COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 04/06 pp 47−58
Volume 3, Number 1, March 2023
DOI: 10.23919/CSMS.2022.0023

A Parallel High-Utility Itemset Mining Algorithm Based on Hadoop

Zaihe Cheng, Wei Shen, Wei Fang*, and Jerry Chun-Wei Lin

Abstract: High-utility itemset mining (HUIM) can consider not only the profit factor but also the profitable factor, which is an essential task in data mining. However, most HUIM algorithms are mainly developed on a single machine, which is inefficient for big data since limited memory and processing capacities are available. A parallel efficient high-utility itemset mining (P-EFIM) algorithm is proposed based on the Hadoop platform to solve this problem in this paper. In P-EFIM, the transaction-weighted utilization values are calculated and ordered for the itemsets with the MapReduce framework. Then the ordered itemsets are renumbered, and the low-utility itemsets are pruned to improve the dataset utility. In the Map phase, the P-EFIM algorithm divides the task into multiple independent subtasks. It uses the proposed S-style distribution strategy to distribute the subtasks evenly across all nodes to ensure load-balancing. Furthermore, the P-EFIM uses the EFIM algorithm to mine each subtask dataset to enhance the performance in the Reduce phase. Experiments are performed on eight datasets, and the results show that the runtime performance of P-EFIM is significantly higher than that of the PHUI-Growth, which is also HUIM algorithm based on the Hadoop framework.

Key words: pattern mining; data mining; Hadoop; parallel; high-utility itemset mining; big data

1 Introduction

Data mining is the process of uncovering useful patterns from a collection of data, which can be used in a wide range of applications, such as fault diagnosis[1], semiconductor manufacturing system[2], and manufacturing scheduling[3, 4]. Frequent itemset mining (FIM) is used to identify the frequent item combinations to facilitate decision-making[5−8], which is one of the earliest data mining tasks. However, the FIM algorithm focuses only on the occurrence frequency of a product or item, ignoring its value or profit. In other words, the most frequent pattern may not be valuable or useful. For instance, frequent items in supermarkets, e.g., eggs and apples, have very low-profit margins. These are not the items decision-makers want, even if they occur frequently. The FIM task was upgraded to the high-utility itemset mining (HUIM) task that can identify item combinations with high-profit margins. The HUIM algorithm focuses not only on the frequency of occurrence of an item but also on its quantity value, as well as the the item’s unit profit value. In FIM, the itemset support is less than an extended itemset, which is known to be anti-monotonic and helps quickly reduce the search space. However, since more factors are considered in the HUIM, the anti-monotonic property is not held in the mining progress. Consequently, the relationship between two utilities of an itemset and one of the extended itemsets

• Zaihe Cheng is with the School of Internet of Things, Wuxi Institute of Technology, Wuxi 214121, China and also with the School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi 214122, China.
• Wei Shen and Wei Fang are with the Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Jiangnan University, Wuxi 214122, China. E-mail: fangwei@jiangnan.edu.cn.
• Jerry Chun-Wei Lin is with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen 5020, Norway.
∗ To whom correspondence should be addressed.
※ This article was recommended by Executive Editor-in-Chief Ling Wang.
Manuscript received: 2022-07-13; revised: 2022-10-20; accepted: 2022-10-31

is complex, increasing the complexity of the mining task. Hence, most HUIM algorithms must redefine new upper bounds to reduce search spaces. Several HUIM algorithms have been derived, such as high temporal fuzzy utility pattern mining[9], sequential itemset mining[10, 11], negative utility itemset mining[12], top-k itemset mining[13−15], closed high-utility itemset mining[16, 17], fuzzy utility mining[18], utility-oriented pattern mining[19], and targeted high-utility itemset querying[20].

An HUIM algorithm should mainly address the following three difficulties in general, a large search space, too much memory usage to run the algorithm, and long runtime consumption to calculate the utility values of itemsets. Depth-first search and breadth-first search are the main approaches in existing HUIM algorithms. The breadth-first search based algorithms mainly include UMing[21], two-phase[22], and IIDS[23]. The UMing algorithm was the first to introduce the HUIM concept and the associated solution. The Apriori framework is used to traverse the search space in UMing. The utility values of the n-itemsets are estimated based on the former ($n-1$)-itemsets. The candidate itemsets are therefore generated. The transaction-weighted utilization (TWU) of downward closure is extensively used in two-phase HUIM models. In the first phase of the method, ($k-1$) candidate sets are used to generate k candidate sets. The algorithm evaluates the utility values of the generated candidate itemsets in the second phase in order to mine high-utility itemsets (HUIs) or the qualified candidate itemsets. Once no candidate itemset is formed in the first stage, the algorithm is terminated, and the final results are generated.

Two-phase and one-phase algorithms are the main types based on depth-first search. For a two-phase algorithm, the algorithm generates candidate itemsets by overestimating the utility values in the database in the first phase. In the second phase, the algorithm calculates the actual utility values by visiting the database. Two-phase algorithms include IHUP[24], CHUI-Mine[13], MU[25], UP-Growth[26], and UP-Growth+[27]. IHUP was the first to introduce a tree-based structure, called IHUP-tree. The IHUP-tree is used to generate the candidate itemsets. The IHUP algorithm constructs the IHUP-tree by scanning the database twice. The first scan traverses the IHUP-tree upward by extending itemsets. The second scan considers the itemsets with utility values larger than the pre-defined threshold as candidate itemsets to reduce the candidate itemsets.

In one-phase HUIM algorithms, the search space is directly traversed according to the order of itemsets. Moreover, the actual utility values of the itemsets are computed without generating candidate itemsets. In Ref. [28], the first one-phase HUIM algorithm was introduced, which is named HUI-Miner. In HUI-Miner, the utility values of the itemsets are directly calculated based on the utility-list structure. Since then, improved one-phase based algorithms were proposed, e.g., FHM[29], HUP-Miner[30], CHUI-Mine[13], HUI-Miner[28], CHUM[31], d2HUP[32], EFIM[15], BAHUI[10], CHUI-Miner[33], PB[34], UFH[35], ULB-Miner[29], and Hminer[14]. In Ref. [15], efficient high-utility itemset mining (EFIM) was introduced, which uses the merging of transaction sets to reduce the dataset size of each item effectively. In EFIM, the subtree and local utilities are regarded as the upper bounds, reducing the search space and improving the runtime and memory usage performance. EFIM is currently one of the fastest HUIM algorithms.

The rapid evolution of information technology has seen a drastic increase in data size. HUIM algorithms are no longer suitable for larger datasets due to the inefficiency of a single computer's CPU processing speed and memory size. Hence, some scholars deployed data mining algorithms in distributed cluster systems to improve their runtime performance and apply them to larger datasets. PHUI-Growth[12] is a MapReduce-based HUIM algorithm based on the MapReduce computing framework. In PHUI-Growth, an iterative parallel structure using breadth-first search is proposed. The search space is reduced by the DLU-MR pruning strategy in the second phase, which can address the issue that a single computer cannot mine large datasets. However, due to the iterative parallel structure and low-performance pruning strategy, the PHUI-Growth runtime remains very long. Therefore, a novel parallel EFIM (P-EFIM) is proposed in this paper. First, the TWU value of each itemset is calculated, and then the dataset is pruned and recoded using these ordered itemsets to improve the dataset utility. Second, the algorithm divides the entire task into multiple subtasks using the MapReduce framework and applies the S-style balancing strategy to distribute the subtasks across all nodes to ensure as close to load-balancing as possible. Finally, the P-

EFIM algorithm applies the one-phase EFIM[15] algorithm based on the depth-first search to mine each subtask on each node rather than using the PHUI-Growth. Additionally, the performance of the proposed P-EFIM is improved by the efficient pruning strategy in EFIM.

2 Definition

2.1 HUIM-related definitions

A transaction dataset $D=\{T_1,T_2,\ldots,T_n\}$ comprises multiple transactions. The T_{ID} refers to a unique identifier, which is associated with each transaction, where $I=\{l_1,l_2,\ldots,l_n\}$ is the set that contains all nonduplicate items in D. Both internal utility value and external utility value are assigned to each item in each transaction. An example transaction database with five transactions is given in Table 1[15]. Table 2 gives the external utility values of seven items.

Definition 1 (Utility of an item/itemset) In transaction T_c, $q(i,T_c)$ denotes the internal utility of item i, and $p(i)$ denotes the external utility value in this paper. $u(i,T_c)=p(i)\times q(i,T_c)$ means the utility value of item i. In transaction T_c, $u(X,T_c)$ denotes the utility value of itemset X, and is calculated as $u(X,T_c)=\sum_{i\in X}u(i,T)$. In the entire transaction database, $u(X)=\sum_{T_c\in g(X)}u(X,T_c)$ is used to calculate the utility value of the itemset X, where $g(X)$ represents all the transactions that include itemset X.

Take item b in transaction T_3 as an example, its utility value is $u(b,T_3)=2\times 2=4$. For the itemset $\{a,b\}$ in T_3, its utility value is $u(\{a,b\},T_3)=u(a,T_3)+u(b,T_3)=5\times 1+2\times 2=9$. Considering the entire transaction database, the utility value of the itemset $\{a,d\}$ is $u(\{a,d\})=u(\{a,d\},T_1)+u(\{a,d\},T_3)=u(a,T_1)+u(d,T_1)+u(a,T_3)+u(d,T_3)=5+2+5+12=24$.

Table 1 Transaction dataset.

T_{ID}	Transaction
T_1	$(a,1)(c,1)(d,1)$
T_2	$(a,2)(c,6)(e,2)(g,5)$
T_3	$(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)$
T_4	$(b,4)(c,3)(d,3)(e,1)$
T_5	$(b,2)(c,2)(e,1)(g,2)$

Table 2 External utilities.

Item	Profit
a	5
b	2
c	1
d	2
e	3
f	1
g	1

Definition 2 (HUI) An HUI refers to the itemset X with the utility value greater than the user defined threshold. An HUIM algorithm is used to find all HUIs.

For example, the threshold is minutil = 30; In Table 1, since $u\{b,d\}=30$, $u\{a,c,e\}=31$, $u\{b,c,d,e\}=40$, and $u\{a,b,c,d,e,f\}=30$, these itemsets are HUIs.

Definition 3 (TWU) For transaction T_c, its utility value is $\mathrm{TU}(T_c)=\sum_{x\in T_c}u(x,T_c)$. For the itemset X, its TWU value is $\mathrm{TWU}(X)=\sum_{T_c\in g(X)}\mathrm{TU}(T_c)$, which is the sum of the transaction utility values of all transactions containing itemset X.

For the itemset $\{g\}$, its TWU value is $\mathrm{TWU}[g]=\mathrm{TU}[T_2]+\mathrm{TU}[T_5]=10+6+6+5+4+2+3+2=38$.

Theorem 1 (TWU pruning) If the TWU value of any itemset X is less than the threshold minutil, then the itemset and its superset are low-utility itemsets[22].

Definition 4 (Utility value of remaining itemsets) Suppose $\succ$ is an ordering scheme for I, and X is a subset. The remaining itemsets refer to the itemsets appearing after the itemset X in a transaction and its utility value is $\mathrm{re}(X,T_c)=\sum_{i\in T_c\wedge(i\succ x,\forall x\in X)}u(i,T_c)$.

Take the remaining itemsets of itemset $\{a,c\}$ in transaction T_3 as an example, $\mathrm{re}(\{a,c\},T_2)=u(e,T_2)+u(g,T_2)=6+5=11$.

Definition 5 (Utility-list structure) The tuples $(T_c,\mathrm{iutil},\mathrm{rutil})$ represent the utility-list structure of itemset X. T_c represents the transaction containing the itemset X. iutil and rutil are the utility values of itemset X in T_c $(u(X,T_c))$.

Given the order I as $\{a,b,c,d,e,f,g\}$, the utility-list structure of itemset $\{a,c\}$ is $\{(T_1,6,2),(T_2,16,11),(T_3,6,23)\}$.

2.2 Key definitions of the EFIM algorithm

The EFIM is currently one of the fastest HUIM algorithms with two effective strategies: merging transactions containing identical items and subtree and local utilities. Merging transactions containing identical items effectively reduce each item's dataset size and help reduce the runtime. The subtree and local utilities are used as two pruning upper bounds to reduce the runtime and the search space.

Definition 6 (Transactions containing identical items) Each item or itemset has its transaction set D_i. The utility values of identical itemsets in the transaction set D_i are summed. That is

$$q(i,T_m)=\sum_{i\in T\in D_i} q(i,T) \tag{1}$$

Definition 7 (Subtree utility) Given an itemset α, and let z be a subset of α. The utility value of this subset is

$$\mathrm{su}(\alpha,z)=\sum_{T\in g(\alpha\cup z)} (u(\alpha,T)+u(z,T)+\mathrm{re}(\alpha,z,T)) \tag{2}$$

Definition 8 (Local utility) Given an itemset α, and let z be a subset of α. The local utility value of this subset is

$$\mathrm{lu}(\alpha,z)=\sum_{T\in g(\alpha\cup z)} (u(\alpha,T)+\mathrm{re}(\alpha,T)) \tag{3}$$

3 P-EFIM Algorithm

To solve the problem that a single computer is difficult to mine large datasets, we propose the parallel HUIM (P-EFIM) algorithm. The P-EFIM algorithm leverages the MapReduce computing framework to mine HUIs efficiently. The key strategy of P-EFIM algorithm is to decompose a dataset into multiple smaller datasets through HDFS and distribute them to different nodes. The proposed P-EFIM algorithm mainly comprises four phases, namely, TWU value ordering, dataset recoding, data decomposition, and data mining. Figure 1 shows the flowchart of the proposed P-EFIM algorithm. In the TWU values ordering phase, the TWU value of each item is calculated in each node and ordered in ascending order with the Map and Reduce operators. In the dataset recoding phase, the algorithm recodes the datasets using the order of the itemsets after pruning and ordering, and prunes the low-utility itemsets. A new dataset is obtained after the recoding phase. In the data decomposition phase, the algorithm decomposes the sorted data into multiple datasets required for each subtask. The decomposition is realized based on the S-style distribution strategy in order to distribute the subtasks among the computing nodes with a relatively balanced load for each node. In the data mining phase, EFIM algorithm is used to mine each subtask on each node individually and then get all HUIs through the collect operator.

3.1 TWU values ordering and datasets recoding

According to Theorem 1, if an itemset is a low-utility itemset, then none of the itemsets containing this itemset are HUIs. Hence, calculating the TWU value of each item and ordering them in ascending order help to reduce the search space[24]. Additionally, pruning the first n items whose TWU values are less than the threshold, and recoding the transaction sets using the mapping table created with the threshold reduce the size of the original dataset. By using the data in Table 1, we can then obtain the TWU values of all items in the database. For example, TWU(a) is calculated as: TWU(a)(= 65); TWU(b) is calculated as: TWU(b)(= 61); TWU(c) is calculated as: TWU(c)(= 66); TWU(d) is calculated as: TWU(d)(= 58); TWU(e) is calculated as: TWU(e)(= 61); TWU(f) is calculated as: TWU(f)(= 30); and TWU(g) is calculated as: TWU(g)(= 38). The ascending order of them is $I=\{f, g, d, b, e, a, c\}$. The items f and g can be pruned if the defined threshold is 40, and the remaining items are $I=\{d, b, e, a, c\}$.

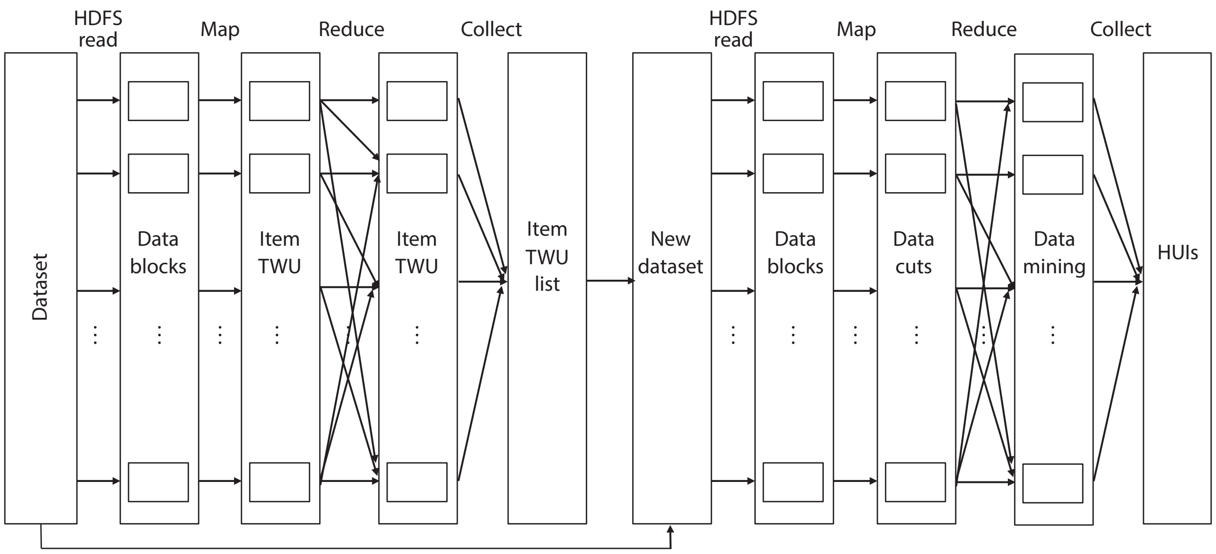

Fig. 1 Flowchart of the proposed P-EFIM algorithm.

The dataset recoding process mainly comprises two phases. In phase one, the algorithm creates an ordered mapping table, as shown in Table 3, using the itemsets that appear after pruning and ordering, e.g., $I = \{d, b, e, a, c\}$. In phase two, the algorithm reconstructs the datasets using the mapping table. It replaces the data in the original datasets with the data in the mapping table, prunes the data absent in the mapping table, and orders the new datasets. The reconstructed datasets are shown in Table 4. The dataset recoding process has two features. First, the datasets with the TWU values less than the threshold are pruned in order to reduce the datasets, memory size, and runtime used by the algorithm. Second, the recoded datasets are ordered, thereby reducing the runtime of the algorithm through binary search.

In Table 4, the transaction set of the itemset A, E is $\langle 7, E, 1\rangle$, $\langle 17, E, 1\rangle$. If two or more transactions contain identical itemsets and equal numbers of itemsets, e.g., E, both transactions can be merged to obtain the set $\langle 24, E, 2\rangle$. Merging transactions containing identical itemsets can reduce transactions, thereby reducing the time spent on traversing the dataset and runtime of the algorithm. Generally, the denser a database, the more transactions can be merged, and the more time can be saved.

In Table 3, suppose itemset $\alpha = C$, then the subsets of C are D and E, and $\text{su}(\alpha, D) = u(C, T_2) + \text{re}(\alpha, C, T_2) + u(\alpha, T_3) + u(D, T_2) + u(D, T_3) + \text{re}(\alpha, D, T_3) =$ 6+6+3+10+5+1=31, $\text{su}(\alpha, E) = u(\alpha, T_5) + u(E, T_5) + u(E, T_3) + u(\alpha, T_4) + u(E, T_4) + u(\alpha, T_2) + u(E, T_2) + u(\alpha, T_3)$ = 3+2+1+3+3+6+6+3 = 27. The subtree utility value is applied to the subnodes of the current node, while the local utility value is applied to all subtrees of the tree where the current node is located. Both pruning strategies are used to reduce the runtime and search space.

Table 3 Mapping table.

Item	New item
d	A
b	B
e	C
a	D
c	E

Table 4 Recoded datasets.

Item	Items data (ITD)
T_1	$(A,2)(D,5)(E,1)$
T_2	$(C,6)(D,10)(E,6)$
T_3	$(A,12)(B,4)(C,3)(D,5)(E,1)$
T_4	$(A,6)(B,8)(C,3)(E,3)$
T_5	$(B,4)(C,3)(E,2)$

3.2 Data decomposition

The search space for HUIM is the arrangement of all itemsets. The EFIM algorithm searches the space from top to bottom in an in-depth manner, whereas the P-EFIM algorithm searches the space by dividing the entire tree into multiple subtrees and distributing these subtrees across computing nodes to run the EFIM algorithm. The data decomposition process comprises two phases. In phase one, the algorithm splits the original data for multiple subtasks. In phase two, the algorithm distributes the subtasks across computing nodes to ensure load-balancing.

3.2.1 Data splitting

The data splitting phase corresponds to the map phase in the MapReduce framework, during which the algorithm converts each transaction set into the $\langle K, V\rangle$ format. The algorithm splits the transaction set by items. Here K is the value of each item and V is the data of the item with the utility value of K, and the set of combinations of the utility values of the items after K. The detailed process is illustrated in Algorithm 1. The data splitting process is illustrated in Fig. 2. The transaction sets in Mapper 1 are $(A,2)(D,5)(E,1)$, and the map converts them to three $\langle K,V\rangle$ transaction sets, namely, $\langle A, \{2, (D,5), (E,1)\}\rangle$, $\langle D, \{5, (E,1)\}\rangle$, $\langle E, \{1\}\rangle$. After splitting, the merged datasets in Reduce 1 with equal K are the datasets required for the subtrees. For example, the dataset in Reduce 1 with $K = \langle A\rangle$ is the dataset for root node $\langle A\rangle$.

3.2.2 Load-balancing strategy

The most important aspect of distributed parallel computing is load-balancing, which entails balancing the load (a task) by distributing it across multiple computing nodes to complete the task collaboratively. When it comes to load-balancing, our concern is on how to distribute subtrees, which are more than nodes, to the computing nodes and ensure a relatively

Algorithm 1 Map algorithm

Input: D: A transaction database;

Output: The itemset after map;

```
1: for it = 0; $i < D$.length; it++ do
2:     $T = D$[it]
3:     for il = 0; il < $T$.length; il++ do
4:         output($T$[it],$T$.substring(il))
5:     end for
6: end for
```

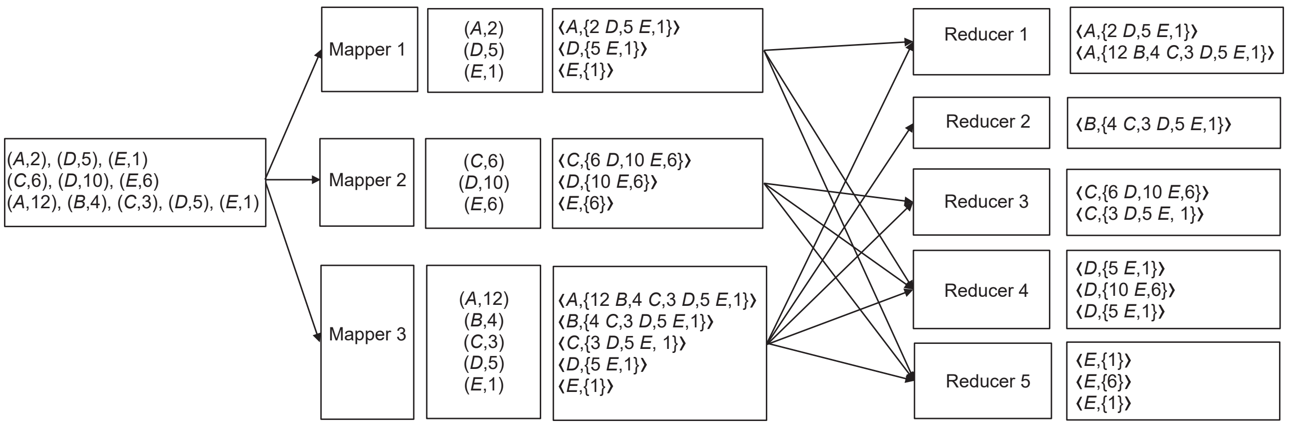
(A,2), (D,5), (E,1)
(C,6), (D,10), (E,6)
(A,12), (B,4), (C,3), (D,5), (E,1)
Mapper 1
Mapper 2
Mapper 3
(A,2) (D,5) (E,1)
(C,6) (D,10) (E,6)
(A,12) (B,4) (C,3) (D,5) (E,1)
⟨A,{2 D,5 E,1}⟩ ⟨D,{5 E,1}⟩ ⟨E,{1}⟩
⟨C,{6 D,10 E,6}⟩ ⟨D,{10 E,6}⟩ ⟨E,{6}⟩
⟨A,{12 B,4 C,3 D,5 E,1}⟩ ⟨B,{4 C,3 D,5 E,1}⟩ ⟨C,{3 D,5 E, 1}⟩ ⟨D,{5 E,1}⟩ ⟨E,{1}⟩
Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
⟨A,{2 D,5 E,1}⟩ ⟨A,{12 B,4 C,3 D,5 E,1}⟩
⟨B,{4 C,3 D,5 E,1}⟩
⟨C,{6 D,10 E,6}⟩ ⟨C,{3 D,5 E, 1}⟩
⟨D,{5 E,1}⟩ ⟨D,{10 E,6}⟩ ⟨D,{5 E,1}⟩
⟨E,{1}⟩ ⟨E,{6}⟩ ⟨E,{1}⟩

Fig. 2 Data decomposition process.

balanced load for each node. By default, the data in ⟨*K*, *V*⟩ format generated in the data splitting process are distributed using the hash value of *K*. This distribution strategy is highly random, and thus, the computing loads of the nodes may differ significantly. From Fig. 3, we observe that the loads of the subtrees are roughly in descending order. Hence, an S-style distribution strategy is introduced in P-EFIM to arrange the subtrees and distribute them to the reduced computing nodes.

For example, the subtrees in Fig. 3 are $\{A,B,C,D,E\}$. If there are five parallel Reduces, the distribution is: $\langle \text{reduce1},\{A\}\rangle,\langle \text{reduce2},\{B\}\rangle,\langle \text{reduce3},\{C\}\rangle,\langle \text{reduce4},\{D\}\rangle$, and $\langle \text{reduce5},\{E\}\rangle$. If there are two parallel Reduces, the distribution is: $\langle \text{reduce1},\{A,D,E\}\rangle$, and $\langle \text{reduce2},\{B,C\}\rangle$.

3.3 Data mining

The data mining phase corresponds to the reduced phase in the MapReduce framework, during which the algorithm handles the data generated in the map phase (data decomposition phase), or mines the data generated in the reduced phase, as shown in Fig. 2, for all HUIs.

Definition 9 (Merging transactions containing identical items) Each item or itemset has its transaction set D_i. In the transaction set, the utility values of identical itemsets are summed. The utility value of itemset *i* is defined as: $q(i,T_m) = \sum i \in T \in D_i q(i,T)$.

In Table 4, the transaction set of itemsets $\{A,E\}$ is $\{\langle 7,\{E,1\}\rangle,\ \langle 17\{E,1\}\rangle\}$. If two or more transactions contain identical itemsets and equal numbers of itemsets, e.g., *E*, both transactions can be merged, and the merged transaction set is $\{\langle 24,\{E,\ 2\}\rangle\}$.

Definition 10 (Subtree utility) Consider the itemset α and let z be a subset of node α in the depth-first search, the utility value of this subset is calculated as $\text{su}\,(\alpha,z) = \sum_{T_c \in g(\alpha\cup\{z\})}(u(\alpha,T_c)+u(z,T_c)+\sum_{i\in T_c \cap i \succ z} u(i,T_c))$.

In Table 4, suppose the itemset $\alpha=\{C\}$, and the subset of *C* is $\{D,E\}$, then $\text{su}(\alpha,D)=u(C,T_2)+u(C,T_3)+u(D,T_2)+u(E,T_2)+u(D,T_3)+u(E,T_3)$=6+3+10+6+5+1=31, $\text{su}(\alpha,E)=u(C,T_2)+u(E,T_2)+u(C,T_3)+u(E,T_3)+u(C,T_4)+u(E,T_4)+u(C,T_5)+u(E,T_5)$ =6+6+3+1+3+3+3 + 2 = 27.

Definition 11 (Local utility) Consider the itemset α and let z be a subset of node α in the depth-first search. The local utility value of this subset is denoted as $\text{lu}(\alpha,z)$ and given as: $\text{lu}(\alpha,z)=\sum_{T_c\in g(\alpha\cup\{z\})}(u(\alpha,T_c)+\text{re}(\alpha,T_c))$.

In Table 4, suppose itemset $\alpha=\{B\}$, and the subset of *B* is $\{C,D,E\}$, then $\text{lu}(\alpha,C)=u(\alpha,T_3)+\text{re}(\alpha,T_3)+u(\alpha,T_4)+\text{re}(\alpha,T_4)+u(\alpha,T_5)+\text{re}(\alpha,T_5)$ = 4+9+8+6+4+5 = 36, $\text{lu}(\alpha,D)=u(\alpha,T_3)+\text{re}(\alpha,T_3)=4+6=10$, and $\text{lu}(\alpha,E)=u(\alpha,T_3)+\text{re}(\alpha,T_3)+u(\alpha,T_4)+\text{re}(\alpha,T_4)+u(\alpha,T_5)+\text{re}(\alpha,T_5)$ = 4+1+8+3+4+2 = 22 .

Algorithm 2 illustrates the underlying process. The mining process is in the Reduce phase of the distributed framework. The data output by Map in the Reduce phase, i.e., ⟨*Key*, *D*⟩ and threshold minutil serve as the input, and the identified HUIs are the output. The algorithm performs a depth-first search with Key as the

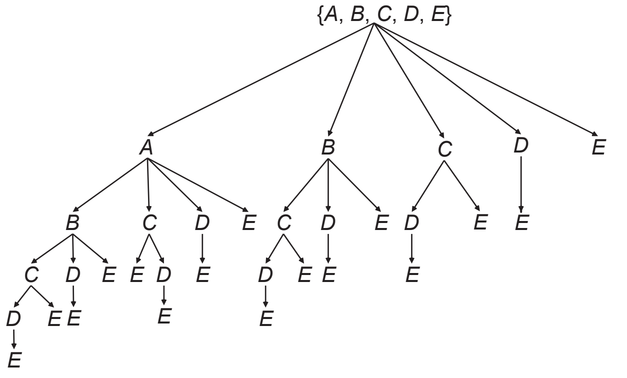

Fig. 3 An example search space.

Algorithm 2 Reduce algorithm

```
Input: D: a transaction database, Key: an item, and minutil: a user-specified threshold;
Output: The set of the HUIs;
1: $\alpha$ = Key, and sort transactions in $D$ according to $\succ$;
2: KeyPrimary($\alpha$) = $i|z \succ i \wedge \mathrm{su}(z,i) \geqslant$ minutil
3: for $j = 0$; $j <$ KeyPrimary($\alpha$).length; $j++$ do
4:     $\alpha$.add(KeyPrimary($\alpha$)[$j$])
5:     Primary($\alpha$) = $\{i|z \succ i \wedge \mathrm{su}(z,i) \geqslant$ minutil$\}$
6:     Secondary($\alpha$) = $\{i|z \succ i \wedge \mathrm{lu}(z,i) \geqslant$ minutil$\}$
7:     create $D$* according to $\alpha$ and D
8:     for $i = 0$; $i <$ rootWidthnode.length; $i++$ do
9:         Search($\alpha, D*$, Primary($\alpha$), Secondary($\alpha$))
10:        $\alpha$.removelast();
11:    end for
12: end for
```

root node, uses the subtree utility value to identify the subnodes of Key and traverses the subnodes. The algorithm reduces the number of nodes at the next layer using the subtree and local utility values. Next, the search algorithm performs the depth-first search using a recursive approach[15].

4 Experimental Results

The proposed P-EFIM algorithm is compared with the performance of the PHUI-Growth algorithm, which is also a parallel HUIM algorithm using the MapReduce framework. The comparison experiments between the PHUI-Growth and P-EFIM ran in a Hadoop cluster environment of five desktop computers. The configuration of each computer is as follows: 2.8 GHz Intel i5-8400 processor, 1 TB HDD, 8 GB RAM, ubuntu 16.04 OS, java 1.9.0_191, and Hadoop 2.7.7.

4.1 Experimental datasets

All datasets for the experiments come from SPFM†. The eight datasets used for the experiments are Chainstore, Kosarak, Pumsb, Accidents, kddcup99, PowerC, RecordLink, and PAMP. Their characteristics are listed in Table 5, including each dataset's average transaction length, item count, and transaction count.

Three real-life transaction datasets, which are Pumsb, Accidents, and Kosarak datasets, are with synthetic utility values in SPMF. For Chainstore dataset, it is with real utilities from a real-life customer transaction dataset of a major grocery store chain. The utility of the remaining four datasets is generated by the generation tool in SPMF.

†http://www.philippe-fournier-viger.com/spmf/

Table 5 Characteristic of the datasets.

Dataset	Transaction count	Item count	Average transaction length
Chainstore	1 112 949	46 086	7.23
Kosarak	990 002	41 270	8.10
Pumsb	49 046	2113	74.00
Accidents	340 183	468	533.80
kddcup99	1 000 000	135	16.00
PowerC	1 040 000	140	7.00
RecordLink	574 913	29	10.00
PAMP	1 000 000	141	23.93

4.2 Comparison on the runtime with different thresholds

The runtime value of an algorithm is a critical criterion for its performance assessment. The thresholds at which both algorithms differ in performance were selected for the different datasets. The runtime values of the two algorithms at different thresholds with Reduce = 12 and 8, respectively, are shown in Figs. 4 and 5. From Fig. 4, the P-EFIM algorithm is at least one order of magnitude faster than the PHUI-Growth algorithm on the Chainstore, Kosarak, and Pumsb datasets and at least five times faster than that on the Accidents, RecordLink, and PAMP datasets. From Fig. 5, the P-EFIM algorithm is one order of magnitude faster than the PHUI-Growth algorithm on the Pumsb dataset and at least three times faster than that on the Accidents, Chainstore, Kosarak, RecordLink, and PAMP datasets. The higher runtime performance of the P-EFIM algorithm is mainly attributed to its efficient parallel structure and higher-performance algorithms. Furthermore, the iterative parallel structure of the hierarchical search approach was replaced by the efficient single-layer parallel structure of the depth-first search approach. The approach can reduce the number of parallel MapReduce tasks and HDD data access operations, thereby improving the performance of the P-EFIM algorithm. The pruning strategies and the transaction merging strategy of the EFIM algorithm significantly improved the performance of each node by reducing the runtime, thereby improving the performance of the distributed parallel algorithms.

4.3 Performance comparison on two load-balancing strategies

The performance of P-EFIM algorithm (Reduce = 8) with two load-balancing strategies is evaluated. P-EFIM with the S-style distribution strategy is denoted as P-EFIM(S), and P-EFIM algorithm with hash-based distribution strategy is denoted as P-EFIM(hash). Standard deviations, which are listed in Table 6, were

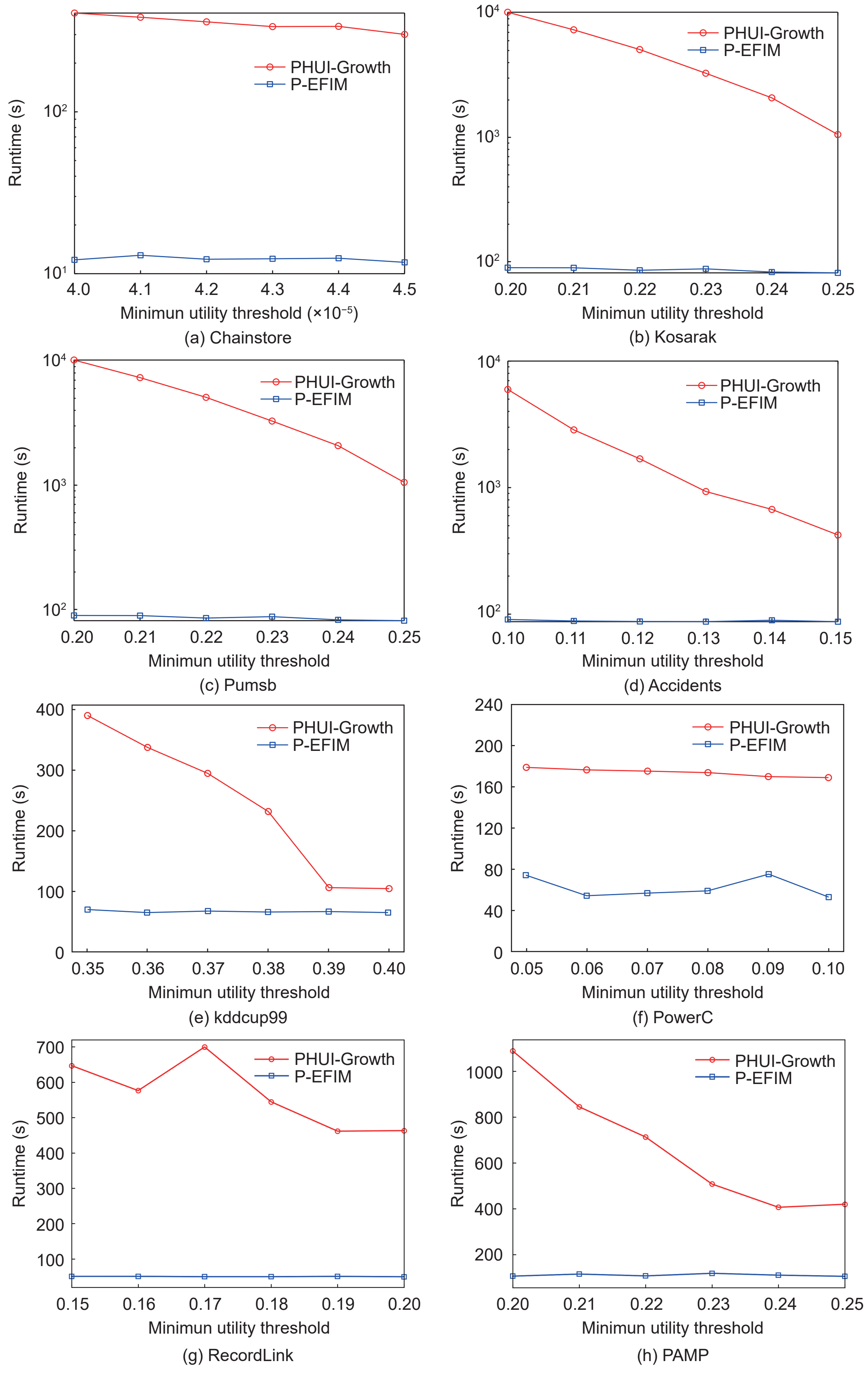

Fig. 4 Runtime values of the two parallel algorithms with Reduce = 12.

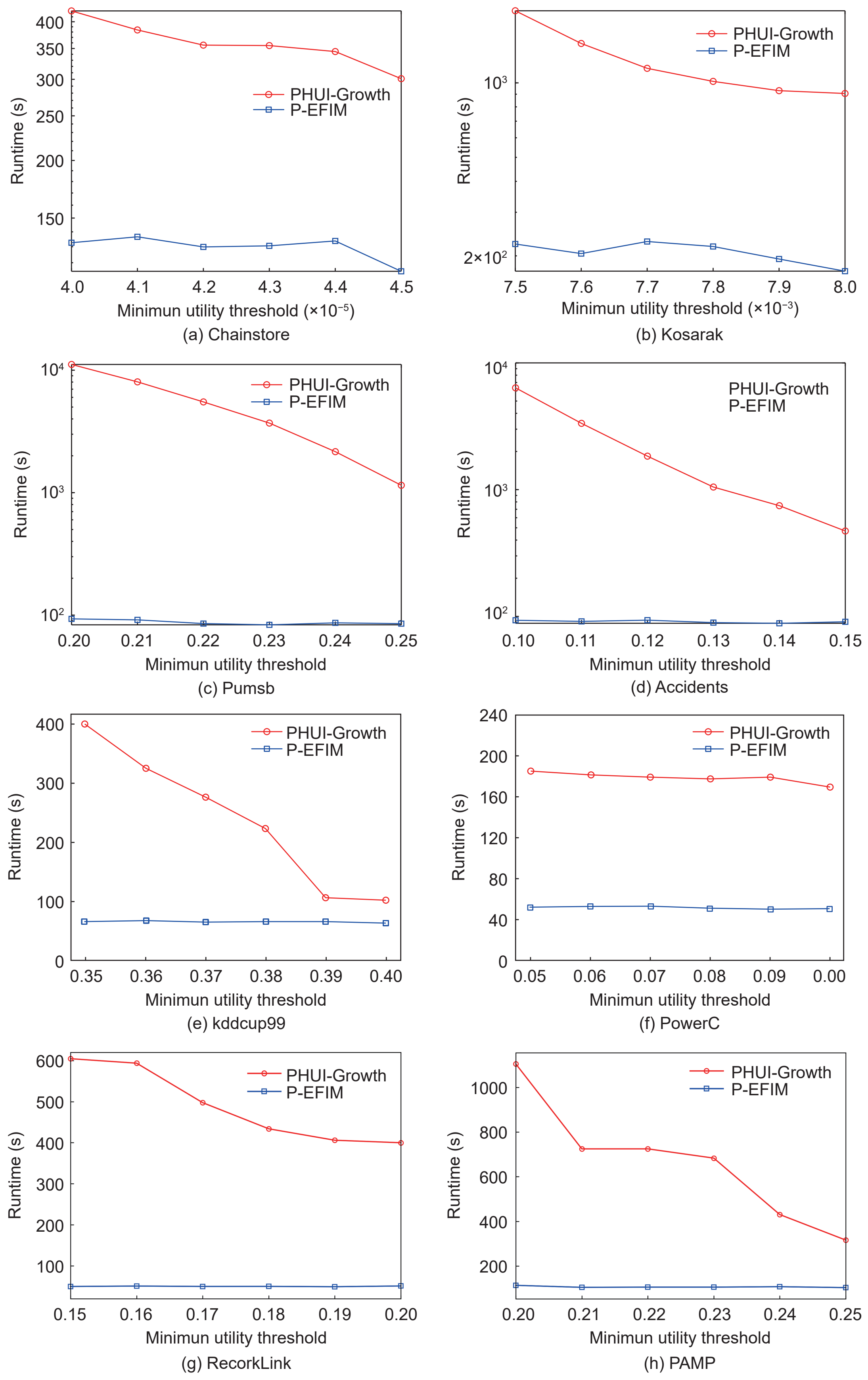

Fig. 5 Runtime values of the two parallel algorithms with Reduce = 8.

Table 6 Standard deviations of the two distribution strategies.

Dataset	minutil	P-EFIM(hash)	P-EFIM(S)
Chainstore	0.000 40	**163.31**	245.86
	0.000 41	300.94	**193.74**
	0.000 42	255.64	**82.53**
Kosarak	0.005 00	639.94	**621.76**
	0.006 00	243.05	**189.87**
	0.007 00	372.10	**318.64**
Pumsb	0.150 00	9365.56	**6973.80**
	0.160 00	3225.53	**2465.36**
	0.170 00	1291.40	**1175.39**
Accidents	0.060 00	449.75	**325.39**
	0.070 00	371.49	**152.53**
	0.080 00	**253.73**	379.48
kddcup99	0.200 00	1377.73	**620.08**
	0.210 00	1563.50	**967.20**
	0.220 00	1497.76	**1135.34**
PowerC	0.050 00	708.71	**577.24**
	0.060 00	758.60	**604.08**
	0.070 00	691.12	**569.88**
RecordLink	0.170 00	496.00	**446.60**
	0.180 00	554.10	**474.11**
	0.190 00	**472.61**	505.01
PAMP	0.020 00	4947.75	**2715.20**
	0.021 00	**7594.63**	7618.90
	0.022 00	7453.76	**2862.00**

calculated to compare the performance of two load-balancing strategies. From Table 6, the S-style distribution strategy is more stable than the hash-based distribution strategy in 20 out of 24 cases. On Pumsb, Kosarak, and kddcup99 datasets, P-EFIM(S) has achieved an overall better performance than P-EFIM(hash). On Chainstore, Accidents, RecordLink, and PAMP datasets, the S-style distribution strategy is slightly worse than the hash-based distribution strategy since the computing loads of the items were not strictly in descending order. The load-balancing performance of the S-style distribution strategy is higher because the computing loads of the nodes are roughly in descending order and can be effectively balanced using the S-style distribution strategy.

4.4 Comparison on the runtime between EFIM and P-EFIM

We also performed comparison experiments on the runtime between EFIM and P-EFIM to show the effectiveness of the proposed parallel strategy. Figure 6

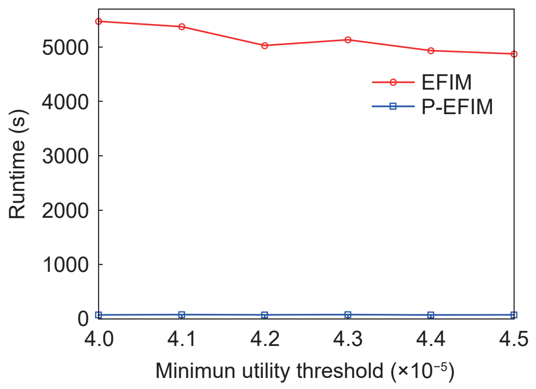

Fig. 6 Comparison on runtime between EFIM and P-EFIM on Chainstore dataset.

shows the runtime of EFIM and P-EFIM (Reduce = 12) on Chainstore dataset. The thresholds used in the experiment are very small, which brings difficulty to mine the HUIs for EFIM. From Fig. 6, it is clear that P-EFIM runs more quickly than EFIM. On the other datasets, since the thresholds are large, EFIM can mine all the HUIs quickly, and the parallel performance of P-EFIM on Hadoop can not show. The comparison results on runtime on the other datasets are not given here.

5 Conclusion

This paper proposes a distributed parallel EFIM (P-EFIM) algorithm based on Hadoop. The P-EFIM algorithm uses TWU values to prune and order datasets, thereby improving their efficiency. The proposed S-style distribution strategy can effectively balance the computing load among the nodes. Additionally, the previous mining algorithm using the hierarchical search was replaced with the EFIM algorithm based on the depth-first search to eliminate the iteration process, thereby significantly improving the efficiency of the P-EFIM algorithm. Experiments with several datasets showed that the runtime performance of the P-EFIM algorithm was much higher than that of the distributed parallel PHUI-Growth algorithm using the Hadoop framework.

References

[1] X. Li, S. Cao, L. Gao, and L. Wen, A threshold-control generative adversarial network method for intelligent fault diagnosis, *Complex System Modeling and Simulation*, vol. 1, no. 1, pp. 55–64, 2021.

[2] Q. Yu, L. Li, H. Zhao, Y. Liu, and K.-Y. Lin, Evaluation system and correlation analysis for determining the performance of a semiconductor manufacturing system, *Complex System Modeling and Simulation*, vol. 1, no. 3,

pp. 218–231, 2021.

[3] J. Li, X. Gu, Y. Zhang, and X. Zhou, Distributed flexible job-shop scheduling problem based on hybrid chemical reaction optimization algorithm, *Complex System Modeling and Simulation*, vol. 2, no. 2, pp. 156–173, 2022.

[4] L. Wang, Z. Pan, and J. Wang, A review of reinforcement learning based intelligent optimization for manufacturing scheduling, *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.

[5] Z. -H. Deng and S. -L. Lv, PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning, *Expert Systems with Applications*, vol. 42, no. 13, pp. 5424–5432, 2015.

[6] T. L. Dam, K. Li, P. Fournier-Viger, and Q. H. Duong, An efficient algorithm for mining top-rank-k frequent patterns, *Applied Intelligence*, vol. 45, no. 1, pp. 96–111, 2016.

[7] 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, 2005.

[8] Z. H. Deng, DiffNodesets: An efficient structure for fast mining frequent itemsets, *Applied Soft Computing*, vol. 41, pp. 214–223, 2016.

[9] T. Ryu, H. Kim, C. Lee, H. Kim, B. Vo, J. C. -W. Lin, W. Pedrycz, and U. Yun, Scalable and efficient approach for high temporal fuzzy utility pattern mining, *IEEE Transactions on Cybernetics*, doi: 10.1109/TCYB.2022.3198661.

[10] 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, 2014.

[11] C. -W. Wu, P. Fournier-Viger, J. -Y. Gu, and V. S. Tseng, Mining compact high utility itemsets without candidate generation, in *High-Utility Pattern Mining*, P. Fournier-Viger, J. C. -W. Lin, R. Nkambou, B. Vo, and V. S. Tseng, eds. Cham, Switzerland: Springer, 2019, pp. 279–302.

[12] Y. C. Lin, C. -W. Wu, and V. S. Tseng, Mining high utility itemsets in big data, in *Proc. 19th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Ho Chi Minh City, Vietnam, 2015, pp. 649–661.

[13] W. Song, Y. Liu, and J. Li, Mining high utility itemsets by dynamically pruning the tree structure, *Applied Intelligence*, vol. 40, pp. 29–43, 2014.

[14] S. Krishnamoorthy, Hminer: Efficiently mining high utility itemsets, *Expert Systems with Applications*, vol. 90, pp. 168–183, 2017.

[15] 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 *Proc. 14th Mexican International Conference on Artificial Intelligence*, Cuernavaca, Mexico, 2015, pp. 530–546.

[16] 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. 1, pp. 357–381, 2019.

[17] J. C. -W. Lin, Y. Djenouri, G. Srivastava, and J. M. -T. Wu, Large-scale closed high-utility itemset mining, in *Proc. 2021 International Conference on Data Mining Workshops* (*ICDMW*), Auckland, New Zealand, 2021, pp. 591–598.

[18] W. Gan, Z. Du, W. Ding, C. Zhang, and H. -C. Chao, Explainable fuzzy utility mining on sequences, *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 12, pp. 3620–3634, 2021.

[19] W. Gan, J. C. -W. Lin, P. Fournier-Viger, H. -C. Chao, V. S. Tseng, and P. S. Yu, A survey of utility-oriented pattern mining, *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1306–1327, 2021.

[20] J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, Targeted high-utility itemset querying, *IEEE Transactions on Artificial Intelligence*, doi: 10.1109/TAI.2022.3171530.

[21] H. Yao and H. J. Hamilton, Mining itemset utilities from transaction databases, *Data and Knowledge Engineering*, vol. 59, no. 3, pp. 603–626, 2006.

[22] Y. Liu, W. Liao, and A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, in *Proc. 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, Hanoi, Vietnam, 2005, pp. 689–695.

[23] 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, 2008.

[24] 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, 2009.

[25] 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, 2014.

[26] V. S. Tseng, C. W. Wu, B. E. Shie, and P. S. Yu, Up-growth: An efficient algorithm for high utility itemset mining, in *Proc. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, 2010, pp. 253–262.

[27] 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, 2013.

[28] M. Liu and J. Qu, Mining high utility itemsets without candidate generation, in *Proc. 21st ACM International Conference on Information and Knowledge Management*, Maui, HI, USA, 2012, pp. 55–64.

[29] 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 *Proc. 21st International Symposium on Methodologies for Intelligent Systems*, Roskilde, Denmark, 2014, pp. 83–92.

[30] S. Krishnamoorthy, Pruning strategies for mining high utility itemsets, *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[31] J. Sahoo, A. K. Das, and A. Goswami, An efficient fast algorithm for discovering closed$^+$ high utility itemsets, *Applied Intelligence*, vol. 45, no. 1, pp. 44–74, 2016.

[32] J. Liu, K. Wang, and B. C. M. Fung, Direct discovery of high utility itemsets without candidate generation, in *Proc. 2012 IEEE 12th International Conference on Data Mining*, Brussels, Belgium, 2012, pp. 984–989.

[33] C. W. Wu, P. Fournier-Viger, J. Y. Gu, and V. S. Tseng, Mining closed+ high utility itemsets without candidate generation, in *Proc. 2015 Conference on Technologies and Applications of Artificial Intelligence*, Tainan, China, 2015, pp. 187–194.

[34] 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, 2014.

[35] S. Dawar, V. Goyal, and D. Bera, A hybrid framework for mining high-utility itemsets in a sparse transaction database, *Applied Intelligence*, vol. 47, no. 3, pp. 809–827, 2017.

Zaihe Cheng received the master degree from Jiangnan University in 2012. He is now a senior visiting scholar of Jiangnan University and an associate professor at the School of Internet of Things, Wuxi Institute of Technology. His research interest is data mining and big data analysis.

Wei Shen received the master degree from Jiangnan University in 2020. His research interest is evolutionary algorithm and data mining.

Wei Fang received the PhD degree from Jiangnan University in 2008. He is now a professor of Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Jiangnan University. His research interest is evolutionary algorithm and its applications.

Jerry Chun-Wei Lin is currently a full professor with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway. He has published more than 360+ research articles in refereed journals (*IEEE TKDF*, *IEEE TCYB*, *IEEE TII*, *IEEE TITS*, *IEEE TIAS*, *IEEE TETCI*, *IEEE SysJ*, *IEEE SensJ*, *IEEE IOTJ*, *ACM TKDD*, *ACM TDS*, *ACM TMIS*, *ACM TOIT*, and *ACM TIST*) and international conferences (IEEE ICDE, IEEE ICDM, PKDD, and PAKDD), 11 edited books, as well as 33 patents (held and filed, 3 US patents). His research interests include data mining, soft computing, artificial intelligence and machine learning, and privacy preserving and security technologies. He is the editor-in-chief of the *International Journal of Data Science and Pattern Recognition*, and the guest editor/associate editor for several IEEE/ACM journals such as *IEEE TFS*, *IEEE TII*, *ACM TMIS*, *ACM TOIT*, and *IEEE Access*. He has been recognized as the most cited Chinese researcher respectively in 2018 and 2019 by Scopus/Elsevier. He is the fellow of IET (FIET) and senior member of both IEEE and ACM.