ISSN 2278-3091
Volume 9, No. 5, September - October 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse78952020.pdf
https://doi.org/10.30534/ijatcse/2020/78952020

A Novel Technique - Absolute High Utility Itemset Mining (AHUIM) Algorithm for Big Data

Dr. Sandeep Dalal[1], Vandna Dahiya[2]
[1]Maharshi Dayanand University, Rohtak, India, sandeepdalal.80@gmail.com
[2]Maharshi Dayanand University, Rohtak, India, vandanadahiya2010@gmail.com

ABSTRACT

Utility itemset mining has manifested as a significant method for mining the data, which finds the itemsets grounded on the utility factors. The utility is defined in terms of quantity and some factor of interest and the algorithm aims to find the itemset, for which the utility is more than a predefined threshold. In the big data era, conventional high utility itemset (HUI) mining techniques are not appropriate for mining the HUIs on an individual system with limited processing capabilities. A distributed algorithm to mine the HUIs for big data has been proposed in this paper called as Absolute High Utility Itemset Mining algorithm or AHUIM algorithm. AHUIM mines the data efficiently on a multi-node cluster using the Apache Spark framework. Various experiments are carried out, which signify that the proposed algorithm outperforms in mining the HUIs in terms of execution time, scalability, storage, etc.

Key words: big data, data mining, distributed computing, Spark, utility mining

1. INTRODUCTION

Frequent Itemset Mining (FIM) is a technique to mine the datasets based on the frequency of occurrence of the items [1]. It discovers the itemsets created on the support and confidence standards of the itemsets. Nevertheless, FIM does not pay attention to the importance of the items. For example, sales of milk and butter might be more frequent in a general store but more profit will be generated from a microwave oven. FIM also considers the presence of the items only once in a particular transaction, whereas, the items with more profit value but with less occurrence may be vital for the users. To overcome these limitations of FIM, high utility itemset mining has surfaced as a superior technique for data mining, that mines the itemsets based on their utility factors. The utility factor of items is composed of two terminologies - internal utility and external utility. The internal utility depends upon the quantity of the item, whereas the external utility can be in the form of cost, weight, profit, effect or any other user objective or preference. Thus utility mining discovers the most useful or valuable items, which are difficult to mine using frequent itemset mining. An itemset is discovered and termed, as a HUI if and only if its utility is greater or equal to a predefined threshold value. The utility values are usually related to various profit factors and are remarkable for a business. For example, in case of retail stores, whereas the FIM only discovers the frequent items bought by the customer, the HUIM discovers the itemsets, which are more profitable for the store also, when they are bought together. These itemsets are then bundled together in the store, proposed to the customers and some discount may also be offered. Recommender systems thus can be developed based on the purchasing history of the customers. Another example of HUIM is click-stream analysis. If a user clicks on various links on the Internet, FIM would recommend all the similar links to the user. But user might be interested in only some of those clicked links. HUIM would recommend the kind of links where user spent more time. Time spent on a particular page is the utility factor here. Other applications areas of utility mining are cross marketing, smarter healthcare, e-commerce, drug designing, etc.

Various HUI techniques have been proposed in recent times for itemset mining but most of them are for small datasets and centered on single machines. The performance of mining starts to degrade with increase in the size of datasets. To mine large datasets, the computing resources of one machine are not enough and put constraints on the scalability of the algorithm. A parallel or distributed framework is required to mine the large datasets. Very few HUIM algorithms have been developed for large datasets that can run on a parallel framework. Vo et al. proposed DTWU-Mining in 2009, which is a distributed algorithm and based on message passing system [2]. The master node distributes the data to the slave nodes, which then compute HUIs locally. But the algorithm deficits the feature of fault-tolerance. Subramanian et al. [3] developed FUM-D in 2013. This algorithm is also based on master-slave architecture but the cost of communication is very high. Lin et al. planned another method called as PHUI-Growth [4] in 2015. PHUI extends the Apriori algorithm with the help of MapReduce framework. Novel strategies like discarding of local unpromising items were proposed. The algorithm scales well with the growing data and can mine the HUIs from huge data but it undergoes from the drawback of manifold scans of the database. Another distributed algorithm is PHUI-Miner [5] that was proposed by Chen et al. in 2016. PHUI-Miner is being implemented on Apache Spark framework using Scala, where the exploration space is alienated equally between the nodes. Sampling and compression techniques have been implemented in this algorithm to mine the data efficiently from large datasets. But these methods also ground the approximation in the mining results. In 2016, Zihayat et al. proposed BigHUSP [6]. BigHUSP is also based on Spark framework. Items with low utility are discarded locally using an overestimated utility model. The pruning strategies in this

algorithm lessen the search space and utility matrix is used to store the utility information. Ashish Tamrakar proposed another Spark-based algorithm EFIM-Par in 2017 [7]. This extends the famous one phase HUIM method EFIM. P-FHM+ was proposed by Sethi et al. [8] in 2018, which extends FHM+ in the parallel system. Itemsets of any desired length can be generated with the help of FHM+. The latest HUIM algorithms for large datasets are pEFIM [9] and PHAUIM [10]. pEFIM was suggested by Nguyen et al. in 2018. It outspreads the EFIM but the framework is different from EFIM-par. pEFIM uses multi-core processor and shared-memory based systems to implement the algorithm. Using depth-first searching approach, various nodes can run independently without overlapping. Sethi et al. developed the algorithm PHAUIM in 2019 where the itemsets are mined by an average factor of utility. It outperforms the other state-of-the-art algorithms for HUI mining of large datasets.

As the big data mining is a novel area, the algorithms in this field are in their early stages of research. The challenges in mining large datasets are different from traditional data mining. This research work contributes the following-

i. A novel high utility itemset mining technique, AHUIM is developed for big datasets that works in the parallel environment of distributed systems.
ii. Pruning strategies for unpromising items.
iii. The technique AHUIM is being assessed for parameters such as storage, scalability, and execution time.

The paper is systematized as follows: Section 2 explains various fundamentals for mining HUI. The new algorithm is being discussed in the next section. Section 4 briefly demonstrates all the experiments conducted for the assessment of the technique. The conclusion and future scope of this research work are followed in the last section.

2. PRELIMINARIES AND PROBLEM STATEMENT

2.1 Distributed Framework

With the arrival of big data, computing resources of one machine are not enough to mine the large datasets. The hardware limits the size of the dataset to be mined based on the processing capability of CPU and storage. The whole scalability of the algorithm also suffers. Efficient and parallel computations are required to mine the big data. Subsequently, to process the huge datasets and extract significant information, a distributed framework is used, which also increases the scalability of the algorithm. There are various distributed frameworks available in the research field such as Apache Hadoop, Apache Spark, etc. [11, 12] Apache Spark has been used for the implementation of the proposed algorithm because of its advantages over other frameworks. It is several times faster than Hadoop as the Hadoop is a disk-based system. Spark does in-memory computations and is used for various data processing applications. The key role is played by a resilient distributed dataset (RDD), which is an assemblage of partitioned-read-only objects. An RDD has the reference to a partitioned object, which is a subdivision of dataset. There is one master node in the Spark environment and multiple worker nodes that perform map-reduce operations. The map operation performs preprocessing on the data for each node and the reduce operation combines the results produced by the mapper.

2.2 Fundamentals of HUI

Consider a transactional dataset DS = $\{T_1, T_2, T_3 \ldots T_n\}$; where each transaction Tr ∈ DS and $1 \leq r \leq n$ has a distinctive identifier, known as Transaction ID. Assume, I be the set of Q unique items; I = $\{I_1, I_2, I_3 \ldots I_Q\}$. Let X be the itemset with k unique items and is called as k-itemset or itemset with length k. Every item in I has a utility factor associated with it which is composed of its quantity, called as internal utility Q and importance factor, called as external utility P. Internal utility is denoted as Q (I_i, Tr) and external utility is denoted as P(I_i). The example dataset with five transactions and seven different items can be seen in the table 1 given below. Table 2 displays the profit value of each item. Some of the important definitions are described briefly in this section.

Table 1: Transactional Dataset - DS

Transaction ID	Transactions (item:quantity)	Transaction-Utility
1)	(t:1), (v:2), (w:2)	15
2)	(t:2), (v:2), (x:1), (z:2)	32
3)	(t:1), (u:1), (w:3), (x:2), (y:3)	37
4)	(u:1), (v:3), (w:2), (x:2)	17
5)	(u:1), (v:3), (x:1), (y:3), (z:2)	43

Table 2: Item-Profit Table

Item	t	u	v	w	x	y	z
Profit value	5	1	2	3	2	6	8

Definition 1**:** *Utility of an item I_i for a transaction Tr*; is expressed by U(I_i, Tr) and expressed as the multiplication of quantity of the item in the transaction and its profit value, i.e., Q(I_i, Tr) * P(I_i).
Example, here the utility for the items t, v and w in transaction T_1 is 1*5 = 5, 2*2 = 4 and 2*3 = 6 respectively.

Definition 2**:** *Utility for an itemset X in a transaction Tr;* is expressed as U(X, Tr) = ∑ U(I_i, Tr) for $I_i \in X$.
Example, for itemset {t, w} in T_1, the utility is U(t, T_1) + U(w, T_1) = 1*5 + 2*3 = 11, itemset {t, v, w} in T_1 = U(t, T_1) + U(v, T_1) + U(w, T_1) = 1*5 + 2*2 + 2*3 = 15.

Definition 3**:** *Transaction Utility;* the transaction utility for a transaction Tr is symbolized as TU(Tr). It is stated as addition of utilities of all the items for that transaction.
Example, for T_1 transaction utility is (1*5+2*2+2*3) = 15. Table 1 shows the TU for all the transactions.

Definition 4**:** *Utility for an Itemset X, U(X) in a dataset DS;* is defined as addition of utilities of the itemset X from every transaction of DS. U(X) = ∑(X, Tr) for Tr ∈ DS ∧ X ⊆ Tr
Example, utility for itemset {v, w} in the above DS is U({v,

w}, T_1) + U({v, w}, T_4) is 10 + 12 = 22.

Definition 5**:** *Total Utility of dataset; indicated as TU is sum of all the transaction utilities of the dataset. TU =* ∑(Tr) for Tr ∈ DS.
Total utility = 15+32+37+17+43 = 144

Definition 6: *Transaction-Weighted Utility (TWU),* for an itemset X is described as the summation of the utilities from every transaction having X as an itemset, TWU(X) = ∑TU(T_r) for X ⊆ Tr ∧ Tr ∈ DS.
Example, TWU of {t, v} = TU(T_1) + TU(T_2) = 15+32 = 47. Table 3 shows the TWU for the 1-itemsets of the given dataset.

Definition 7: *HTWUI;* an itemset X is characterized as high transaction-weighted utility itemset (HTWUI) if TWU of the itemset is larger than or equal to minimum threshold. The minimum threshold Th is defined as product of TU and a user specified threshold ratio α. Thus, Th = TU× α
HTWUI= {X | TWU (X) ≥ Th}.

Definition 8: *High Utility Itemset (HUI);* any itemset X is characterized as high utility itemset (HUI) if its utility is no less than a minimum utility threshold specified by the user, otherwise, X is a low utility itemset.

Definition 9: *Total Order;* represented by symbol ~ is the itemsets arranged in ascending direction of their TWU values. According to TWU values of table 3, the total ordering can be shown as: w ~ z ~ y ~ t ~ u ~ v ~ x. Items are arranged in the enumerated tree based on this order as shown in figure 1.

Definition 10: *Revised Transactional Dataset;* is a dataset where the items with TWU less than or equal to Th are clipped and the residual items of transactions are arranged with ascending values of TWU.
RT-DS ← Sorted TWU. The clipped items are called as unpromising items. If Th is assumed as 70, item w is called unpromising item and it is removed from the revised transaction dataset. Table 4 represents the RT-DS for the given dataset.

Table 3: TWU for 1-itemset

Itemset	t	u	v	w	x	y	z
TWU	84	97	107	69	129	80	75

Table 4: Revised Transactional Dataset (RT-DS)

Tid	Transactions (item:utility)
1)	(t:5), (v:4)
2)	(z:16), (t:10), (v:4), (x:2)
3)	(y:18), (t:5), (u:1), (x:4)
4)	(u:1), (v:6), (x:4)
5)	(z:16), (y:18), (u:1), (v:6), (x:2)

Definition 11: *Remaining Utility;* RU(A, Tr) for a transaction Tr from RT-DS is the summation of utilities of items followed by itemset A of that transaction Tr.
RU(A, Tr) = ∑U(I_i, Tr) for all I_i ~ I, ∀ I ∈ A
Example, RU for itemset {y, t} for transaction T_3 = U(u, T_3) + U(x, T_3) = 1+4 = 5

Definition 12: *Itemset Extension*: For an itemset χ, Ex(χ) defines the items that are followed by χ in total ordering.
Example, Extension of itemset {u}, Ex(u)= {v, x}; for itemset {t}, Ex(t) = {u, v, x}

Definition 13: *Projected Dataset:* For an itemset Ψ, the projected dataset is denoted as Ψ - DS. It is defined as:
Ψ - DS = { Ψ - Tr | Tr ∈ DS ∩ Ψ - Tr not equals to θ}.
Here, Ψ - Tr is the projection for transaction and Ψ - Tr = {I_i | I_i ∈ Tr ∩ I_i ∈ Ex(χ)}

Definition 14*: Local Utility;* LU(χ, q), for an itemset χ and item q is-
LU(χ, q) = ∑ [U(χ, Tr) + RU(χ, Tr)] for all (χ ∪ q) ∈ Tr from RT-DS of χ
Example, let χ = t
LU(t, v) = U(t) + RU(t) for all transactions having itemset t and item v of RT-DS
= [5+(4)]T_1 + [10+(4+2)]T_2
=25

Definition 15: *Subtree Utility;* SU(χ, q), for an itemset χ and an item q, which have the extension of χ.
SU(χ, q) = ∑ [U(χ, Tr) + U(q, Tr) + (∑ U(I_i, Tr) for (I_i ∈ Tr ∩ Ex(χ)) for (χ ∪ q) ∈ Tr]
Example, let χ = Φ
SU(Φ, t) =[0 + U(t) + U(Ex(t))] for all transaction having item t of RT-DS
= $[0+5+(4)]_{T1}$ + $[0+10+(4+2)]_{T2}$ + $[0+5+(1+4)]_{T3}$
=35
Now, let χ = t
SU(t, v) = [U(t) + U(v) + U(Ex(v))] for all transactions having itemset t followed by v of RT-DS
=[5+4+(0)]T_1 + [10+4+(2)]T_2
=25

The SU and LU are the pruning strategies. If SU(χ, q) < Th, then the itemset {χ∪ q} and its following items can be clipped from the search space. Similarly if LU(χ, q) < Th, then the branches with itemset χ and item q can be clipped.

Problem Statement

For a dataset 'DS', if the minimum utility threshold is 'Th', the aim is to determine all the high utility itemsets in the distributed environment by parallel mining process over multiple nodes.

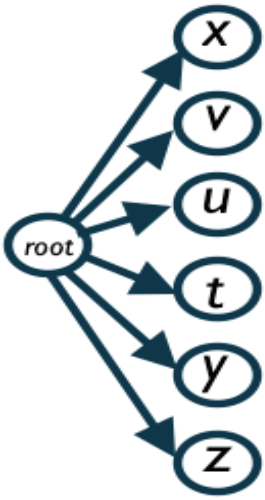

Figure 1: 1-HTWUI Tree

3. PROPOSED METHOD – AHUIM

A novel technique, Absolute High Utility Itemset Mining algorithm or AHUIM is presented in this section, which extends the state-of-the-art EFIM algorithm. The novel algorithm models EFIM on a parallel framework to work in the distributed environment. This way, fast and efficient processing of large datasets or big data is carried out by exploiting the 'divide and conquer' approach. AHUIM uses Apache Spark framework where RDD is the format for data storage. The data is partitioned on multiple nodes, by a method inspired from PFP [13], which then process the task independently. The algorithm AHUIM can be briefly described as follows.

3.1 Tree Structure of the Items

Utility list is built for all the items based on their quantity and profit factor. This utility list is used to compute the transaction weighted utility or TWU of all the 1-itemsets. Items, whose TWU values are less than the threshold, are rejected as unpromising items. Remaining promising items are then arranged in the increasing order of their TWU values in the transactions and this dataset is called as revised transactional dataset. The Spark framework provides the functionality of RDD, which is used for this revised dataset to read it multiple times. The sorted items are used to build the enumerated tree structure, which is distributed among the available nodes.

3.2 Search Space Division

The enumerated tree, which is composed of items to explore and their sub-trees, is divided among the multiple worker nodes so that it can be executed efficiently in the distributed framework. The division is made based on the number of items and nodes. Y. Chin et al. [13] has proposed a division approach, which is being implemented here to divide the workload uniformly among the nodes. If there are N nodes, and M sorted items according to their TWU values in the itemset M= $\{I_1, I_2, I_3 \ldots I_M\}$, then the items are divided among the nodes such that item I_1, I_2, I_3… are allotted to node 1, 2, 3…and N and after that N, N-1, N-2 and henceforth. Aimed at the instance of 4 nodes, and 9 items, {O, P, Q … V, W}, then the items O, P, Q, R are allotted to nodes 1, 2, 3 and 4. The items S, T, U, V are allotted to nodes 4, 3, 2 and 1 and the item W is allotted again to node 1. This division of search space can be portrayed with the procedure of the algorithm I-Node-Item Distribution. The nodes are the keys and the items are the values. The idea is to store the items in a list based on their TWU values. Then values and keys are fetched from the list and put into a new hashmap. The job is to sort the hashmap according to the values.

Algorithm I: Node-Item Distribution

Input: **N**-Number of nodes,
L-List of sorted-TWU items

Output: EHP- a hashmap, for mapping the items to nodes (N: I)

1. EHP ←hashmap( )
2. N ←1
3. inc ←1
4. f ←false
5. **for** each item $I_i \in$ L **do**
   a. HP[I_i] ←N
   b. N ←N + inc
   c. **If** node is the first or last node
      i. **If** f then
      ii. f ←false
         a) **if** node is the first one
         b) inc ←1
         c) **else**
         d) inc ← −1
         e) **end**
      iii. **else**
      iv. f ←true
      v. inc ←0
      vi. **end**
   d. **end**
6. **end**
7. return EHP

3.3 Pruning Strategies

Pruning strategies are used to discard the part of search space, which seems redundant or unpromising to avoid the unnecessary traversing. Apart from transactional weighted utility that is discussed in earlier section, two innovative strategies are being used called as Absolute Local Utility (ALU) and Absolute Subtree Utility (ASU). These absolute utilities suggest more absolute upper limit than the defined utilities of EFIM algorithm and restrict more items, thus boosting the pruning capabilities of the algorithm and reduce the search space further. They are stated as follows.

Definition 16: *Absolute Local Utility (ALU):* An absolute upper bound than the existing local utility and is represented as ALU. Few of the overemphasized utilities are being pruned as the utility is calculated from the projected dataset. Let there are two itemsets, fi and β. The itemset fi is defined as the *following itemset* and it is described as:

$fi(ß) = \{ x \mid ALU(ß, x) \geq Th\}$

where,

$ALU(\chi, q) = \sum[U(\chi, Tr) + ARU(\chi, Tr)]$ for all $(\chi \cup q) \in Tr$, from RT-DS of χ]

The following items, fi are utilized for generating a new aspirant itemset for assessment. The purpose of absolute local utility is to prune some of the overemphasized utilities using the constraints given in the definition.

Definition 17: *Absolute Remaining Utility (ARU):* The absolute remaining utility is used for the computation of possible utilities of the following itemset from its projected dataset.

The concept of remaining utility is being used for the estimation of potential utility for the following items of an itemset. If some items are not present in fi(X) and Ex(X), then there are least chances by them to offer any kind of utility to the following aspirant itemset. Thus the upper bound ARU decreases the estimated utility further as it traverses the projected dataset and the method could able to achieve a reduced upper bound for mining and efficiently decreases the expanse of traversing the aspirant itemset. Thus, ARU can be expressed as:

$ARU(A, Tr) = \sum U(I_i, Tr)$ for all $[(I_i \sim I) \cap (I_i \in fi(A) \cap (I_i \in Ex(A)), \forall I \in A]$ from RT-DS of A only.

Example, as calculated earlier, Remaining Utility for itemset {y, t} for transaction $T_3 = U(u, T_3) + U(x, T_3) = 1+4 = 5$. But if fi{y, t} and Ex{y, t} does not have an item u, then the traversing method will not visit the item u combined with itemset {y, t}. So, the utility of u is not required to be added here in the remaining utility of {y, t}. Consequently, the improved remaining utility called as absolute remaining utility for transaction T_3 for the itemset {y, t} is calculated from the projected dataset of {y, t}= $U(x, T_3) = 4$. This utility value is more precise and firmer upper bound and is utilized for defining the new absolute local utility. Reading only those transactions, which have some item between ß and x in the total order, and clipping all other transactions, can do another improvement.

Definition 18: *Absolute Subtree Utility (ASU):* The subtree utility, SU(χ, q) as stated earlier adds the utility of itemset χ, the utility of item q, and the utility of Ex (χ ∪ q) for some correlated transaction. It in fact, adds some of the utilities from unpromising items. Hence, some of the unpromising items are visited and clipped off later. An amendment in this technique is suggested with an improved sub tree utility, called as absolute sub tree utility that limits the process of vising unpromising items by calculating the ASU from the projected dataset only.

$ASU(\chi, q) = \sum [U(\chi, Tr) + U(q, Tr) + (\sum U(I_i, Tr)$ for $(I_i \in Tr \cap Ex(\chi))$ for $((\chi \cup q) \cap (q \in (fi(\chi) \cap Ex(\chi)) \in Tr)]$, from RT-DS of χ

Example, as calculated earlier, $SU(\{t\}, v) = [5+4+(0)]T_1 + [10+4+(2)]T_2 = 25$

But assume that there is some present itemset {t} and let fi{t} = {v, u, z} and Ex{t} = {v, u, x}. Now SU({t}, v) includes the utility of item x also, which is not present in fi{t}, $x \notin fi\{t\}$. So, adding the utility of x into SU({t}, v) is worthless as the method will not add them in the following steps. Absolute subtree utility does not add such utilities. Here $ASU(\{t\}, v) = [5+4+(0)]T_1 + [10+4+(0)]T_2 = 23$, which is much precise and stricter upper bound than SU.

3.4 Proposed Method - AHUIM

The proposed method AHUIM takes the transactional dataset and the minimum utility threshold as input. An itemet F is taken as an empty itemset. The method then calculates the local utility for all the items, which is initially equal to their TWU values. The initial local utilities or TWU values of items are used to compute the extension of F by comparing with the minimum utility threshold. These items of F are then arranged with their sorted TWU values. And the items that are not a part of extension of F (i.e. the unpromising items based on the TWU value) are discarded here only as they cannot be a part of any larger HUI set. Dataset is then reviewed by arranging all the remaining items of a transaction in order of their rising values of TWU. Any empty transaction is deleted here. The algorithm then calculates the subtree utility for all the items of F. The concept of succeeding and following items of an itemset is being used to create larger itemsets based on the local and subtree utilities.

The worker nodes in the cluster setup are accountable to mine the HUIs for their allotted search space. For example, node 1 mines the HUIs for items z and x. The binary recursive search procedure is used for recursively identifying the HUIs from a node. The main algorithm is displayed as Absolute_HUIM, which provides all the HUIs on termination.

Algorithm II: Absolute_HUIM

Input: **DS-**Transactional dataset,
I-List of items
Th- Minimum threshold rate

Output: Itemsets with high utility

1. F = empty itemset
2. Compute locU (F, I_i) ∀ items $I_i \in I$ by DS scan
3. Compute fi (F) = {$I_i \mid I_i \in I \wedge$ locU (F, I_i) ≥ Th}
4. For fi (F), arrange according to increasing TWU values
5. Prune items $I_i \notin$ fi (F) and remove null transactions
6. Sort the remaining transactions by ∼
7. Compute the SU (F, I_i) for every item $I_i \in$ fi (F) by DS scan
8. The succeeding_items for F, si (F) = {$I_i \in$ fi (F) ∧ SU (F, I_i) ≥ Th}
9. Return **Search** (F, DS, si (F), fi (F), Th)

A recursive search process is then carried out to execute the depth-first search of F. The method is represented in algorithm III. Input parameters are current itemset F, projected dataset of F, following and succeeding items of itemset F and minimum threshold Th. For each item belongs to succeeding items of F, a loop is called for another itemset Ω such that Ω = F ∪{I_i}. A dataset scan is done to compute utility of itemset Ω. The projected dataset of itemset Ω is created in the next step. For the itemset Ω, if the utility is larger than or equal to specified minimum threshold, then it can be considered as a HUI. The dataset is then visualized another time to compute absolute sub tree utility and absolute local utility for each item q belongs to Ω. The process is then called recursively for Ω to resume the search procedure by extending Ω. At the end of the algorithm, high utility itemets are reverted as output.

Algorithm III: Search

Input: **F**: itemset
F–DS: the projected dataset of F
si(F): succeeding items of F
fi(F): following items of F
Th: minimal threshold

Output: I_{HUI}**:** itemsets of high utility

1. I_{HUI} = emptyset
2. **for** every item $I_i \in$ si (F)
   a. Ω = F ∪ {I_i}
   b. Scan F−DS
   c. Compute U(Ω)
   d. Construct Ω –DS // projected dataset of Ω
   e. if U(Ω) ≥ Th,
   f. Ω ⟶ I_{HUI}
   g. end
   h. if Ω – DS = empty,
   i. Compute ASU(Ω, q)
   j. Compute ALU(Ω,q) ∀ items q ∈ fi(F) from Ω –DS scan
   k. si (Ω) = q ∈ fi(F) | ASU (Ω, q) ≥ Th
   l. fi (Ω) = q ∈ fi(F) | ALU (Ω, q) ≥ Th
   m. I_{HUI} ∪ **Search** (Ω, Ω –DS, si (Ω), fi (Ω), Th)
   n. end
3. end
4. return I_{HUI}

The overall flow graph of the algorithm AHUIM with the parallel framework of Spark is represented in the figure 2.

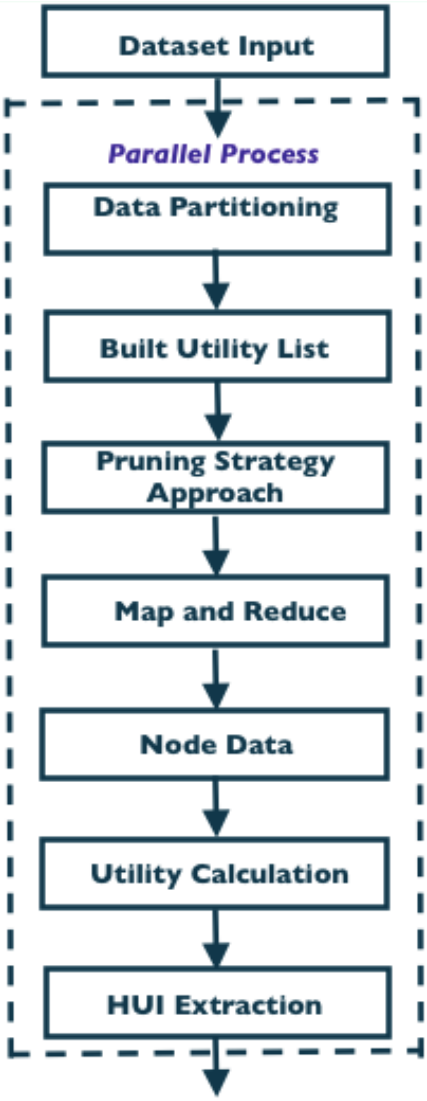

Figure 2: Flow Graph of AHUIM

4. EXPERIMENTAL SETUP

For experiments, Spark cluster is created with a main/driver node and ten slave nodes on a system with 60 GB main memory with AMD EPYC 7282 16-Core Processor @ 2.80 GHz. The operating system is Windows 10. All the nodes have following software configuration: Apache spark 3.0, python 3.7 and Spyder4 as the IDE.

4.1 Dataset

The experiments have been performed on three real-world datasets- Chess, Connect and Mushroom. Chess and Connect are the datasets for game. Mushroom is a dataset for different

variety of Mushrooms. These datasets have been taken from SPMF Library, which is a data-mining library for various algorithms and codes. Attributes of the datasets are shown in the table. #Transactions, #Items, #AverageItems, which represent the total transactions in the dataset, figure of unique items, the mean amount of items for a transaction respectively. Detailed information about the datasets has been given in the repository [14]. For relatively larger datasets, small datasets have been multiplied with a scalar to get a larger dataset. The experiments have been conducted five times for each threshold ratio and the average values have been taken.

Table 5: Various Datasets (Source-SPMF Library)

Dataset	#Transactions	#Items	#AverageItems
Chess	3196	75	37
Connect	67557	129	43
Mushroom	8416	119	23
Chess30x	95880	75	37
Connect2x	135114	129	43
Mushroom20x	168320	119	23

4.2 Assessment of the Algorithm

To assess the performance of the algorithm, the AHUIM is compared with two versions of itself – AHUI_basic and AHUIM_SA. The AHUI_basic does not apply the strategies, which have been recommended for the novel algorithm. The algorithm AHUIM_SA is a stand-alone system for AHUIM, which lanes on only one node without any parallel framework. The following measures have been used: A. Execution Time, B. Scalability and C. Memory Utilization.

A. Execution Time

Table 6 displays the time-performance of the three algorithms on the datasets with distinctive values of threshold. It can be seen that AHUIM performs better than other two versions because of the pruning strategies and large number of working nodes. For example, for the dataset Chess30x, the algorithm AHUIM is around 12 times speedier than AHUIM_SA and 5 times speedier than AHUI_basic for the threshold value 3.7. Similarly, for the threshold value 2.7, the algorithm AHUIM is around 23 times faster than AHUIM_SA and 10 times faster than AHUI_basic for the threshold value 2.7. Thus, AHUIM is much faster than AHUIM_SA since the data is being handled in parallel. AHUIM_SA performs all the processing on a single system, where as the algorithm AHUIM distributes the data among ten nodes to process simultaneously. The average execution time of AHUIM on Chess30x is around 1 minute, whereas average execution time of AHUIM_SA is around 20 minutes.

Table 6: Execution Time for Different Datasets

Dataset	Execution Time in Seconds			
	Threshold	AHUIM	AHUIM_SA	AHUI_Basic
Chess30x	1.7	15.81772	2100.9865	400.872
	2.7	101.74081	2208.2558	524.9863
	3.7	112.0043	2212.0045	628.6273
	4.7	115.5771	2356.5486	690.6457
Connect2x	1.7	481.8532	4092.8372	1012.862
	2.7	509.72583	3509.7258	1192.923
	3.7	424.0159	3865.8986	978.4588
	4.7	411.6716	5698.2656	1361.954
Mushroom20x	1.7	36.6659	4792.8372	290.7865
	2.7	136.2395	3369.9856	465.5487
	3.7	136.5605	3989.4164	498.6584
	4.7	34.7257	3549.669	386.5489

The graphical representations of execution time can be perceived in figures 3, 4 and 5.

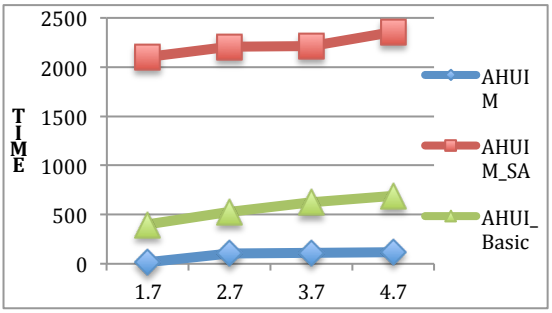

Figure 3: Execution Time in Seconds for Chess30x Dataset for Different Thresholds.

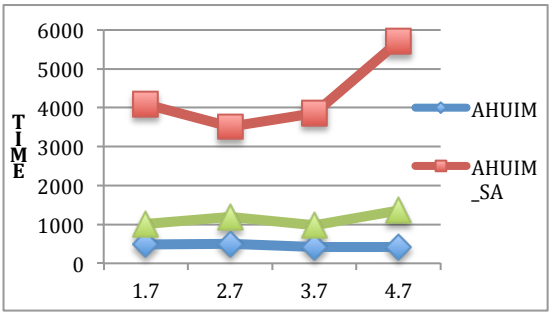

Figure 4: Execution Time in Seconds for Connect2x Dataset for Different Thresholds.

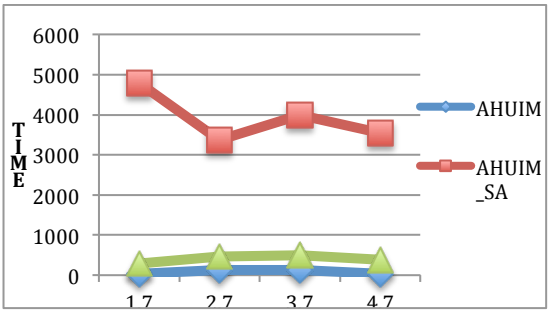

Figure 5: Execution Time in Seconds for Mushroom20x Dataset for Different Thresholds.

B. Scalability

Scalability test has been performed in two ways. First, the algorithm AHUIM is run on different datasets of fixed size but with varying number of nodes. The result can be seen in figure 6 with nodes 1, 5 and 10. With expansion in number of working nodes, execution time of the algorithm decreases approximately linearly. This confirms that the algorithm has the ability to partition the search space among various nodes efficiently and perform the task of mining independently.

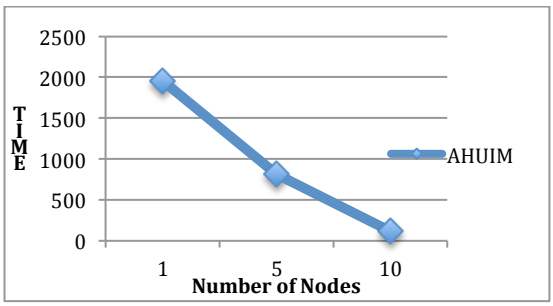

Figure 6: Execution Time in Seconds for Chess30x for Different Nodes (Th=4.7)

Second method to examine the scalability of the algorithm is used by increasing the size of the database linearly in some proportion (like 100%, 200%, 300%, etc.). The dataset can be duplicated by a scalar d, such that experimental dataset = original dataset*d, and d = 1, 2, 3 etc. The execution time is then measured for each of the dataset as shown in table 7.

Table 7: Execution Time in Seconds on Different Size of Dataset

Algorithm	Chess10x	Chess20x	Chess30x
AHUIM	78.9856	94.675	115.5771
AHUIM_SA	872.98	1451.823	2356.548

From figure 7, it can be perceived that running time of AHUIM breeds very slowly and almost flat with increase in datasets. Both the methods confirm the scalability of the algorithm. But as shown in figure 8, the running time for AHUIM_SA increases with very high pace as the size of data increases. Thus, it is seen that the novel algorithm accomplishes better for big data sets. The scalability of the algorithm can be related to raise the efficiency because of the suggested strategies of search space pruning and efficient data partition.

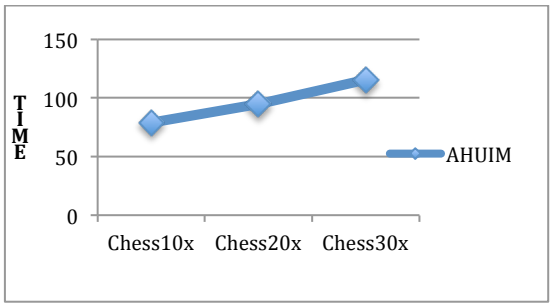

Figure 7: Execution Time in Seconds for Different Size of Chess Dataset (Th=4.7)

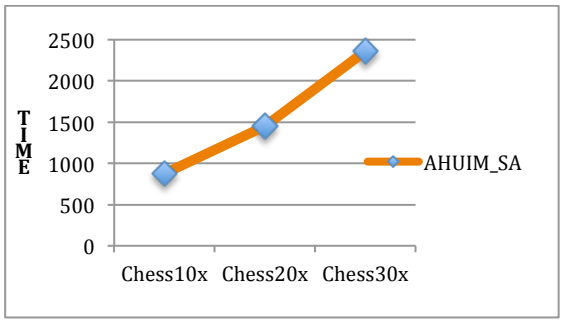

Figure 8: Execution Time in Seconds for Different Size of Chess Dataset (Th=4.7)

C. Memory Utilization

The utilization of memory is calculated in terms of percentage of main memory allotment by the algorithm during run time. The Python has inbuilt system library psutil to calculate the memory utilization by a process. The values are shown in the table for different datasets. Although the memory consumption is almost equal for all the datasets, it can be seen that dataset Chess30x consumes less memory than other two datasets as it is having less number of transactions and distinct items than other datasets.

Table 8: Memory Utilization by AHUIM for Different Datasets

Dataset	Connect2x	Chess30x	Mushroom30x
Main Memory Utilization by AHUIM	3.18E-05	2.54E-05	3.19E-05

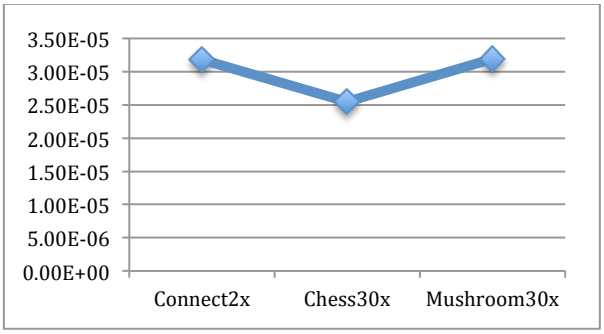

Figure 9: Memory Utilization by AHUIM for Different Datasets

5. CONCLUSION AND FUTURE WORK

The traditional HUIM algorithms are not suitable for the storage and processing of big data. The standalone architecture of the system is being interchanged with the distributed processing. A novel technique to mine the HUIs from large datasets is being projected in this research work. The method extends the EFIM technique of data mining on the parallel framework of Apache Spark with effective pruning strategies. Comprehensive experiments advocate the performance of AHUIM for time, scalability and memory consumption. As a part of further research, the novel algorithm can be enhanced with other features of HUIM such as mining top-k itemsets, rare itemsets, itemsets with negative utilities, etc.

REFERENCES

1. Agrawal, R. and R. Srikant, **Fast Algorithms for Mining Association Rules in Large Databases**, *International conference on very large databases.* San Francisco, Morgan Kaufmann publishers, pp. 487–499, 2003
2. Vo B., Nguyen H., Ho T. B., Le B. **Parallel Method for Mining High Utility Itemsets from Vertically Partitioned Distributed Databases,** *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems,* KES, Volume: 5711 LNAI Issue: PART 1 Page: 251-260, 2009.
3. Subramanian, K., Kandhasamy, P., Subramanian, S., "**A Novel Approach to Extract High Utility Itemsets from Distributed Databases**", Computing and Informatics, 31 (6), 1597-1615, 2013.
4. Lin, J. C. W., Li, T., Fournier-Viger, P., Hong, T.P., Zhan, J., Voznak, M., **"An Efficient Algorithm to Mine High Average-Utility Itemsets",** Advance Engineering Informatics, 30 (2), 233-243, 2016
5. Chen, Y., An, A., **"Approximate Parallel High Utility Itemset Mining"**, Big Data Res. 6, 26-42, 2016.
6. Zihayat, M., Hut, Z. Z., an, A., & Hut, Y., **Distributed and Parallel High Utility Sequential Pattern Mining**, *In 2016 IEEE International Conference on Big Data (Big Data)* pp. 853-862. IEEE, 2016.
7. Ashish Tamrakar, **High Utility Itemsets Identification in Big Data**, M.S. thesis, University of Nevada, Las Vegas, 2017.
8. Sethi, K. K., Ramesh, D. Edla, D.R., **"P-FHM+: Parallel High Utility Itemset Mining Algorithm for Big Data Processing"**, Procedia Computer Science. 132, 918-927, 2018.
9. Nguyen, T. D., Nguyen, L.T., Vo, B., **"A Parallel Algorithm for Mining High Utility Itemsets"**, Springer, Cham, pp. 286-295, 2018.
10. Sethi, K. K., Ramesh, D., Sreenu, M., **"Parallel High Average-Utility Itemset Mining Using Better Search Space Division Approach"**, Springer, Cham, pp 233-243, 2019.
11. Borthakur, D., **"The Hadoop Distribued File System: Architecture and Design"**, Hadoop Project Website 11, 21, 2019
12. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., MaCauley, M., Stoica, I., **Resilient Distributed Datasets: A Fault-Tolerant abstraction for In-memory Cluster Computing**, *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2010.
13. H. Li., Y. Wang, D. Zhang, E.Y. Chang, **PFP: Parallel FP-Growth for Query Recommendation.** *In: Proceedings of the 2008 ACM Conference on Recommender Systems*, ACM, 2008, pp. 107-114.
14. www.archive.ics.uci.edu/ml/datasets
15. M. Zihayat and A. A. Mining, **"Top-k High Utility Patterns Over Data Streams"**, In Information Sciences, 2014.
16. C. F. Ahmed, S. K. Tanbeer, and B. Jeong., **"A Novel Approach for Mining High-Utility Sequential Patterns in Sequence Databases"**, In ETRI Journal, vol. 32, pp. 676–686, 2010.
17. Jimmy Ming-Tai Wu, Jerry Chun-Wei Lin, and Ashish Tamrakar, **"High-Utility Itemset Mining with Effective Pruning Strategies**", ACM Transactions. Knowledge Discovery Data 13, 6, Article 58, 22 pages, 2019.
18. Dalal Sandeep, Dahiya Vandna., **"Review of High Utility Itemset Mining Algorithms for Big Data**", Journal of Advanced Research in Dynamical and Control Systems- JARDCS, 10(4), pp: 274-283, 2018.
19. Zida, S., Fournier-Viger, P., Wu, C.-W., Lin, J.C.-W., Tseng, V.S., **Efficient Mining of High Utility Sequential Rules,** *In: Proc. 11th Intern. Conf. on Machine Learning and Data Mining,* pp. 157– 171. *Springer, 2015.*
20. Ryang, H., Yun, U., **"High Utility Pattern Mining Over Data Streams With Sliding Window Technique",** Expert Systems with Applications 57, 214–231, 2016.
21. Vandna Dahiya, Sandeep Dalal, **"Big data Mining: Current Status and Future Prospects",** International Journal of Advanced science and Technology, Volume 29, No 3, pp. 4659- 4670, 2020.
22. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S., **FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co- Occurrence Pruning**, *In: Proc. 21st Inter. Symp. Methodologies for Intelligent Systems, Springer.* pp. 83–92, 2014.
23. Ibrahim A. Atoum, Nasser A. Al-Jarallah, **"Big Data Analytics for Value-Based Care: Challenges and Opportunities"**, International Journal of Advanced Trends in Computer Science and Engineering. Volume 8(6), pp 3012-3016, 2019.
24. G. Sowmya, N.Sirisha, G. Divya Jyothi, **"Use of Big Data to Measure Attentiveness of the Student"**, International Journal of Advanced Trends in Computer Science and Engineering. Volume 9(2), pp 2350-2352, 2020.
25. Heungmo Ryang, Unil Yun, and Keun Ho Ryu, **"Fast Algorithm For High Utility Pattern Mining With The Sum Of Item Quantities"**, Intelligent Data Analysis. 20(2), 395–415, 2016.
26. Guangming Guo, Lei Zhang, Qi Liu, Enhong Chen, Feida Zhu, Chu Guan, **"High Utility Episode Mining Made Practical and Fast",** Advanced Data Mining and Applications, pp 71-84, 2014, Springer, 2016.
27. X. Wu, X. Zhu, G. Q. Wu, and W. Ding, **Data Mining with Big Data.** ***Knowledge and Data Engineering***,

IEEE Transactions. 26 (1) p. 97-107, 2016.

28. Dalal Sandeep, Dahiya Vandna, **"Various Research Opportunities in High Utility Itemset Mining,"** International Journal of Recent Technology and Engineering (IJRTE), Volume 8(4), pp – 2455-2461, 2019.
29. https://www.philippe-fournier-viger.com/spmf/- An open source Data Mining Library
30. https://hadoop.apache.org/
31. https://spark.apache.org/