinUtil and CumSup ≥ MinSup.
3. Sort items in descending order (in X*) per CTWU.
4. Restart scanning database D to utility profiling and
   CULS for each item for x ∈ X*.
5. Form the proposed linked list structure.
6. Execute Search-CSLFC-HUIM (ϕ, CSLS of items ∈
   X*, MinUtil, MinSup).
```

Algorithm-1 Pseudocode for probabilistic high-frequent and high-utility itemsets
Pseudocode-2 Targeted High-Frequent and High-Utility Itemsets
Input: *x*: An itemset in utility profile list; *Z* probabilistic *Z* items pertaining to the *x* itemset; *MinCumUtil* and *MinCumSup*
Output: Targeted probabilistic frequent CHUI

```
1. Initiate a loop.
for (each $x \in ULB$) do
      Boolean $Satisfy_{MinCumUtil} = True$;
            if ($CULS.CumSup(x) < MinCumSup$) then
               Continue;
              If ($u_D(x) < MinCumSup = False;$)
            End if
             Map CSLFC − HUIM$CandidateFor EachSupport$;
      If $\begin{pmatrix} CumUtil(x) > MinCumUtil\ satis \\ ft\ MinCumUtil \end{pmatrix}$ then
      Boolean $isProbabilisticFrequentClosedItem = True$;
    For each ($z \in Z$) do
        if (CSLS (the last item in CULS, the last item in Z)> $MinCumUtil$, then
              if ($TIDset(x) == TIDset(xz)$) then
                 $isProbabilisticFrequentClosedItem = False$;
                 End if
             End if
      If ($isProbabilisticFrequentClosedItem = True$), then
            CSLFC − HUIM$CandidateForEachSupport.add(x)$;
            Construct CULS and CSLS
      End if
      End for
    Else
               For each ($z \in Z$) do
                  If each ($CSLS$(the last item in CULS, the last item in Z)> $MinCumUtil$, then
                     Construct the CSLS.
                  End if
               End for
      End if
            For (each list (say, $DoubleList$) in $PHCHUI CandidateForEachSupport$)
              do
                 $PFCHUI.add(Generate_{ItemPHCHUI})$;
           End for
           End if
       End for
```

Algorithm-3 Pseudocode for generating targeted high-utility itemsets

```
Pseudocode-3 Generate_ItemCSLFC-HUIM
Input: CSLFC-HUIMCandidateForEachSupport
Output: Targeted probabilistic frequent CHUI
$CandidateForEachSupport$;
For (each $DoubleList$ in $PFCHUI$
$CandidateForEachSupport$) do
        For (int $i = 0, j = 1 < DoubleList.size() - 1 \&\& j <$
$DoubleList.size(); i++, j++$) do
                $shortSingleList = DoubleList.get(i)$;
                Long$SingleList = DoubleList.get(j)$;
                For (each itemset $p \in PLs$) do
                        $isCloseditemset = True$;
                        For (each itemset $q \epsilon PL1$) do
                    If ($itemContain(q, p)$) then
                        $isCloseditemset = False$
                        Break;
                    End if
                End for
            If ($isProbabilisticFrequentClosedItem$) then
                $PFCHUI.add(p)$;
            End if
        End for
            If ($DoubleList.size() == 1$) then
                $longSingleList = DoubleList.get(0)$
            End if
            $\mathrm{CSLFC-HUIM}.add(all\ itemsets\ in\ PL1)$;
        End for
    End for
```

III. RESULTS

Multiple benchmark datasets were used to assess the efficacy of the proposed CSLFC-HUIM model. In addition, a comparison was made with different state-of-the-art HIUM methods, where the relative performance was characterized in terms of time and memory consumption. Six datasets were obtained from the SPMF Open-Source Data Mining Library [55] (Table 2). Table 2 presents the data specifications, including the total number of transactions, total items, the average length of transactions, and the type of data (sparse or dense). Dense data encompass several items per transaction, whereas sparse data contain relatively fewer items per transaction. The utility per transaction varies according to the number of items per transaction and its corresponding occurrence throughout the dataset. Accordingly, we examined the performance using different datasets, including both sparse and dense (data) structures. To assess efficacy, the performance was tested over different threshold values, including *MinCumSup* and *MinCumUtil*, for the different datasets.

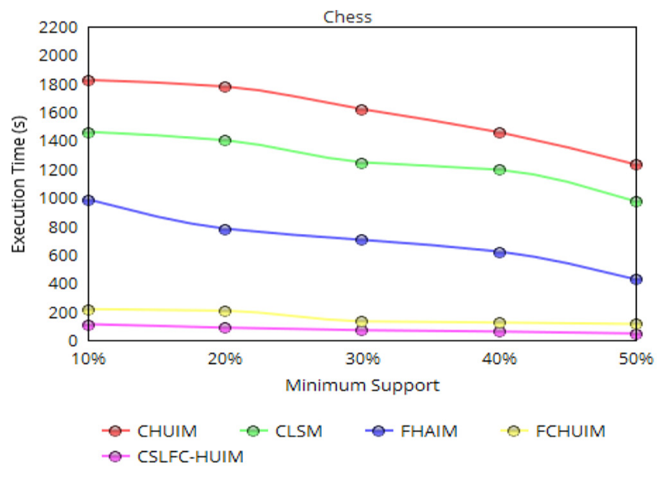
Table 2. Data specifications [55]

Dataset	Transactions	Items	Average length of transaction	Size (MB)	Type of the Data
Chess	3196	75	37	656.6 KB	Dense
Accident	340,183	468	33.8	66.2	Dense
Retail	88,162	16,470	10.3	6.7	Sparse
Chainstore	1,112,949	46,086	7.3	39.1	Sparse
Kosarak	990,002	41,270	8.1	21.6	Sparse
Connect	67,557	129	43	16.9	Dense

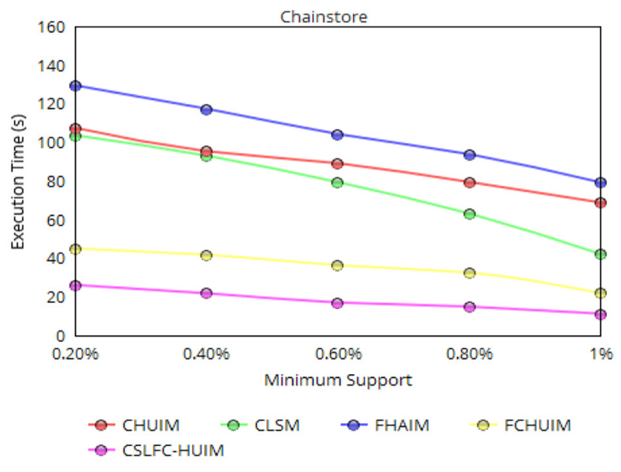
A. Runtime Analysis: Cumulative Utility vs. Runtime

Four state-of-the-art algorithms were considered: closed high-utility itemset mining (CHUI-Miner [56]), CLS-Miner [57], fast high average-utility itemset miner (FHUIM) [58], and frequent and closed HUIM (FCHUIM) [59]. The CHUI-miner (CHUIM) and FCHUIM models were developed based on the closed HUIM [56] principle, where they focus mainly on improving the utility-list structure. The CSLFC-HUIM model can be considered an extension of these methods. However, unlike in [56-57-59], CSLFC-HUIM uses multiple support values to perform pruning. It applies statistics (probabilistic statistical values), including conviction, confidence, and lift, to design cumulative support values to refine pruning. Because lift is derived as an eventual feature by applying support value, confidence, and conviction, we considered this value at the place of the support value. CSLFC-HUIM uses the lift score as the support value to perform CULS and derive a nested utility list structure. It significantly reduces the candidate itemsets in the list structure, which decisively improves the computation. Fig. 2 shows the execution time analysis over different input datasets for different utility threshold values. The results indicate that the CSLFC-HUIM consumes less time.

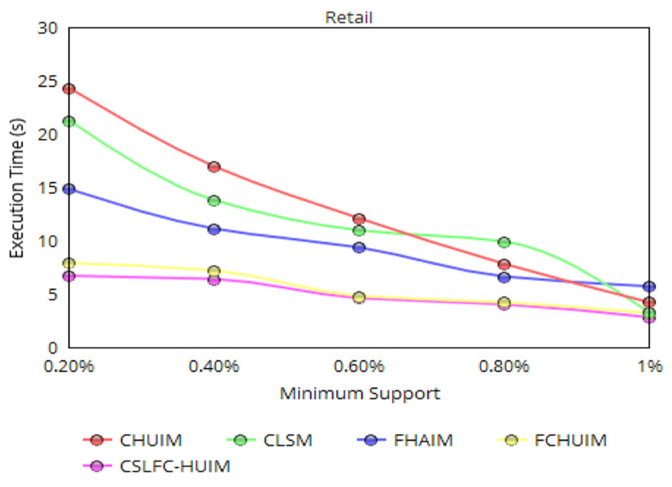
This ability of CSLFC-HUIM improved the search space and computation. Moreover, the nested loop architecture further improved the computational efficacy and time consumption. In contrast, classical approaches, such as FCHUIM and CLSM, which are TWU pruning models, require more time. We varied the utility score for the different test datasets (i.e., Chess, Chainstore, Retail, Accident, Kosarak, and Connect) to assess the relative performance. The results confirm that the proposed CSLFC-HUIM model outperforms other state-of-the-art methods. For the chess data, where the utility threshold was varied from 350000 to 550000, CSLFC-HUIM performed better than other state-of-the-art models, such as CHUIM, CLSM, and FHAIM; however, the performance of FCHUIM was found to be close to that of the proposed CSLFC-HUIM model. This is because, similar to the proposed CSLFC-HUIM model, FCHUIM applies a support threshold-driven nested-list structure for pruning, which improves time efficiency. The simulation confirms that the CSLFC-HUIM model outperforms other state-of-the-art methods, whereas FCHUIM, CLSM, FHAIM, and CHUIM perform in decreasing order. CHUIM is a two-phase method



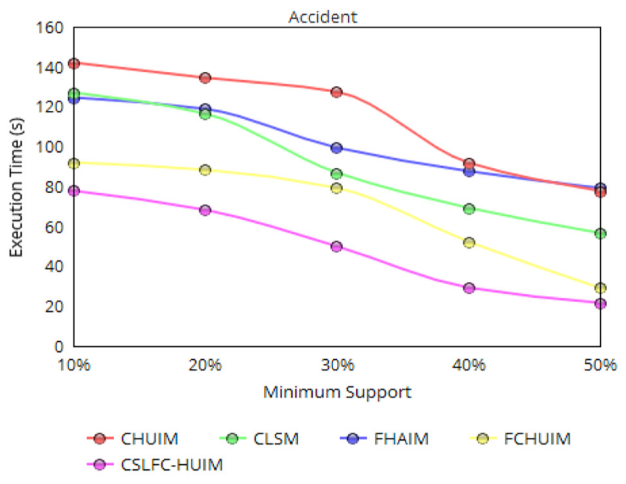
(a) Chess



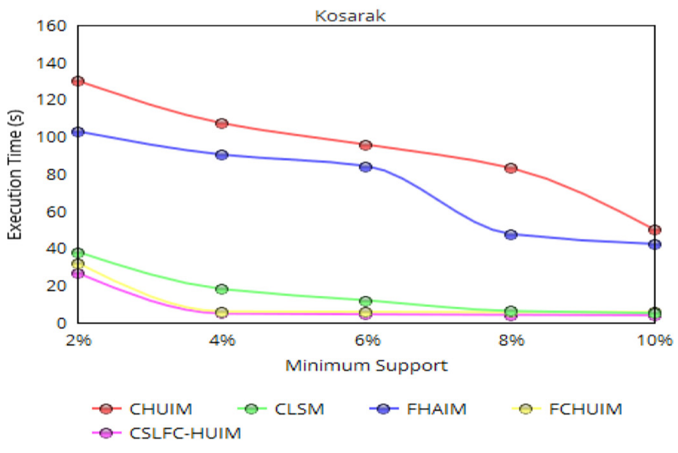
(b) Chainstore



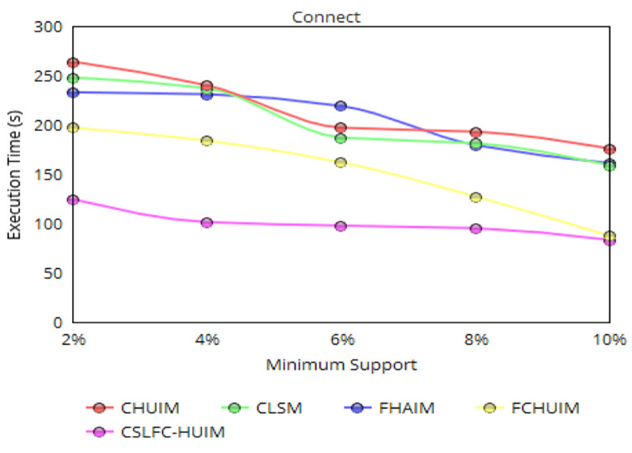
(c) Retail



(d) Accident



(e) Kosarak



(f) Connect

Fig. 2. Execution time analysis over different utility threshold values for different input datasets.

that consumes more time than other methods. In contrast, FHAIM focuses on applying a strict upper bound in terms of the support threshold, which also reduces time efficiency. This is because it cannot avoid the presence of itemsets that have support values lower than the defined threshold. The presence of residual itemsets reduces the performance (Fig. 2). In CSLFC-HUIM, increasing the utility threshold decreases time consumption owing to reduced residual itemsets in the search space and, hence, enhances performance.

B. Run-Time Analysis: Support Threshold vs. Run-time

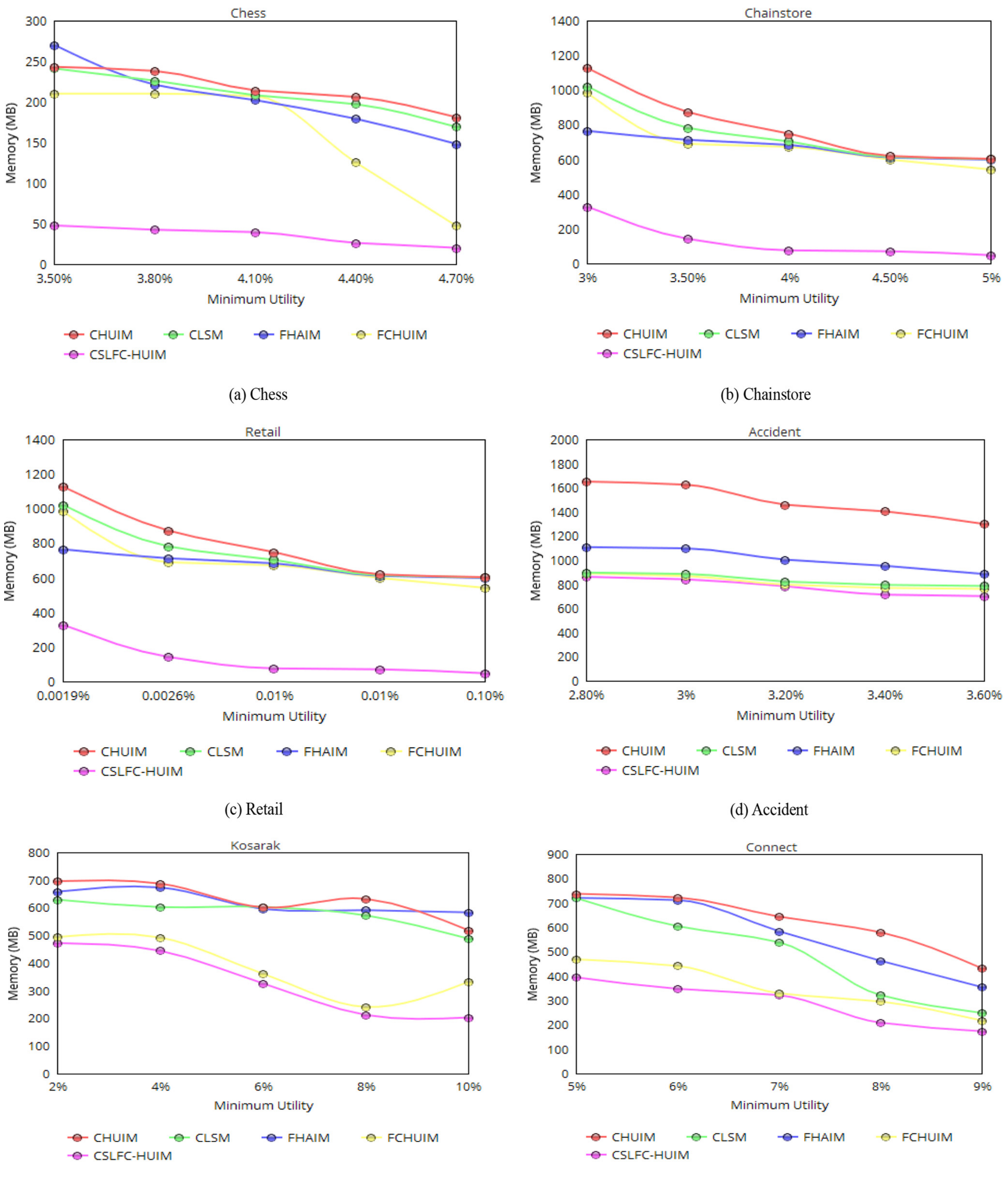
Because increasing the support threshold helps a model reduce the itemsets in the list structure (in this case, the CULS nested list structure), thereby decreasing the computational time, we varied the support threshold for different inputs in percentiles. For instance, in the chess data, we varied the support threshold from 10 to 50% of the target itemsets. The simulation results (Table 3) revealed that the proposed CSLFC-HUIM model performs better than other state-of-the-art methods. The reduction in the residual itemsets in the CULS nested list structure helped reduce the execution time. The results reveal CSLFC-HUIM performs almost four times faster than the classical CHUIM methods. Compared to CLS-Miner, CSLFC-HUIM performs computations almost three times faster. The chainstore dataset carried a total of 1,112,949 transactions with a significantly large number of items (i.e., 46,086), and the different algorithms were simulated with 0.20, 0.40, 0.60, 0.80, and 1% of the item sets. The proposed CSLFC-HUIM model requires an average of 18.2 seconds to process these data for different target itemset batches. In contrast, FCHUIM, CLSM, FHAIM, and CHUIM take 35.5 seconds, 76.2 seconds, 104.7 seconds, and 88 seconds, respectively. For the Chess dataset, the average time consumptions for CHUIM, CLSM, FHAIM, and FCHUIM were 1582.6 seconds, 1255 seconds, 703.2 seconds, and 157.8 seconds, respectively. Compared to these algorithms, CSLFC-HUIM took merely 74.6 seconds, signifying superior performance in HIUM tasks. Similar outputs were found for the Kosarak data, where the existing methods showed higher time consumption (CHUIM (93.18s), CLSM (15.86s), FHAIM (73.4s), and FCHUIM (11.0s)). Compared to these state-of-the-art methods, CSLFC-HUIM took 8.8 seconds, signifying superior time efficacy and exhaustion towards HUIM.

IV. DISCUSSION AND CONCLUSIONS

The average memory utilization by CHUIM, CLSM, FHAIM, and FCHUIM algorithms was 216.4 MB, 208 MB, 204 MB, and 159.8 MB, respectively. In contrast, the proposed CSLFC-HUIM model consumes merely 35 MB of memory (Fig. 3), confirming its robustness in terms of memory utilization. Interestingly, CSLFC-HUIM performs almost four times lower in terms of memory exhaustion than the recently proposed FCHUIM algorithm and almost 6 times lower than CHUIM, CLS-Miner, and FHAIM algorithms. For the Chainstore data, the memory utilization measurement shows that the state-of-the-art CHUIM, CLSM, FHAIM, and FCHUIM consume 793 MB, 744 MB, 673 MB, and 696 MB of resources, respectively. In contrast, CSLFC-HUIM consumes only 132 MB of average memory over the different utility thresholds. A similar pattern can be found in the retail dataset, where CSLFC-HUIM exhibits almost five times lower memory (154 MB) than the state-of-the-art methods (CHUIM (732 MB), CLSM (686 MB), FHAIM (671 MB), and FCHUIM (576 MB)). For the Accident dataset, the

Table 3. Relative runtime analysis with varying minimum cumulative support value (support threshold) for the different datasets

Data	MinCum-Sup	CHUIM	CLSM	FHAIUM	FCHUI	CSLFC-HUIM
Chess	10%	1825	1461	986	217	111.3
	20%	1779	1402	781	206	87
	30%	1620	1247	703	131	69
	40%	1458	1195	621	123	59.6
	50%	1231	972	425	112	46.2
Chainstore	0.20%	107.4	103.6	129.4	45	26.1
	0.40%	95.3	93.02	117.3	41.8	21.8
	0.60%	89.1	79.3	104.2	36.3	17.02
	0.80%	79.4	63.1	93.8	32.5	14.91
	1%	68.8	42.1	79.2	21.9	11.26
Retail	0.20%	24.31	21.27	14.83	7.9	6.7
	0.40%	16.96	13.8	11.12	7.2	6.4
	0.60%	12.09	10.96	9.36	4.8	4.6
	0.80%	7.82	9.91	6.63	4.21	4.02
	1%	4.21	3.27	5.71	3.2	2.81
Accident	10%	142	127	124.5	92	77.76
	20%	134.4	116.4	118.81	88.3	68.2
	30%	127.4	86.8	99.32	79.2	49.91
	40%	91.8	69.2	87.6	52.2	29.2
	50%	77.7	56.6	79.1	29	21.6
Kosarak	2%	129.91	38	102.8	32	26.62
	4%	107.2	18	90.3	6	4.9
	6%	95.7	12	84.2	5.8	4.47
	8%	83.2	6.2	47.6	5.6	4.2
	10%	49.9	5.1	42.3	5.63	3.98
Connect	2%	264	248	233	197	124.4
	4%	240	237	231	184	101.3
	6%	197	187	219	162	97.76
	8%	193	181.3	179.3	127.3	95.21
	10%	176	159	161	87.3	83.22



(a) Chess

(b) Chainstore

(c) Retail

(d) Accident

(e) Kosarak

(f) Connect

Fig. 3. Relative runtime analysis with varying minimum utility values for different datasets.

memory consumption of CHUIM (1486 MB), CLSM (836 MB), FHAIM (1008 MB), and FCHUIM (816 MB) was higher than that of the proposed CSLFC-HUIM model (779 MB). For the Kosarak dataset, the average memory utilization of CHUIM, CLSM, FHAIM, and FCHUIM were 626 MB, 578 MB, 620 MB, and 384 MB, respectively, whereas CSLFC-HUIM consumed only 331.6 MB, signifying better resource efficiency. Similarly, for the Connect dataset, CHUIM (622 MB), CLSM (486 MB), FHAIM (566 MB), and FCHUIM (350 MB) were found to be resource-exhaustive than the proposed CSLFC-HUIM model (288 MB). The overall results confirm that the proposed CSLFC-HUIM can yield both time and memory efficiency to satisfy major HUIM purposes and enterprise demands.

This study proposed a lightweight high-utility closed-element set discovery model (CSLFC-HUIM) for HUIM model mining tasks. The proposed model focuses on improving the utility list structure with frequent and important item sets in the search space to improve overall computational efficiency. Unlike classical HUIM methods, the proposed model applies a lift measure or a value derived as a cumulative score to perform pruning. The use of the lift measure as a support threshold, called cumulative support, and corresponding cumulative utility values helps improve the nested list named cumulative utility list structure (CULS) to refine the search space and the optimal residual itemsets to reduce computational time and memory exhaustion. The use of the cumulative score as a support value as the threshold realized both closed-HUIM and frequent closed-HUIM. The implementation of the cumulative summary list structure with a pre-check-assisted nested-list structure helped the CSLFC-HUIM model exhibit better computational time and memory utilization. Interestingly, the average runtime required (i.e., time-consumption) by the different HUIM methods showed that the proposed method consumed only 42.71 seconds, whereas the other methods like CHUIM, CLSM, FHAIM, and FCHUI consumed 350.92 seconds, 275.5 seconds, 199.57 seconds, and 71.57 seconds, respectively. This confirms that the proposed CSLFC-HUIM model has exceedingly better performance than other state-of-the-art methods, and the role of improved multiple conditions or thresholds in the form of a lift score helped reduce the search space, while CULS enabled swift computation to perform HUIM. Simulations on different benchmark datasets show that the CSLFC-HUIM is nearly five times more responsive while ensuring significantly lower memory consumption. The relative performance with other state-of-the-art methods, such as CHUIM, FHAIM, CLSM, and FCHUIM, confirms that CSLFC-HUIM exhibits superior performance, where the role of the probabilistic derived lift score as a threshold cannot be ignored. As a matter of fact, CSLFC-HUIM is superior towards HUIM tasks; however, it could not address the need for top-N most frequent itemset identification. Future research will focus on identifying the top-N items to assist enterprises in making proactive supply chains and marketing decisions in the business horizon.

REFERENCES

[1] P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, “A survey of sequential pattern mining,” *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54-77, 2017.

[2] M. J. Zaki, “Scalable algorithms for association mining,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 12, no. 3, pp. 372-390, 2000. DOI: 10.1109/69.846291.

[3] J. Han, J. Pei, and M. Kamber, “*Data mining: concepts and techniques,*” Elsevier, Amsterdam, 2011.

[4] R. Agrawal and R. Srikant, “Mining sequential patterns,” in *Proceedings of the Eleventh International Conference on Data Engineering*, Taipei, Taiwan, pp. 3-14, 1995. DOI: 10.1109/ICDE.1995.380415.

[5] K. K. Sethi and D. Ramesh, “A fast high average-utility itemset mining with efficient tighter upper bounds and novel list structure,” *The Journal of Supercomputing*, vol. 76, no. 12, pp. 10288–10318, Mar. 2020. DOI: 10.1007/s11227-020-03247-5.

[6] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” in *ACM Sigmod Record*, vol. 22, no. 2, pp. 207-216, Jun. 1993. DOI: 10.1145/170035.170072.

[7] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, vol. 1215, pp. 487-499, 1994.

[8] P. Fournier-Viger, L. C. W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, “A survey of itemset mining,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 4, Jul. 2017. DOI: 10.1002/widm.1207.

[9] T. Wei, B. Wang, Y. Zhang, K. Hu, Y. Yao, and H. Liu, “FCHUIM: Efficient Frequent and Closed High-Utility Itemsets Mining,” *IEEE Access*, vol. 8, pp. 109928-109939, 2020. DOI: 10.1109/ACCESS.2020.3001975.

[10] G. Grahne and J. Zhu, “Fast algorithms for frequent itemset mining using fp-trees,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 17, no. 10, pp. 1347-1362, Oct. 2005. DOI: 10.1109/TKDE.2005.166.

[11] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, “H-mine: hyper-structure mining of frequent patterns in large databases,” in *Proceedings of the 2001 IEEE International Conference on Data Mining*, San Jose, USA, pp 441-448, 2001. DOI: 10.1109/ICDM.2001.989550.

[12] V. S. Tseng, B. E. Shie, C. W. Wu, and P. S. Yu, “Efficient algorithms for mining high utility itemsets from transactional databases,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772-1786, Aug. 2013. DOI: 10.1109/TKDE.2012.59.

[13] R. Chan, Q. Yang, and Y. D. Shen, “Mining high utility itemsets,” in *Third IEEE International Conference on Data Mining*, Melbourne, USA, pp. 19-26, 2003. DOI: 10.1109/ICDM.2003.1250893.

[14] H. Yao, H. J. Hamilton, and C. J. Butz, “A foundational approach to mining itemset utilities from databases,” in *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 482-486, Apr. 2004. DOI: 10.1137/1.9781611972740.51.

[15] W. Song, Y. Liu, and J. Li, (2014). “BAHUI: Fast and memory efficient mining of high utility itemsets based on bitmap,” *International Journal of Data Warehousing and Mining*, vol. 10, no. 1, pp. 1-15,

2014. DOI: 10.4018/ijdwm.2014010101.

[16] Y. Liu, W. K. Liao, and A. N. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, pp. 689-695, 2005. DOI: 10.1007/11430919_79.

[17] Y. C. Li, J. S. Yeh, and C. C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data and Knowledge Engineering, vol.* 64, no. 1, pp. 198-217, Jan. 2008. DOI: 10.1016/ j.datak.2007.06.009.

[18] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Transactions on Knowledge and Data Engineering, vol.* 21, no. 12, pp. 1708-1721, Dec. 2009. DOI: 10.1109/TKDE. 2009.46.

[19] V. S. Tseng, C. W. Wu, B. E. Shie, and P. S. Yu, "UP-growth: an efficient algorithm for high utility itemset mining," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington DC, USA pp. 253-262, 2010. DOI: 10.1145/1835804.1835839.

[20] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, Maui Hawaii, USA, pp. 55-64, 2012. DOI: 10.1145/2396761.2396773.

[21] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning," in *International Symposium on Methodologies for Intelligent Systems,* Cham, vol. 8502, pp. 83-92, 2014. DOI: 10.1007/ 978-3-319-08326-1_9.

[22] T. P. Hong, C. H. Lee, and S. L. Wang, "Effective utility mining with the measure of average utility," *Expert Systems with Applications,* vol. 38, no. 7, pp. 8259-8265, Jul. 2011. DOI: 10.1016/j.eswa.2011. 01.006.

[23] G. C. Lan, T. P. Hong, and V. S. Tseng, "A projection-based approach for discovering high average utility itemsets," *Journal of Information Science and Engineering,* vol. 28, no. 1, pp. 193-209, 2012.

[24] C. W. Lin, T. P. Hong, and W. H. Lu, "Efficiently mining high average utility itemsets with a tree structure," in *Asian Conference on Intelligent Information and Database Systems*, Hue City, Vietnam, pp. 131-139, 2010. DOI: 10.1007/978-3-642-12145-6_14.

[25] A. Y. Peng, Y. S. Koh, and P. Riddle, "mHUIMiner: A fast high utility itemset mining algorithm for sparse datasets," in *Advances in Knowledge Discovery and Data Mining*, Jeju, South Korea, pp. 196-207, 2017. DOI: 10.1007/978-3-319-57529-2_16.

[26] J. Pei, J. Han, and L. V. Lakshmanan, "Pushing convertible constraints in frequent itemset mining," *Data Mining and Knowledge DiscoveryM* vol. 8, no. 3, pp. 227-252, May 2004. DOI: 10.1023/ B:DAMI.0000023674.74932.4c.

[27] K. K. Sethi and D. Ramesh, "HFIM: a Spark-based hybrid frequent itemset mining algorithm for big data processing," *The Journal of Supercomputing,* vol. 73, no. 8, pp. 3652-3668, Jan. 2017. DOI: 10.1007/s11227-017-1963-4.

[28] G. Pyun, U. Yun, and K. H. Ryu, "Efficient frequent pattern mining based on linear prefix tree," *Knowledge-Based Systems*, vol. 55, pp. 125-139, Jan. 2014. DOI: 10.1016/j.knosys.2013.10.013.

[29] U. Yun, G. Lee, and K.H. Ryu, "Mining maximal frequent patterns by considering weight conditions over data streams," *Knowledge-Based Systems*, vol. 55, pp. 49-65, Jan. 2014. DOI: 10.1016/ j.knosys.2013.10.011.

[30] C. W. Lin, T. P. Hong, and W. H. Lu, "An effective tree structure for mining high utility itemsets," *Expert Systems with Applications*, vol. 38, no. 6, pp. 7419-7424, Jun. 2011. DOI: 10.1016/j.eswa.2010.12. 082.

[31] V. S. Tseng, B. E. Shie, C. W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772-1786, Aug. 2013. DOI: 10.1109/TKDE. 2012.59.

[32] U. Yun, H. Ryang, and K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Systems with Applications,* vol. 41, no. 8, pp. 3861-3878, Jun. 2014. DOI: 10.1016/j.eswa.2013.11.038.

[33] G. C. Lan, T. P. Hong, and V. S. Tseng, "An efficient projection-based indexing approach for mining high utility itemsets," *Knowledge and Information Systems,* vol. 38, no. 1 pp. 85-107, Jan. 2014. DOI: 10.1007/s10115-012-0492-y.

[34] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications,* vol. 42, no. 5, pp. 2371-2381, Apr. 2015. DOI: 10.1016/j.eswa.2014.11.001.

[35] S. Zida, P. Fournier-Viger, J. C. W. Lin, C. W. Wu, and V. S. Tseng, "EFIM: a highly efficient algorithm for high-utility itemset mining," in *Mexican International Conference on Artificial Intelligence*, Cuernavaca, Mexico, pp. 530-546, 2015. DOI: 10.1007/978-3-319-27060-9_44.

[36] S. Krishnamoorthy, "HMiner: efficiently mining high utility itemsets," *Expert Systems with Application*, vol. 90, pp. 168-183, Dec. 2017. DOI: 10.1016/j.eswa.2017.08.028.

[37] W. Song, Y. Liu, and J. Li, "BAHUI: fast and memory efficient mining of high utility itemsets based on bitmap," *International Journal of Data Warehousing and Mining,* vol. 10, no. 1, pp. 1-15, Jan. 2014. DOI: 10.4018/ijdwm.2014010101.

[38] J. C. W. Lin, L. Yang, P. Fournier-Viger, J. M. T. Wu, T. P. Hong, L. S. L. Wang, and J. Zhan, "Mining high utility itemsets based on particle swarm optimization," *Engineering Applications of Artificial Intelligence,* vol. 55, pp. 320-330, Oct. 2016. DOI: 10.1016/ j.engappai.2016.07.006.

[39] P. Fournier-Viger, J. C. W. Lin, C. W. Wu, V. S. Tseng, and U. Faghihi, "Mining minimal high-utility itemsets," in *International Conference on Database and Expert Systems Applications*, Porto, Portugal,, pp. 88-101, 2016. DOI: 10.1007/978-3-319-44403-1_6.

[40] P. Fournier-Viger, J. C. W. Lin, Q. H. Duong, and T. L. Dam, "FHM +: faster high-utility itemset mining using length upper-bound reduction," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Morioka, Japan,, pp. 115-127, 2016. DOI: 10.1007/978-3-319-42007-3_11.

[41] T. Lu, B. Vo, H. T. Nguyen, and T. P. Hong, "A new method for mining high average utility itemsets," in *IFIP International Conference on Computer Information Systems and Industrial Management,* Ho Chi Minh City, Vietnam, pp. 33-42, 2014. DOI: 10.1007/978-3-662-45237-0_5.

[42] C. W. Lin, T. P. Hong, and W. H. Lu, "Efficiently mining high average utility itemsets with a tree structure," in *Asian Conference on Intelligent Information and Database Systems,* Hue City, Vietnam, pp. 131-139, 2010. DOI: 10.1007/978-3-642-12145-6_14.

[43] J. C. W. Lin, T. Li, P. Fournier-Viger, T. P. Hong, J. Zhan, and M. Voznak, "An efficient algorithm to mine high average-utility itemsets," *Advanced Engineering Informatics,* vol. 30, no. 2, pp. 233-243, Apr. 2016. DOI: 10.1016/j.aei.2016.04.002.

[44] J. C. W. Lin, S. Ren, P. Fournier-Viger, and T. P. Hong, "EHAUPM: efficient high average-utility pattern mining with tighter upper bounds," *IEEE Access,* vol. 5, pp. 12927-12940, 2017. DOI: 10.1109/ACCESS.2017.2717438.

[45] U. Yun and D. Kim, "Mining of high average-utility itemsets using novel list structure and pruning strategy," *Future Generation*

Computer Systems, vol. 68, pp. 346-360, Mar. 2017. DOI: 10.1016/j.future.2016.10.027.

[46] J. C. W. Lin, S. Ren, T. P. Fournier-Viger, T. P. Hong, J. H. Su, and B. Vo, “A fast algorithm for mining high average-utility itemsets,” *Applied Intelligence,* vol. 47, no. 2, pp. 331-346. Sep. 2017. DOI: 10.1007/s10489-017-0896-1.

[47] J. C. W. Lin, S. Ren, and P. Fournier-Viger, “MEMU: more efficient algorithm to mine high average utility patterns with multiple minimum average-utility thresholds,” *IEEE Access,* vol. 6, pp. 7593-7609, 2018. DOI: 10.1109/ACCESS.2018.2801261.

[48] J. M. T. Wu, J. C. W. Lin, M. Pirouz, and P. Fournier-Viger, “TUB-HAUPM: tighter upper bound for mining high average-utility patterns,” *IEEE Access,* vol. 6, pp. 18655-18669. DOI: 10.1109/ACCESS.2018.2820740.

[49] T. Truong, H. Duong, B. Le, and P. Fournier-Viger, “Efficient vertical mining of high average-utility itemsets based on novel upper-bounds,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, no. 2, pp. 301-314, Feb. 2018. DOI: 10.1109/TKDE.2018.2833478.

[50] T. Truong, H. Duong, B. Le, P. Fournier-Viger, and U. Yun, “Efficient high average-utility itemset mining using novel vertical weak upper-bounds,” *Knowledge-Based Systems*, vol. 183, p.104847, Nov. 2019. DOI: 10.1016/j.knosys.2019.07.018.

[51] V. S. Tseng, C. W. Wu, P. Fournier-Viger, and P. S. Yu, “Efficient algorithms for mining the concise and lossless representation of high utility itemsets,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 27, no. 3, pp. 726-739, Mar. 2015. DOI: 10.1109/TKDE.2014.2345377.

[52] C. W. Wu, P. Fournier-Viger, J. Y. Gu, and V. S. Tseng, “Mining closed+ high utility itemsets without candidate generation,” in *Proceedings of the 2015 Conference on Technologies and Applications of Artificial. Intelligence,* Tainan, Taiwan, pp. 187-194, 2015. DOI: 10.1109/TAAI.2015.7407089.

[53] P. Fournier-Viger, S. Zida, W. J. C, Lin, C. W. Wu, and V. S. Tseng, “EFIM-closed: Fast and memory efficient discovery of closed high-utility itemsets,” in *Machine Learning and Data Mining in Pattern Recognition,* New York, USA, pp. 199-213, 2016. DOI: 10.1007/978-3-319-41920-6_15.

[54] T.-L. Dam, K. Li, P. Fournier-Viger, and Q. H. Duong, “CLS-Miner: Efficient and effective closed high-utility itemset mining,” *Frontiers of Computer Science,* vol. 13, no. 2, pp. 357-381, 2019. DOI: 10.1007/s11704-016-6245-4.

[55] SPMF: Java open-source data mining library [Internet], Available: http://www.philippe-fournier-viger.com/spmf/.

[56] C. W. Wu, P. Fournier-Viger, J. Y. Gu, and V. S. Tseng, “Mining closed+ high utility itemsets without candidate generation,” in *Proceedings of the Conference on Technologies. and Applications of Artificial. Intelligence*, Tainan, Taiwan, pp. 187-194, 2015. DOI: 10.1109/TAAI.2015.7407089.

[57] T. L. Dam, K. Li, P. Fournier-Viger, and Q. H. Duong, “CLS-Miner: Efficient and effective closed high-utility itemset mining,” *Frontiers of Computer Science,* vol. 13, no. 2, pp. 357-381, Apr. 2019. DOI: 10.1007/s11704-016-6245-4.

[58] K. K. Sethi and D. Ramesh D., “A fast high average-utility itemset mining with efficient tighter upper bounds and novel list structure,” *The Journal of Supercomputing,* vol. 76, no. 12, pp. 10288-10318, Mar. 2020. DOI: 10.1007/s11227-020-03247-5.

[59] T. Wei, B. Wang, Y. Zhang, K. Hu, Y. Yao, and H. Liu, “FCHUIM: Efficient Frequent and Closed High-Utility Itemsets Mining,” *IEEE Access*, pp. 109928-109939, 2020. DOI: 10.1109/ACCESS.2020.3001975.

Siva S

He received his Master of Computer Applications (M.CA) from the University of Madras, India in 2004. He has over 16 years of experience in the field of software research and development. He is currently employed as a senior engineering manager in a company in Bangalore, India, where a healthcare mobile solution is being developed using IoT and AI. Currently, he is pursuing a Ph.D. at REVA University, Bangalore, India. He is a member of IEEE.

Dr. Shilpa Chaudhari

Currently, she is working as an Associate Professor at the Department of CSE, MSRIT,
Bangalore. She has been a technology educator and corporate trainer since 1999. In the last 18 years, she has served in various academic positions in technical institutes in Maharashtra and Karnataka. Her areas of research and teaching include network security,RTOS, computational intelligence, wireless networks, and embedded system development. She is a professional member of the Computer Society of India (CSI), a Life member since 2013, and an IEEE Member #94611631, Bangalore Section.