Parallel High Average-Utility Itemset Mining Using Better Search Space Division Approach

Krishan Kumar Sethi[1], Dharavath Ramesh[1](✉), and M. Sreenu[2]

[1] Indian Institute of Technology (ISM) Dhanbad, Dhanbad 826004, Jharkhand, India
{kksethi,ramesh.d.in}@ieee.org
[2] Ashoka Institute of Engineering and Technology, Hyderabad 500082, Telangana, India

Abstract. Since the last decade, High Utility Itemset (HUI) mining has emerged as a popular pattern mining approach. HUI mining discovers a set of itemset with their profit more than a user defined profit threshold. High Average-Utility Itemset (HAUI) mining is an improvement over HUI mining that involves the length of items to refine the patterns and keep a fair mining process. In the era of big data, traditional HAUI mining algorithms are not suitable to process large transaction dataset on standalone system due to limitation of processing resources. Therefore, several distributed frameworks have been developed to process big data on cluster of commodity hardwares. This paper presents a parallel version of the traditional HAUI-Miner algorithm and names it as Parallel High-Average Utility Itemset Miner (PHAUIM). PHAUIM is a Spark-based distributed algorithm which splits the dataset into multiple chunks and distributes on cluster nodes to process each data chunk in parallel. In addition, an improved approach for search space division is developed. Proposed search space division technique fairly assigns the workload to each node and upgrades the performance. Comprehensive experiments have been performed to measure the performance of PHAUIM in terms of speedup and data scalability. PHAUIM is also compared with traditional HAUIM.

Keywords: High average-utility itemset mining · Big data
Apache-Spark · Search space

1 Introduction

Discovery of useful patterns from a dataset is the task of pattern mining techniques [7,10]. Association rule and frequent itemset are two major constructs in the traditional pattern mining approaches that are still used for several real-life applications. Frequent Itemset Mining (FIM) [1,2,8,11] searches all frequent

Supported by Indian Institute of Technology (ISM) Dhanbad.

itemsets from a transaction dataset which have frequency(also called as support) more than a user defined support threshold. Apriori algorithm is a fundamental algorithm which performs a level-wise searching to discover a set of frequent itemsets [2]. It applies a Downward Closure (DC) property to prune the unpromising candidate itemsets from the search space. The DC property states that if any item in the search space is not frequent then none of its superset can be frequent. However, support is not an important measure to define useful patterns for various organizations. Therefore, FIM is extended by incorporating two new parameters called utility and quantity of itemset. Utility of an itemset means the profit obtained by that itemset. This new field of pattern mining is named as high utility itemset mining [3,28]. An itemset is termed as high utility itemset if its utility is no less than a user defined minimum utility threshold. Utility measure does not follow the DC property unlike the apriori algorithm. Therefore, a novel measure called Transaction Weighted Utility (TWU) [22] is introduced to retain the DC property in HUI mining. Several two-phase algorithms [17,23,26] have adopted TWU model to apply pruning. All two-phase algorithms search a set of strong candidate itemsets at the end of first phase and calculate the actual utility value to find the HUIs in the second phase. HUI-Miner [21] is the earliest algorithm which discovers HUIs from a dataset in one-phase. It has introduced a list structure called utility list to store utility of itemset and required information to apply pruning method. Thereafter, several improved one-phase algorithms [9,13,14,31] have been developed.

HUI miner suffers from a major limitation that length of the itemset affects the quality of the patterns. It is most likely that an itemset with larger length will has more chances to be a HUI regardless its usefulness. Therefore, an alternative of utility measure called average-utility is introduced to remove the impact of length constraint from the HUI miner. Average-utility of an itemset is measured by dividing the utility by its length. Pattern mining using average-utility is named as high average-utility itemset mining [12,15,16]. TWU is not applicable in HAUI mining algorithms. Therefore, a novel measure called Average Utility Upper-Bound (AUUB) [12] is introduced to acquire the downward closure property. Several HAUI mining algorithms [15,18] have adopted AUUB model to serve the pruning method. However, these algorithms are two-phase in nature and require more time and computing resources to search HAUIs. HAUI-Miner [18] is the first one-phase algorithm which produces candidate itemsets and HAUIs together. It has presented a list structure called Average-Utility (AU) list. Average-utility list is similar concept like utility list of HUI-Miner [21]. It requires a dataset scan to construct the AU-list of 1-itemset. Then, joining of AU-list of $(k-1)$-itemset is performed to build the AU-list of k-itemset. Various advanced HAUI mining algorithms [19,29] have been developed with tighter upper-bounds and efficient pruning methods.

A big explosion in technology and data industries have generated huge volume of data, lying on the servers. This data is termed as big data [4]. To apply pattern mining techniques on big transaction data is a challenging task because of two factors. First, big transaction data generates massive number of candidate itemset which might not fit in the system memory. Second, most of the pattern mining algorithms are developed for the standalone system which might not suitable to store and process big data. Many distributed frameworks such as hadoop [27], Spark [30] have been developed to effectively process big data with cluster computing approach. Many studies [5,20,24,25] have been performed to apply pattern mining techniques such as FIM, HUI mining on big data. Mining in distributed systems requires to divide and distribute the processing load on multiple machines so that processing can be achieved independently in a parallel manner. Therefore, search space division is a challenging task for data processing in distributed manner.

In this paper, parallel version of HAUI-Miner algorithm [18] named parallel HAUI-Miner is proposed. PHAUIM is a Spark based distributed algorithm. Apache Spark is considered as one of the best distributed framework for iterative data processing. Its in-memory processing and flexible programming environment uplift it over the hadoop framework. It introduces a new data structure called Resilient Distributed Dataset (RDD) that is collection of immutable distributed objects. Data in RDD is automatically distributed across the cluster nodes by spark framework to parallelize data processing. Moreover, a better search space division method is introduced to fairly divide the workload on each node. This method considers both AUUB values of items and length of subspace to define a weight factor for each node. It assures that no node with higher weight gets more workload than lower weight node. This paper has the following contributions.

1. A parallel algorithm PHAUIM is presented which models the traditional HAUI-Miner to work in distributed framework, i.e., Spark. Apache-Spark provides fast processing for iterative algorithms.
2. A better search space division method is introduced to assign the workload fairly on each computing node.
3. Substantial experiments have been conducted to evaluate the performance of proposed algorithm PHAUIM. It has also been compared with the base algorithm HAUI-Miner. It is observed that PHAUIM outperforms HAUI-Miner in speedup.

Remaining paper is arranged in the following manner. Section 2 explains the terms and definitions to describe HUI-mining and HAUI-mining. The proposed algorithm PHAUIM is discussed in Sect. 3. Section 4 contains experiments and performance evaluation of the PHAUIM. Section 5 concludes the paper.

2 Prerequisites

2.1 High Average-Utility Itemset Mining

Let $D = \{t_1, t_2, ..., t_n\}$ be a transaction dataset with n transactions and m distinct items $I = \{i_1, i_2, ..., i_m\}$. Each transaction $t_p \in D (p \in [1, n])$ is denoted as a unique identifier called as Transaction identifier (Tid) and contains a set of items from I. Let $X = \{i_1, i_2, ..., i_k\}$ be a set of k distinct items from I that is called k-itemset. Each item $x \in I$ in a transaction t_p is associated with a positive integer called as purchasing quantity or internal utility of x. Internal utility of the item x in transaction t_p is denoted as $IU(x, t_p)$. Each item also acquires a positive integer for the whole dataset that is called per unit profit or external utility. External utility of item x is denoted as EU(x). External utility values for all the itemsets are stored in the profit table such as PT $= \{EU(i_1), EU(i_2), ..., EU(i_m)\}$. A sample transaction data and profit table are shown in respective Tables 1 and 2. These tables are used as running example throughout the paper.

Table 1. A sample dataset

Tid	item:quantity
t_1	a:2, c:3, d:1, e:5
t_2	a:3, b:4, c:1, e:2
t_3	b:4, c:1, d:4, e:2, f:1
t_4	a:2, b:3, c:3, d:1, e:4
t_5	a:1, b:3, c:2, d:2
t_6	a:2, b:5, c:2, f:2

Table 2. Profit table

Items	Per unit profit
a	2
b	3
c	3
d	2
e	4
f	1

Definition 1 (Item utility)**.** Let x be an item in a transaction t_p, utility of x in t_p is denoted as $U(x, t_p)$ and defined as follows.

$$U(x, t_p) = IU(x, t_p) \times EU(x) \tag{1}$$

Definition 2 (Itemset utility)**.** Let X be an itemset in transaction t_p, utility of X in t_p is denoted as $U(X, t_p)$ and defined as follows.

$$U(X, t_p) = \sum_{x \in X \& X \subseteq t_p} U(x, t_p) \tag{2}$$

Utility of itemset X in dataset D is denoted as $U(X)$ and defined as:

$$U(X) = \sum_{X \subseteq t_p \& t_p \in D} U(X, t_p) \tag{3}$$

Definition 3 (Average-utility)**.** Average-utility of a k-itemset X in transaction t_p is denoted as $AU(X, t_p)$ and defined as follows.

$$AU(X, t_p) = \frac{U(X, t_p)}{\mid X \mid} = \frac{\sum_{x \in X \& X \subseteq t_p} U(x, t_p)}{k} \tag{4}$$

Where, $\mid X \mid$ is the cardinality of itemset X, i.e. $\mid X \mid = k$. Average-utility of X in dataset D is denoted as $AU(X)$ and computed as follows.

$$AU(X) = \sum_{X \subseteq t_p \& t_p \in D} AU(X, t_p) \tag{5}$$

For example, in the running dataset, $AU(\text{ac}) = AU(\text{ac}, t_1) + AU(\text{ac}, t_2) + AU(\text{ac}, t_4) + AU(\text{ac}, t_5) + AU(\text{ac}, t_6) = 6.5 + 4.5 + 6.5 + 4 + 5 = 26.5$.

Definition 4 (Total dataset utility)**.** Total dataset utility of the dataset D is denoted as $TDU(D)$ and defined as follows.

$$TDU(D) = \sum_{t_p \in D} TU(t_p) \tag{6}$$

Where, $TU(t_p)$ is transaction utility of t_p that is computed as:
$TU(t_p) = \sum_{x \in t_p} U(x, t_p)$.

Definition 5 (Minimum average-utility threshold)**.** Minimum average-utility threshold is a user defined percentage value of TDU. Let δ be the user defined percentage value, then minimum average-utility threshold (denoted as $minUtil$) is defined as follows.

$$minUtil = TDU \times \delta \tag{7}$$

For example, in the running dataset, $TDU(D) = 182$. Let δ be 18% then $minUtil = 182 \times 0.18 = 32.76$.

Definition 6 (High average-utility itemset)**.** The itemset X is called as High Average-Utility Itemset (HAUI) if $AU(X) \geq minUtil$.

$$HAUIs \leftarrow \{X | AU(X) \geq minUtil\} \tag{8}$$

For example, in the running dataset, $AU(\text{ac}) = 26.5 < minUtil$. Therefore, itemset (ac) is not a HAUI.

Definition 7 (Transaction maximum utility)**.** Transaction maximum utility (TMU) for a transaction t_p is denoted as $TMU(t_p)$ and defined as the maximum utility value in t_p.

$$TMU(t_p) = max(U(x, t_p) | x \in t_p) \tag{9}$$

Definition 8 (Average-utility upper bound)**.** The average-utility upper bound for itemset X is denoted as $AUUB(X)$ and defined as follows.

$$AUUB(X) = \sum_{X \subseteq t_p \& t_p \in D} TMU(t_p) \tag{10}$$

For example, in the running dataset, AUUB values of the items are as follows. $AUUB(\text{a}) = 72$, $AUUB(\text{b}) = 64$, $AUUB(\text{c}) = 84$, $AUUB(\text{d}) = 57$, $AUUB(\text{e}) = 60$, $AUUB(\text{f}) = 27$.

Property 1 (Downward closure property using AUUB)**.** If an itemset X has AUUB value lesser than $minUtil$ then neither X nor any superset of X is HAUI.

Property 1 is used to apply the pruning step in HAUI-mining algorithms. AUUB provides the upper-bound of average-utility for itemsets to remove the weak candidate itemsets. For example, in the running dataset, $AUUB(\text{f}) = 27 < minUtil$. Therefore, item f is a weak candidate and can be removed from the search space.

Definition 9 (Processing order and revised dataset)**.** In the first scan of dataset, AUUB of all the items is computed. Then, the dataset is revised and used in place of original dataset. There are following two constrains for the dataset revision.

1. All the items with AUUB value lesser than $minUtil$ are removed. As per the Property 1, such items are no longer needed.
2. Items in a revised transaction are arranged as per the ascending order of the AUUB values.

Definition 10 (Average-utility list)**.** Average-utility list for an itemset X is a list of tuples $<Tid, U, TMU>$, where Tid is the transaction ID which stores X, U is the utility of X in Tid and, TMU is the maximum utility of transaction Tid. AU-list of 1-itemset is constructed by a scan of revised dataset. Thereafter, AU-list of k-itemset ($k \geq 2$) can be constructed by joining the AU-list of $(k-1)$-itemset. Hence, no extra dataset scans are required to built the AU-list of larger itemsets. Let $X.AUL$ (m length) and $Y.AUL$ (n length) be the AU-list of two itemsets X and Y. To construct the AU-list of itemset XY, i.e. $XY.AUL$ requires following rules to be employed.

1. All the common Tids in both the AU-lists are put in $XY.AUL$. Tid in both the AU-lists are sorted, therefore searching of common Tids requires $(m + n)$ comparison.
2. For each common Tid, U field is measured as $X.U{+}Y.U$.
3. $Y.TMU$ value is assigned to $XY.TMU$.

Property 2 (Pruning using AU-list)**.** Let X be an itemset and $SUM(X.TMU)$ be the sum of all TMU values in the AU-list of X. If $SUM(X.TMU)$ is lesser than $minUtil$, then neither X nor any superset of X can be HAUI. Therefore, X and all its extensions can be pruned from the search space.

3 Proposed Algorithm: Parallel High Average-Utility Itemset Miner

Parallel High Average-Utility Itemset Miner (PHAUIM) is a Spark based algorithm which is suited to run on big transaction dataset in a distributed manner. PHAUIM works on divide and conquer concept where large problem is divided into smaller problems and solved individually by each node to make mining process faster. Execution flow of the PHAUIM is depicted in Fig. 1. Transaction data are partitioned into smaller and equal size of chunks and distributed across the cluster nodes. The distribution is part of Hadoop framework where we used HDFS as file system. The Data are loaded in a RDD $<Transaction>$ and each node processes the assigned data in parallel. Then, each item is associated with TMU value to form a paired RDD $<item, TMU>$. Thereafter, TMU values of each item across all the nodes is summed to produce another RDD $<item, AUUB>$ and collected at the master node. The transaction data is revised as per the List($[item, AUUB]$) where, each transaction omits all the items with AUUB value lesser than $minUtil$ and remaining items are sorted in ascending order of their AUUB values. All the items from the List($[item, AUUB]$) are extracted and mapped to the number of nodes in the cluster. Thus, a List($[item, Node]$) is obtained which is used for search space division such that only assigned items and their extensions are explored by each node. Moreover, the required transactions from the revised dataset are extracted for each node to reduce the processing complexity. Each node constructs average-utility list, i.e., $items.AUL$ for each assigned item. Then a recursive HAUI mining procedure is applied on $items.AUL$ to produce a set of HAUIs. The HAUIs produced by all the nodes are combined to produce the final output.

The pseudo code of PHAUIM is shown in Algorithm 1. It takes three inputs: transaction dataset D, number of nodes N, minimum average-utility threshold $minUtil$ and produces a list of HAUIs as output. In the first dataset scan, it applies flatmap() method to the D and each item x in transaction t_p is mapped to a key-value form (x, TMU) (line 4). TMU values of each item across the cluster nodes is summed to built another RDD $<x, AUUB>$ using reduceByKey() method (line 5). All the items with AUUB lesser then $minUtil$ are pruned out (line 6) and remaining items along with their AUUB are assigned to RDD $itemAUUB$. Thereafter, a list of items $items$ from the RDD $itemAUUB$ is extracted where items are sorted in ascending order of their AUUB values (line 11). Then, mapping() method is invoked to assign a set of items to each node, i.e., $itemNode$ so that the node can search the HAUIs from all the extensions of assigned items (line 12). Both the $itemAUUB$ and $itenNode$ are broadcasted to each node using broadcast variable (line 13). In the second dataset scan, the dataset is revised and the AUUB value of each item is recalculated from the revised dataset (lines 14–16). Thereafter, revised dataset is again scanned to assign required data to each cluster node using DataGenerator() method (lines 17–18). Each cluster node constructs the AU-list for every assigned item x (lines 19–22). Then, the HAUIMiner() method is recursively called to generate a set of

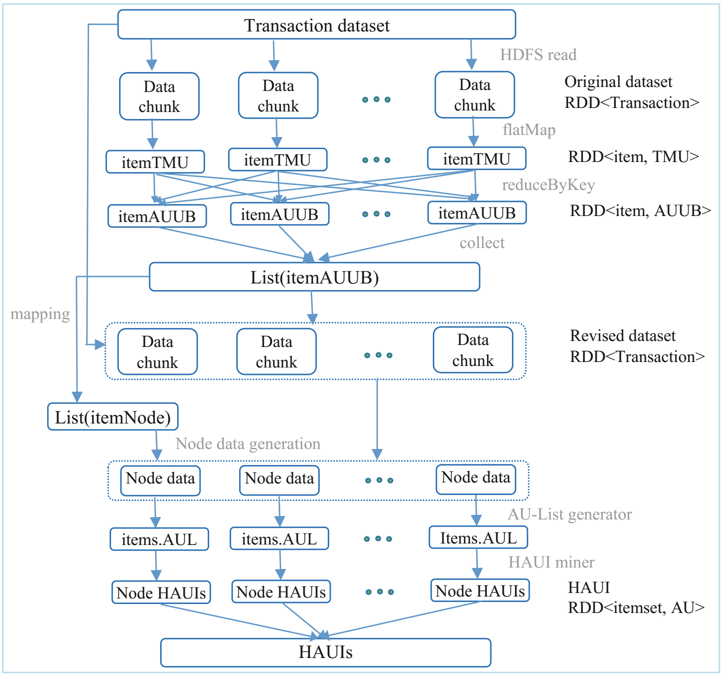


Fig. 1. Execution flow of PHAUIM

HAUIs. All the three methods mapping(), DataGenerator() and HAUIMiner() are discussed in detail in the following subsections.

3.1 Search Space Division

Entire Search space is divided into several subspaces and processed by individual cluster node to speedup the mining. For example, in the running dataset, after the first scan, five items (along with their AUUB values) {d: 57, e: 60, b: 64, a: 72, c: 84} are discovered and sorted in ascending order of their AUUB values. The search space is represented in form of enumeration tree [21] as shown in Fig. 2. Each Node with an item is responsible to mine a set of HAUIs such that any node with item d discovers all the itemsets with item d, node with item e mines all the itemsets having e but not d, node with item b mines itemsets having b but not item d and e, and so on. A fair division of search space is a difficult task. PHAUI-Miner [5] is the first algorithm to implement the HUI-Miner [21] in Spark framework. It assigns the items to cluster nodes in the following sequence. Let $\{1, 2, ..., N\}$ be the number of nodes in the cluster and $\{i_1, i_2, ..., i_k\}$ be the items to be assigned to the nodes. Items are assigned to nodes 1, 2,..., N and, then N, N − 1, ..., 1 and so on. For example, let there are 2 nodes in the cluster then, items assignment will be like this: {Node 1: d, a, c; Node 2: e, b}.

Algorithm 1. PHAUIM

```
INPUT:    $D$: Transaction dataset, $N$: number of nodes
          $minUtil$ : Minimum average-utility threshold
OUTPUT:   $HAUI$: List of high average-utility itemsets
    //First dataset scan
 1: flatMap ($D$)
 2: for each transaction $t_p$ in $D$ do
 3:     for each item $x$ in $t_p$ do
 4:         map ($x \rightarrow (x, TMU)$)
 5:         redeuceByKey (_ + _)
 6:         filter ($x \rightarrow x.\_2 \geq minUtil$)
 7:         $itemAUUB \leftarrow (x, AUUB)$
 8:     end for
 9: end for
10: end flatMap
11: $items = itemAUUB$.toList.sortBy($x \rightarrow (x.\_2, x.\_1)$).map($x \rightarrow x.\_1$)
12: $itemNode$= mapping($itemAUUB, N$)
13: Boradcast the hashmap $itemAUUB$ and $itemNode$
    //Second dataset scan
14: Revise the dataset: remove the items whose $AUUB < minUtil$
15: Each transaction is sorted in ascending order of AUUB values.
16: Recalculate the AUUB values.
    //Third dataset(revised) scan
17: $NodeTrasnactions$= DataGenerator($revisedData, itemNode$)
18: $NodeData$=$NodeTrasnactions$.groupByKey()
19: flatMap ($NodeData$)
20: for each Node $N$ in $NodeData$ do
21:     for each item $x$ at the Node $N$ do
22:         P= $x.AUL$
23:     end for
24:     $HAUI \leftarrow$ HAUIMiner($Null$, $P$, $minUtil$, $itemNode$, $NodeID$, 1 )
25: end for
26: end flatMap
27: Output ($HAUI$)
```

It is observed that the existing search space division method may not perform efficiently because of two factors. First, There is a big difference in number of itemsets in each subspace, e.g., items d, e, b, a, c has 16, 8, 4, 2, 1 itemsets in their respectively subspaces. Second, there may be a large gap in AUUB values of the subspace itemsets. A large AUUB value means more entries in AU-list. Therefore, processing of a larger AU-list requires some extra time. To accommodate these constraints, in this paper, a novel search space division method is proposed that considers both subspace size and items AUUB value for a fair division. Each node is assigned a weight that is computed by using both size of subspace and AUUB value of items. An item is assigned to a node which has the lowest weight after that item assignment. Search space division method mapping() is depicted in Algorithm 2. It takes two inputs: *itemAUUB* and number of nodes N and,

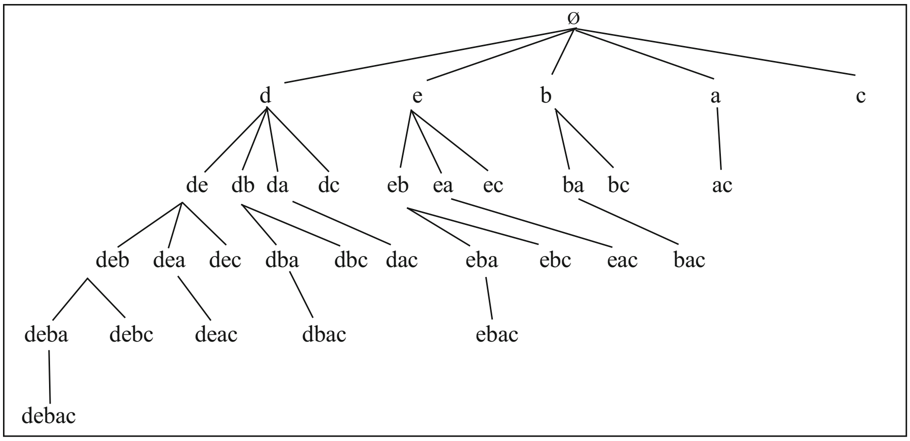


Fig. 2. Search space for the running dataset

produces a hashmap *itemNode* which maps each item to a node. For each item x in *itemAUUB*, the weight of x is computed by adding the subspace size to AUUB value of item (lines 3–4). If a node is not assigned any item then x is assigned to that node and nodeWeight is also updated (lines 8–11, 19–20). Otherwise, for each node, $afterWeight$ is measured which is weight of the node after assignment of x (line 12). Item x is assigned to a node which has lowest value of $afterWeight$ (lines 13–16, 19–20). In the end, *itemNode* is returned as an output. In the running dataset, search space division after applying the proposed method is as follows: {Node 1: d, a; Node 2: e, b, c}.

3.2 Data Generation on Node

Each node is responsible to discover HAUIs from the assigned subspace. Therefore, only required transactions or subset of transactions are stores at each node. If a node is assigned an item x then only those transactions are stored which have x. Moreover, if x is not the first item in revised dataset then subset of transaction, i.e., from x to end of the transaction will be stored.

The method DataGenerator() for node data generation is depicted in Algorithm 3. It takes two inputs: *revisedData* and *itemNode* and, produces a paired RDD $NodeTransaction$ <$NodeID$, *transaction*> as an output. For each transaction t_p in *revisedData*, algorithm searches each item x of t_p in *itemNode* and grabs the NodeID (lines 1–5). If $NodeID$ has not been delivered as output then subset of t_p from the item x to the end of the t_p is associated with $NodeID$ to make a key-value pair and given in the output (lines 6–10).

Algorithm 2. Search space division: mapping()

```
INPUT:   $itemAUUB$: List of items along with AUUB values
         $N$ : Number of nodes
OUTPUT:  $itemNode$: A hashmap that maps each item to a NodeID
1:  $itemNode$ = empty hashmap
2:  $len$=Length of $itemAUUB$
3:  for each item $x$ in $itemAUUB$ do
4:      $weight = 2^{len-1} + x.AUUB$
5:      NodeID=-1
6:      $minimum = Int.MaxValue$
7:      for each node $n$ in $N$ do
8:          if $nodeWeight(n)$ is equal to 0 then
9:              $NodeID = n$
10:             break
11:         else
12:             $afterWeight = nodeWeight(n) + weight$
13:             if $nodeWeight(n)$ is lesser than $minimum$ then
14:                 $minimum = afterWeight$
15:                 $NodeID = n$
16:             end if
17:         end if
18:     end for
19:     $itemNode \leftarrow (item, NodeID)$
20:     $nodeWeight(NodeID) + = weight$
21:     $len = len$ -1
22: end for
23: return $itemNode$
```

3.3 HAUI Mining

The HAUIMiner() procedure is similar as described in HAUI-Miner [18]. However, Each cluster node runs its own HAUIMiner() for the assigned search space. The Pseudo code for the HAUI mining method is depicted in Algorithm 4. Initially, P is set to null and $extensionOfP$ is assigned the AU-list of all 1-itemset. For assigned items to a node, the search space is explored in top-down manner. If the itemset X from the $extensioOfP$ have average-utility more than $minUtil$ then it is assigned to output set $HAUI$ (lines 3–4). Then, pruning strategy is applied such that, if $X.AUUB = SUM(X.TMU) \geq minUtil$ then extensions of X are explored in the search tree (line 6). Each itemset Y after X in $extensionOfP$ is concatenated with X and a construct() method is called to build the AU-list of itemset XY (lines 7–9). Construct() method is not separately defined in this paper, as it is already given in HAUI-Miner [18]. Thereafter, HAUIMiner() method is called recursively to explore the extensions of itemset XY (line 11). At the end of the algorithm, a set of HAUIs is discovered.

Algorithm 3. Node data generation: DataGenerator()

```
INPUT:   $revisedData$: Revised dataset
         $itemNode$: mapping of item to NodeID
OUTPUT:  $NodeTransaction$: 0 or more $<NodeID, transaction>$
 1: flatMap ($revisedData$)
 2: for each transaction $t_p$ in $revisedData$ do
 3:     $added \leftarrow emptyhasmap$
 4:     for each item $x$ in each $t_p$ do
 5:         search item $x$ in $itemNode$ and grab its $NodeID$
 6:         if $NodeID$ is not in $added$ then
 7:             $added \leftarrow NodeID$
 8:             $T \leftarrow$ subset of transaction $t_p$ from item $x$ to the end
 9:             output $(NodeID, T)$
10:         end if
11:     end for
12: end for
13: end flatMap
```

4 Experiments and Results Analysis

In order to evaluate the performance of PHAUIM, a comparison with HAUI-Miner [18] is performed. All the experiments were conducted on a Spark cluster of six nodes. Four nodes were having CPU Xeon(R) CPU E3-1225 v5 clocked @ 3.30 GHz and two nodes were configured with core i7-7700HQ clocked @ 3.8 GHz. Each node was equipped with 8 GB of DDR4 RAM and 2 TB of hard drive. Each node was having following software modules: Ubuntu 16.04 OS, Spark version 2.2.1, Java v-8.01 and Scala v-1.12.4. PHAUIM is evaluated in terms of run time performance and scalability.

4.1 Datasets

Both the algorithms were run for four distinct real-life datasets: accidents, chess, retail and mushroom. Attributes of each dataset are shown in Table 3. All the datasets were taken from the SPMF library [6].

Table 3. Datasets

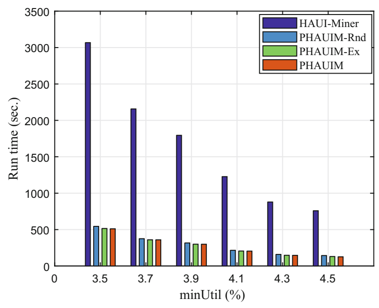
Dataset	# of transactions	# of items	*avg.len.*	Type
Accidents	340,183	468	33.8	Dense
Chess	3,196	75	37	Dense
Retail	88,162	16,407	10.3	Sparse
Mushroom	8,124	119	23	Dense

Algorithm 4. HAUI Mining on node: HAUIMiner()

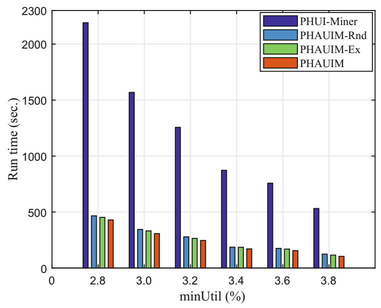
```
INPUT:  $P$: AU-list of itemset P, $extensionOfP$: AU-list of all 1-extensions of P,
        $itemNode$: mapping of item to NodeID, $minUtil$: Minimum
        average-utility threshold, $NodeID$: Node ID of current node,
        $length$: length of current itemset
OUTPUT: $HAUI$ : A set of high average-utility itemsets with P prefix
 1: for each AU-list $X$ in $extensionOfP$ do
 2:     if $P$ is not empty OR $itemNode(X.itemset)$ is equal to $NodeID$ then
 3:         if $\frac{SUM(X.U)}{length} \geq minUtil$ then
 4:             $HAUI \leftarrow HAUI \cup X$
 5:         end if
 6:         if $SUM(X.TMU) \geq minUtil$ then
 7:             $exAUL \leftarrow Null$
 8:             for each AU-list $Y$ in $extensionOfP$ do
 9:                 $exAUL \leftarrow exAUL \cup construct(P.AUL, X, Y)$
10:             end for
11:             HAUIMiner$(X, exAUL, minUtil, itemNode, NodeID, length + 1)$
12:         end if
13:     end if
14: end for
```

4.2 Runtime Performance

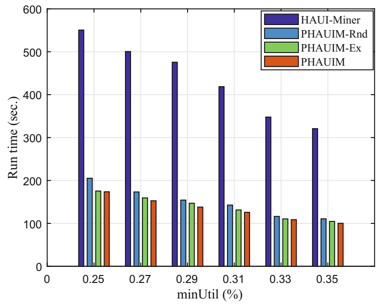
Run time performance of PHAUIM is evaluated by comparing it with traditional HAUI-Miner. Moreover, to measure the performance of search space division strategy, three distinct variations of PHAUIM were implemented: PHAUIM-Rnd, PHAUIM-Ex and PHAUIM. PHAUIM-Rnd divides the search space items in random manner. On the other hand, PHAUIM-Ex includes the existing method discussed in PHUI-Miner [5], while PHAUIM includes the proposed search space division technique. All three provide same functionality with different search space division technique. Run time of both the algorithms was measured with respect to different value of *minUtil*, as depicted in Fig. 3. It can be observed from the results that each version of PHAUIM outperforms the HAUI-Miner. The reason is that, HAUI-Miner performs all the computation on a single machine while PHAUIM assigns the workload to multiple nodes to search the HAUIs faster. PHAUIM-Rnd takes more time to terminate than the other two PHAUIM-Ex and PHAUIM. Random division of search space may assign some nodes more workload than others. Therefore, such nodes produces the results lately than other nodes and increase job completion time. PHAUIM-Ex takes slightly more time than the PHAUIM. PHAUIM handles the items with large gap in AUUB value and make a fair assignment. On the other hand, there is a fix pattern of item assignment in PHAUIM-Ex. It is also observed that PHAUIM shows the best performance for chess and accidents datasets.



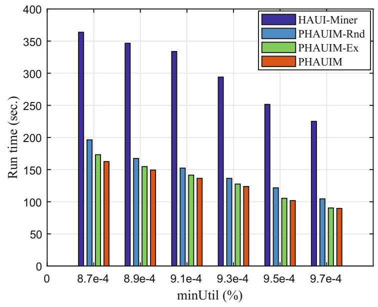
(a) chess dataset



(b) accidents dataset



(c) mushroom dataset

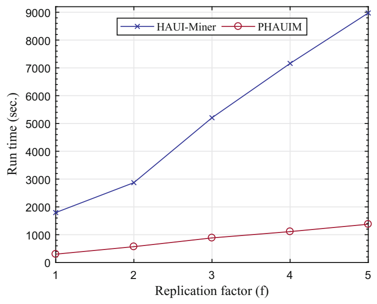


(d) retail dataset

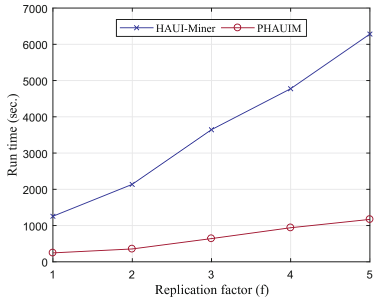
Fig. 3. Run time performance of PHAUIM

4.3 Scalability Test

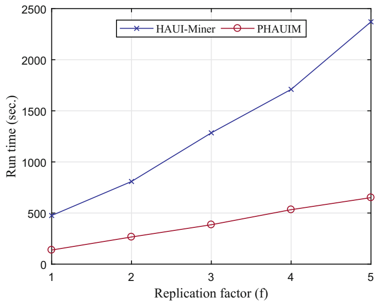
Scalability test is performed to analyze nature of both the algorithms while data size is linearly increased. To evaluate the scalability of PHAUIM, both the algorithms were run by varying the data size in a certain proportion. The run time of each algorithm was measured to check the performance. Each dataset is replicated by a factor f such that experiment data = original data $\times f$, where $f = 1, 2, .., 5$. Both the HAUI-Miner and PHAUIM were run with replicated benchmark datasets. The results were taken with a fixed value of $minUtil$ for each dataset. Value of $minUtil$ for chess, accidents, mushroom and retail dataset was set to respectively 3.9%, 3.2%, 0.29 % and 9.1e−4 % of the total dataset utility. The results are depicted in Fig. 4. From the results, it can be noticed that, with increase in data size, run time also grows linearly. The reason is that, with the increase in number of transactions, there are more entries in the AU-list of itemset and requires additional run time. Run time of the HAUI-Miner for all datasets increases sharply with rise in data size. In contrast, PHAUIM run time increases slowly and remains close to x-axis.



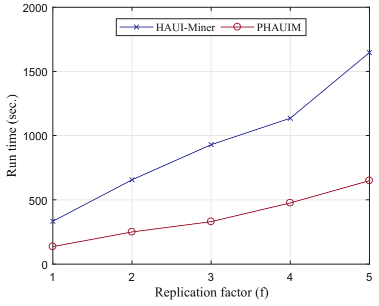
(a) chess dataset



(b) accidents dataset



(c) mushroom dataset



(d) retail dataset

Fig. 4. Scalability test of PHAUIM

5 Conclusion

This paper presents a detailed study about the pattern mining approaches such as HUI (High utility itemset) mining and HAUI (High-average utility itemset) mining. With the introduction to big data, pattern mining has acquired many challenges such as searching of high average-utility itemsets from a big transaction dataset. Huge number of candidate itemsets are generated while HAUI searching which requires massive computing resources and memory. Standalone system is not suitable to process big data in efficient manner. In this paper, a parallel version of traditional HAUI-Miner called PHAUIM is proposed to discover HAUIs from big transaction datasets. PHAUIM is a Spark-based distributed algorithm that model the HAUI-Miner to work in parallel on a multi node cluster. Search space is divided into subspaces and distributed to cluster nodes. Each node individually processes its subspace and produces a set of HAUIs. A novel search space division method is also presented that considers the AUUB of items along with the subspace size to divide the search space fairly. Various experiments have been conducted to evaluate the performance of PHAUIM with respect to four benchmark datasets. Moreover, it is also compared with HAUI-Miner. Experimental results show that PHAUIM outperforms the HAUI-Miner with a huge margin.

Acknowledgment. This research work is supported by Indian Institute of Technology (ISM), Dhanbad. The authors wish to express their gratitude and heartiest thanks to the Department of Computer Science & Engineering, Indian Institute of Technology (ISM), Dhanbad, India for providing their research support.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. ACM SIGMOD Rec. **22**, 207–216 (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
3. Chan, R., Yang, Q., Shen, Y.-D.: Mining high utility itemsets. In: 2003 Third IEEE international conference on Data mining, ICDM 2003, pp. 19–26. IEEE (2003)
4. Chen, C.L.P., Zhang, C.-Y.: Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf. Sci. **275**, 314–347 (2014)
5. Chen, Y., An, A.: Approximate parallel high utility itemset mining. Big Data Res. **6**, 26–42 (2016)
6. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Cheng-Wei, W., Tseng, V.S.: SPMF: a Java open-source pattern mining library. J. Mach. Learn. Res. **15**(1), 3389–3393 (2014)
7. Fournier-Viger, P., Lin, J.C.-W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Sci. Pattern Recognit. **1**(1), 54–77 (2017)
8. Fournier-Viger, P., Lin, J.C.-W., Vo, B., Chi, T.T., Zhang, J., Le, H.B.: A survey of itemset mining. Wiley Interdisc. Rev.: Data Mining Knowl. Discov. **7**(4), e1207 (2017)
9. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS (LNAI), vol. 8502, pp. 83–92. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08326-1_9
10. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier, Amsterdam (2011)
11. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM SIGMOD Rec. **29**, 1–12 (2000)
12. Hong, T.-P., Lee, C.-H., Wang, S.-L.: Effective utility mining with the measure of average utility. Expert Syst. Appl. **38**(7), 8259–8265 (2011)
13. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Syst. Appl. **42**(5), 2371–2381 (2015)
14. Krishnamoorthy, S.: HMiner: efficiently mining high utility itemsets. Expert Syst. Appl. **90**, 168–183 (2017)
15. Lan, G.-C., Hong, T.-P., Tseng, V.S.: Efficiently mining high average-utility itemsets with an improved upper-bound strategy. Int. J. Inf. Technol. Decis. Making **11**(05), 1009–1030 (2012)
16. Lan, G.-C., Hong, T.-P., Tseng, V.S., et al.: A projection-based approach for discovering high average-utility itemsets. J. Inf. Sci. Eng. **28**(1), 193–209 (2012)
17. Li, Y.-C., Yeh, J.-S., Chang, C.-C.: Isolated items discarding strategy for discovering high utility itemsets. Data Knowl. Eng. **64**(1), 198–217 (2008)
18. 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. Adv. Eng. Inform. **30**(2), 233–243 (2016)

19. Lin, J.C.-W., Ren, S., Fournier-Viger, P.: MEMU: more efficient algorithm to mine high average-utility patterns with multiple minimum average-utility thresholds. IEEE Access **6**, 7593–7609 (2018)
20. Lin, Y.C., Wu, C.-W., Tseng, V.S.: Mining high utility itemsets in big data. In: Cao, T., Lim, E.-P., Zhou, Z.-H., Ho, T.-B., Cheung, D., Motoda, H. (eds.) PAKDD 2015. LNCS (LNAI), vol. 9078, pp. 649–661. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18032-8_51
21. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, pp. 55–64. ACM (2012)
22. Liu, Y., Liao, W., Choudhary, A.: A fast high utility itemsets mining algorithm. In: Proceedings of the 1st International Workshop on Utility-Based Data Mining, pp. 90–99. ACM (2005)
23. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
24. Sethi, K.K., Ramesh, D.: HFIM: a Spark-based hybrid frequent itemset mining algorithm for big data processing. J. Supercomput. **73**(8), 3652–3668 (2017)
25. Sethi, K.K., Ramesh, D., Edla, D.R.: P-FHM+: parallel high utility itemset mining algorithm for big data processing. Procedia Comput. Sci. **132**, 918–927 (2018). International Conference on Computational Intelligence and Data Science
26. Tseng, V.S., Wu, C.-W., Shie, B.-E., Yu, P.S.: 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, pp. 253–262. ACM (2010)
27. White, T.: Hadoop: The Definitive Guide. O'Reilly Media, Inc., Newton (2012)
28. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: Proceedings of the 2004 SIAM International Conference on Data Mining, pp. 482–486. SIAM (2004)
29. Yun, U., Kim, D.: Mining of high average-utility itemsets using novel list structure and pruning strategy. Future Gener. Comput. Syst. **68**, 346–360 (2017)
30. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, pp. 2. USENIX Association (2012)
31. Zida, S., Fournier-Viger, P., Lin, J.C.-W., Wu, C.-W., Tseng, V.S.: EFIM: a highly efficient algorithm for high-utility itemset mining. In: Sidorov, G., Galicia-Haro, S.N. (eds.) MICAI 2015. LNCS (LNAI), vol. 9413, pp. 530–546. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27060-9_44