Parallel Mining for High Utility Itemsets Mining by Efficient Data Structure

Nguyen Manh Hung[1] and Dau Hai Phong[2]
[1] Military Technical Academy, Hanoi, Vietnam
[2] Thang Long University, Hanoi, Vietnam
E-mail: manhhungk12@mta.edu.vn, phong4u@gmail.com
Correspondence: Dau Hai Phong
Communication: received 5 July 2017, revised 8 August 2017, accepted 16 August 2017

Abstract: Mining high utility itemsets in transaction database is an important task in data mining and widely applied in many areas. Recently, many algorithms have been proposed, but most algorithms for identifying high utility itemsets need to generate candidate sets by overestimating their utility and then calculating their exact utility value. Therefore, the number of candidate itemsets is much larger than the actual number of high utility itemsets. In this paper, we introduce the Retail Transaction-Weighted Utility (RTWU) structure and propose two algorithms: EAHUI-Miner algorithm and PEAHUI-Miner parallel algorithm. They have been experimented and compared to the two most efficient algorithms: EFIM and FHM. Results show that our algorithm is better with sparse datasets.

Keywords: *High utility mining, EAHUI-miner, PEAHUI-miner, utility-list, RTWU.*

I. Introduction

Today, searching for hidden knowledge in huge volume datasets grows rapidly and becomes a very interesting problem. High utility mining, which aims to find a benefit value of an itemset greater than a given threshold, is a difficult problem. Unlike frequent itemset mining, in the mining of high utility itemsets, it is allowed to evaluate the importance of each item in the dataset.

One of the challenges in mining high utility itemsets is that they do not have the closure property [1], which ensures that if X is not a high utility itemset then all itemsets containing X are also not high utility itemsets. This leads to increasing the number of candidates and search space.

Some high utility mining algorithms proposed a two-phase mining approach, such as Udepth [2], PB [3], HP [4], UP-Growth [5] and UP-Hist [6]. However, algorithms using this approach generate a large number of candidates and require multiple scans of the database. Recently, to avoid the generation of a large number of candidates, algorithms such as HUI-Miner [7], EFIM [8] and FHM [9] attempted to directly search for high utility itemsets in only one phase.

In 2012, Liu *et al.* proposed the HUI-Miner algorithm [7] that uses a utility-list structure to store utility information of each itemset and information for reducing search space. Different from previous algorithms, HUI-Miner does not generate candidate high utility itemsets. After constructing the initial utility-lists from a mined database, HUI-Miner can mine high utility itemsets from these utility-lists.

In 2015, Zida and Fournier-Viger proposed the EFIM algorithm [8] that relies on two upper-bounds named sub-tree utility and local utility to more effectively prune the search space. It also introduced a novel array-based utility counting technique named Fast Utility Counting to calculate these upper-bounds in linear time and space. Moreover, to reduce the cost of database scans, EFIM proposes efficient database projection and transaction merging techniques. An extensive experimental study on various datasets showed that EFIM is in general two to three orders of magnitude and memory requirement is up to eight times less than the state-of-art algorithms, such as d2HUP, HUI-Miner, HUP-Miner, FHM, and UP-Growth+.

The FHM algorithm [9] restricted the high join cost of the HUI-Miner algorithm [7] based on the closure property of the Transaction-Weighted Utility (TWU). It does not join itemsets containing the pair (x, y) having $TWU(x, y)$ smaller than a given minimum utility threshold. However, as reviewed in [4], the complexity of TWU is more than necessary.

In this paper, we propose the Retail Transaction-Weighted Utility (RTWU) structure, the EAHUI-Miner sequential algorithm, and the PEAHUI-Miner parallel algorithm for mining high utility itemsets by efficiently pruning candidate itemsets using the RTWU structure.

The paper is organized as follows. First, Section II reviews related works. Then, Sections III and IV respec-

TABLE I
DATABASE TRANSACTION

tid	Transactions
1	b:1, c:2, d:1, g:1
2	a:4, b:1, c:3, d:1, e:1
3	a:4, c:2, d:1
4	c:2, e:1, f:1
5	a:5, b:2, d:1, e:2
6	a:3, b:4, c:1, f:2
7	d:1, g:5

TABLE II
UTILITY TABLE

Item	a	b	c	d	e	f	g
Utility	1	2	1	5	4	3	1

tively present the EAHUI-Miner and PEAHUI-Miner algorithms. Section V gives experimental evaluation. Finally, Section VI concludes the paper.

II. RELATED WORKS

Let $\mathcal{I} = \{i_1, i_2, i_3, \ldots, i_n\}$ be a set of items and D be a database composed of a utility table and a transaction table. Each item in $\mathcal{I}$ has a utility value in the utility table. Each transaction T_j in the transaction table T has a unique transaction identifier ($tid = j$), and is a subset of $\mathcal{I}$, in which each item is associated with a count value. An itemset is a subset of $\mathcal{I}$ and is called a k-itemset if it contains k items.

To better explain the concept, we give a database transaction represented as a table, as Table I, and utilities of the items, as shown in Table II.

Definition 1 ([7]): The **internal** utility of item i_k in transaction T_j, denoted as $O(i_k, T_j)$, is the count value associated with i in T_j in the transaction table of D.

For examples, in Table I, $O(a, T_2) = 4$ and $O(d, T_2) = 1$.

Definition 2 ([7]): The **external** utility of item i_k, denoted as $S(i_k)$, is the utility value of i_k in the utility table of D.

For examples, in Table II, $S(a) = 1$ and $S(d) = 5$.

Definition 3 ([7]): The **utility of item** i_k in transaction T_j, denoted as $U(i_k, T_j)$, is the product of $O(i_k, T_j)$ and $S(i_k)$, that is,

$$U(i_k, T_j) = S(i_k) \times O(i_k, T_j).$$

For examples, $U(a, T_2) = 1 \times 4 = 4$ and $U(d, T_2) = 5 \times 1 = 5$.

Definition 4 ([7]): The **utility of itemset** X **in transaction** T_j, denoted as $U(X, T_j)$, is the sum of the utilities of all the items in X in T_j in which X is contained, that is,

$$U(X, T_j) = \sum_{(i_k \in X) \wedge (X \subseteq T_j)} U(i_k, T_j).$$

For example, $U(\{ad\}, T_2) = 1 \times 4 + 5 \times 1 = 9$.

Definition 5 ([7]): The **utility of itemset** X, denoted as $U(X)$, is the sum of the utilities of X in all the transactions T_j containing X in D, that is,

$$U(X) = \sum_{(T_j \in D) \wedge (X \subseteq T_j)} U(X, T_j).$$

For example, in Table I, $U(\{ad\}) = U(\{ad\}, T_2) + U(\{ad\}, T_3) + U(\{ad\}, T_5) = (1 \times 4 + 5 \times 1) + (1 \times 4 + 5 \times 1) + (1 \times 5 + 5 \times 1) = 28$.

Definition 6 ([7]): The **utility of transaction** T_j, denoted as $TU(T_j)$, is the sum of the utilities of all the items in T_j, that is,

$$TU(T_j) = \sum_{i_k \in X} U(i_k, T_j),$$

and the total utility of D is the sum of the utilities of all the transactions in D.

For example, $TU(T_3) = 1 \times 4 + 1 \times 2 + 5 \times 1 = 11$ and $TU(T_4) = 1 \times 2 + 4 \times 1 + 3 \times 1 = 9$.

Definition 7 ([7]): The **transaction-weighted utility of itemset** X in D, denoted as $TWU(X)$, is the sum of the utilities of all the transactions containing X in D, that is,

$$TWU(X) = \sum_{(X \subseteq T_j) \wedge (T_j \in D)} TU(T_j).$$

For example, $TWU(\{ad\}) = TU(T_2) + TU(T_5) = 18 + 26 = 44$.

Definition 8 (High transaction-weighted utility itemset (HTWU) [10]): For a given itemset X, X is called a **high transaction-weighted utility itemset** if $TWU(X) \geq \lambda_{\min}$, where $\lambda_{\min}$ is a user-specified threshold.

For example, if $\lambda_{\min} = 12$ and $TWU(\{ad\})$ then $\{ad\}$ is $HTWU$.

Definition 9 ([7]): Given an itemset X and a transaction (or itemset) T with $X \subseteq T$, the **set of all the items after** X **in** T is denoted as T/X.

For example, in Table I, $T_2/\{abc\} = \{de\}$ and $T_2/\{cd\} = \{e\}$.

Definition 10 ([7]): The **remaining utility of itemset** X in transaction T, denoted as $ru(X, T)$, is the sum of the utilities of all the items in T/X in T, that is,

$$ru(X, T) = \sum_{i \in (T/X)} U(i, T).$$

For example, $ru(\{abc\}, T_2) = 5 \times 1 + 4 \times 1 = 9$.

Definition 11 ([7]): Each element in the utility-list of an itemset X contains three fields: tid, $iutil$, and $rutil$, where tid is the identifier of the transaction, T_{tid}, containing X, $iutil$ is the utility of X in T_{tid}, *i.e.*, $U(X, T_{tid})$, and $rutil$ is the remaining utility of X in T_{tid}, *i.e.*, $ru(X, T_{tid})$.

Lemma 1 ([7]): Given the utility-list of an itemset X, if the sum of all the iutils and rutils in the utility-list is less

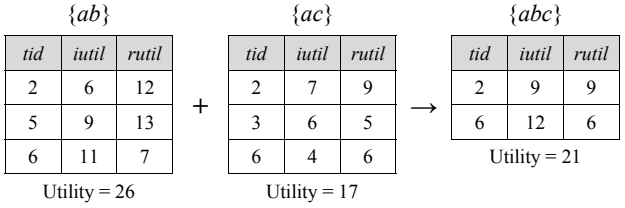

Figure 1. Joint utility-list of itemsets $\{ab\}$ and $\{ac\}$.

than a given threshold $\lambda_{\min}$, any extension X' of X is not high-utility.

For example, the joint utility-list of itemsets $\{ab\}$ and $\{ac\}$ from Table I and Table II is shown in Figure 1.

Let us join utility-list of two itemsets based on the transaction identifier tid. For example, $iutil(\{abc\}, T_2) = iutil(\{ab\}, T_2) + iutil(\{ac\}, T_2) - iutil(\{a\}, T_2) = 6+7-4 = 9$. Similarly, $iutil(\{abc\}, T_6) = iutil(\{ab\}, T_6) + iutil(\{ac\}, T_6) - iutil(\{a\}, T_6) = 11 + 4 - 3 = 12$. Then, calculate $rutil(\{abc\}, T_2) = rutil(\{c\}, T_2) = 9$ and $rutil(\{abc\}, T_6) = rutil(\{c\}, T_6) = 6$.

Assuming that $\lambda_{\min} = 30$, we have the total utility of the itemset $\{abc\}$ is 21, so the itemset $\{abc\}$ is not a high utility one. But the sum of the *iutil* and *rutil* of the itemset $\{abc\}$ is $21 + 15 = 36$. Thus, the itemset $\{abc\}$ should still be able to generate high utility itemsets.

The FHM algorithm is based on the observation that although HUI-Miner performs a single phase and thus does not generate candidates as does the two-phase model, it explores the search space of itemsets by generating itemsets and a costly join operation has to be performed to evaluate the utility of each itemset. To reduce the number of joins that are performed, FHM proposes a pruning strategy named Estimated Utility Co-occurrence Pruning (EUCP) that can prune itemsets without having to perform joins. Specifically, FHM stores the TWU value of all pairs (a, b) and is based on the closure property of TWU, so all itemsets containing pairs (a, b) and having a TWU value smaller than $\lambda_{\min}$ will not join the utility-list of itemsets.

Furthermore, the FHM algorithm mines high itemsets in depth. Assume that items are arranged in the alphabet order, $\{aX\}$ is an itemset with the prefix a, $\{bX\}$ is an itemset with the prefix b. Thus, $\{bX\}$ does not contain item a, but since computation of $TWU(\{bX\})$ may still contain the utility of item a, the value of $TWU(\{bX\})$ is the upper bound of $U(\{bX\})$ and larger than necessary, so using $TWU(\{bX\})$ to prune candidate itemsets is not effective.

III. EAHUI-Miner Algorithm

In this section we present the RTWU structure and EAHUI-Miner sequential algorithm for mining high utility itemsets using the RTWU structure to effectively pruning candidate itemsets.

TABLE III
Utility-list of $\{cd\}$

tid	$iutil$	$itemutil$	$rutil$
1	7	5	1
2	8	5	4
3	7	5	0

Definition 12: Extended utility-list of an itemset P is a list of items in which each item consists of 4 fields: tid, $iutil$, $itemutil$ and $rutil$, where tid is the identity of transaction containing P, *i.e.*, T_{tid}, $iutil$ is the utility of itemset P in transaction T_{tid}, *i.e.*, $U(P, T_{tid})$, $itemutil$ is utility of item x in T_{tid}, *i.e.*, $U(x, T_{tid})$, and $rutil$ is remaining utility of transaction T_{tid} except the utility of P, starting from the item after item x.

For example, utility-list of $\{cd\}$ is shown in Table III.

Definition 13: The remaining transaction-weighted utility of an item (x, y) (x is before y in order) in a transaction T is the sum of the remaining utilities of all the items in T from item x, denoted as $RTWU(x, y, T)$, that is,

$$RTWU(x, y, T) = \sum_{i \in [T/x']} rutil(i, T),$$

where $[T/x']$ is the transaction T that does not contain items before item x in T.

For example, $RTWU(c, d, T_2) = 4$.

Definition 14: The remaining transaction-weighted utility of an item pair (x, y) in database D is the sum of the remaining transaction-weighted utilities of pair (x, y) in all transactions containing items x and y in D, denoted as $RTWU(x, y)$, that is,

$$RTWU(x, y) = \sum_{(T \in D) \wedge ((x,y) \subseteq T)} RTWU(x, y, T).$$

For example, $RTWU(c, d) = 1 + 4 + 0 = 5$.

Definition 15: The RTWU structure is determined by a triplet $(x; y; c) \in \mathcal{I} \times \mathcal{I} \times R$, where $\mathcal{I}$ is the set of items in the database, x and y are two items of $\mathcal{I}$ (x is before y in order), and c is a real number such that $c = RTWU(x, y)$.

Definition 16: Sum of all the $iutils$ in the extended utility-list of itemset Px, denoted as $Px.sumiutils$, *i.e.*,

$$Px.sumiutils = \sum_{(Px \in T) \wedge (T \in D)} U(P, T).$$

Definition 17: Sum of all the $rutils$ in the extended utility-list of itemset Px, denoted as $Px.sumrutils$, *i.e.*,

$$Px.sumiutils = \sum_{(Px \in T) \wedge (T \in D)} U(T \setminus Px, T).$$

$\{ab\}$

tid	*iutil*	*itemutil*	*rutil*
2	4	2	12
5	5	4	13
6	3	8	7

$\{ac\}$

tid	*iutil*	*itemutil*	*rutil*
2	4	3	9
3	4	2	5
6	3	1	6

Figure 2. Extended utility-lists of itemsets $\{ab\}$ and $\{ac\}$

Definition 18: Sum of all the *itemutils* in the extended utility-list of itemset Px, denoted as $Px.sumitemutils$, *i.e.*,

$$Px.sumitemutils = \sum_{(Px \in T) \wedge (T \in D)} U(x, T).$$

Lemma 2: Given two extended utility-lists of itemsets Px and Py of P. If $\min(Px.sumiutils, Py.sumiutils) + RTWU(x, y)$ is smaller than $\lambda_{\min}$ then Pxy and expand of Pxy are not high transaction weight utility itemsets.

Proof: For all pairs (x, y), we have

$$U(Pxy) = UTIL(P) + UTIL(xy),$$

where $U(Pxy)$ is utility of itemset Pxy in D, $UTIL(P)$ is utility of P in transactions containing Pxy, $UTIL(xy)$ is utility of itemset $\{xy\}$ in transactions containing Pxy.

According to Definition 12 of an extended utility-list and join operation of two extended utility-lists of Px and Py, implemented as join operation of HUI-Miner [7] and FHM [9] algorithms, we have

$$UTIL(P) \leq \min(Px.sumiutils, Py.sumiutils).$$

According to Definition 13 for $RTWU(x, y)$, we have

$$UTIL(xy) \leq RTWU(x, y).$$

Thus,

$$\begin{aligned} U(Pxy) &= UTIL(P) + UTIL(xy) \\ &\leq \min(Px.sumiutils, Py.sumiutils) + RTWU(x, y). \end{aligned}$$

□

For example, in Figure 2, extended utility-list of itemsets $\{ab\}$ and $\{ac\}$with $\lambda_{\min} = 30$.

According to the FHM algorithm, with the itemset $P = \{abc\}$ having the utility-list shown in Figure 1, we have $P.sumiutils + P.sumrutils = 21 + 15 = 36$ and $TWU(\{ab\}) = TU(T_2) + TU(T_5) + TU(T_6) = 18 + 22 + 18 = 58$. Similarly, $TWU(\{ac\}) = 47$ and $TWU(\{bc\}) = 46$. Thus, the itemset P contains pairs (a, b), (a, c) and (b, c), the TWU of each is greater than 30, and since $P.sumiutils + P.sumrutils > 30$, it continues to join the utility-list of the itemset P with utility-lists of other itemsets to find high utility itemsets.

Above, $P = \{abc\}$ has

$$P.sumiutils + P.sumrutils = 36 > 30.$$

Algorithm 1: Building extended utility-list

Inputs:
$Px.ULs$ ▷ extended utility-list of Px;
$Py.ULs$ ▷ extended utility-list of Py;
Output:
$Pxy.ULs$ ▷ extended utility-list of Pxy;
function CONSTRUCT(Px, Py)
  $Pxy.ULs = Null$;
  for each item $Ex \in Px.ULs$ **do**
    if ($\exists\ Ey \in Py.ULs$ and $Ex.T_{tid} = Ey.T_{tid}$) **then**
      $Exy = \langle Ex.T_{tid}, Ex.iutils + Ex.itemutils, Ey.itemutils, Ey.rutils \rangle$;
      Insert Exy into $Pxy.ULs$;
    end if
  end for
  Return $Pxy.ULs$;
end function

However, according to Lemma 2, we have

$$\begin{aligned} &\min(\{ab\}.sumiutils, \{ac\}.sumiutils) + RTWU(b, c) \\ &= \min\{(4 + 5 + 3), (4 + 4 + 3)\} + (9 + 6) \\ &= 11 + 15 = 26 < 30. \end{aligned}$$

Thus, it does not join the utility-list of itemset $\{abc\}$ with utility-lists of other itemsets.

Based on Lemma 2, we propose to improve the FHM algorithm using the RTWU structure, which is presented in the next section.

1. Building the Extended Utility-List

In the first database scan, the extended utility-list of 1-itemset of P is empty (*i.e.*, $P.iutil = 0$). Extended utility-lists containing k-itemsets ($k \geq 2$) are built using Algorithm 1.

2. Mining High Utility Itemsets

Algorithm 2 shows the pseudo-code of the EAHUI-Miner algorithm. For each utility-list X in ULs, if the sum of all the utilities of items in X exceeds $\lambda_{\min}$, then the extension associated with X is high utility and is returned by the algorithm. In EAHUI-Miner, we use the EUCS structure of the FHM algorithm [9] and the RTWU structure of Definition 14.

IV. PEAHUI-MINER PARALLEL ALGORITHM

In this section, we present the PEAHUI-Miner parallel algorithm derived from the EAHUI-Miner algorithm in Section III. This is a dynamic distribution parallel algorithm following a fine-grain and shared memory model to

Algorithm 2: High Utility Itemset Mining Algorithm

Inputs:
ULs ▷ high utility 1-itemset;
$\lambda_{\min}$ ▷ minimum utility threshold;
Output:
$HUIs$ ▷ high utility itemsets.
procedure EAHUI-MINER($ULs, \lambda_{\min}$)
  for each utility-list of $Px \in ULs$ **do**
    if ($Px.sumiutils + Px.sumitemutils \geq \lambda_{\min}$) **then**
      return Px;
    end if
    if ($Px.sumiutils + Px.sumitemutils + Px.sumrutils \geq \lambda_{\min}$) **then**
      $exULs = Null$;
      for each Py after $Px \in ULs$ **do**
        if (EUCS$(x, y) \geq \lambda_{\min}$) &
          $\min(Px.sumiutils, Py.sumiutils) + RTWU(x, y) \geq \lambda_{\min}$) **then**
          $exULs = exULs +$ Construct(Px, Py);
        end if
        Call EAHUI-Miner($exULs, \lambda_{\min}$);
      end for
    end if
  end for
end procedure

improve load balancing between processes. The flowchart of the algorithm is shown in Figure 3.

The main parallel step is implemented by the parallel loop structure (#pragma omp parallel for) of the OpenMP library for utility-list of 1-itemsets. The fine-grain parallel is a method which divides each element in the ULs of 1-itemsets for each process.

The number of processes is usually much smaller than the number of elements in the ULs. The parallel loop structure will divide the next element in ULs for each processor when it has finished processing the previous element.

V. EXPERIMENTS

1. Environment and Data

We performed experiments to evaluate the performance of the proposed EAHUI-Miner and PEAHUI-Miner algorithms. Experiments were carried out on a computer with a third generation 64-bit core i7 processor running Windows 7 and 8 GB of RAM. We compared the performance of EAHUI-Miner with the state-of-the-art algorithms, EFIM and FHM, which are downloaded from the homepage of Fournier-Viger [11]. Our two algorithms are coded by Java, and the OpenMP library is used for the parallel algorithm. Experiments were performed using a set of standard

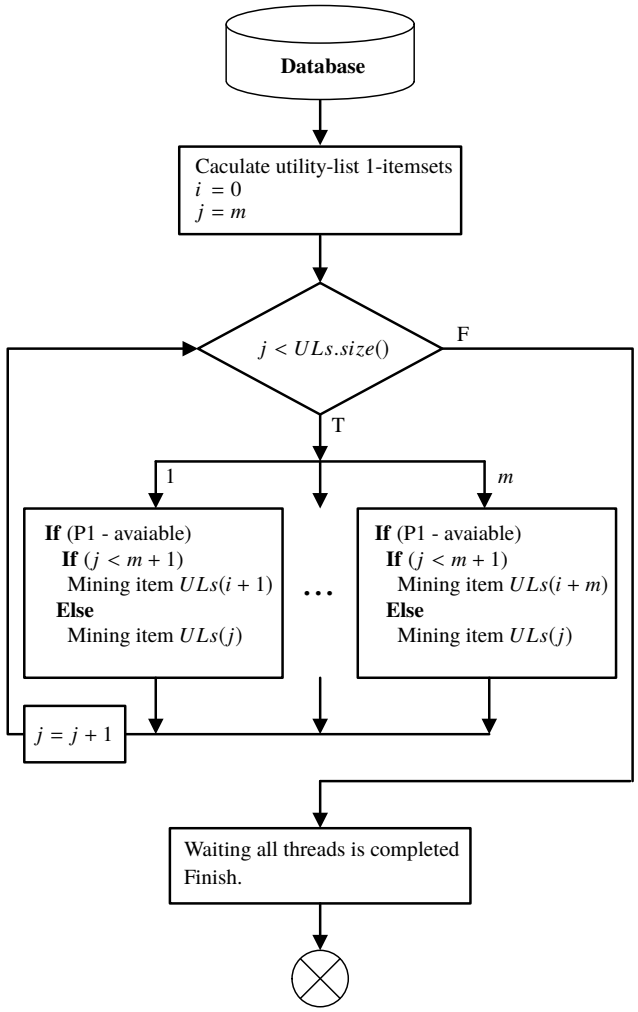

Figure 3. Flowchart of the PEAHUI-Miner algorithm.

TABLE IV
DATASET CHARACTERISTICS

Database	AvgLen	#Trans	#Items
T10I4D100K	10	100,000	1,000
T10I4D200K	10	200,000	1,000
Foodmart	4.4	4,141	1,559
Mushroom	23	8,124	119

TABLE V
NUMBER OF CANDIDATE ON DIFFERENT DATASETS

	$\lambda_{\min}$	FHM	EAHUI-Miner
T10I4D100K	2,500	153,016	125,647
T10I4D200K	2,500	925,994	891,585
Foodmart	1,000	259,876	258,921
Mushroom	100,000	1,588,018	1,587,927

datasets namely T10I4D100K, T10I4D200K, Foodmart and Mushroom. Table IV summarizes their characteristics.

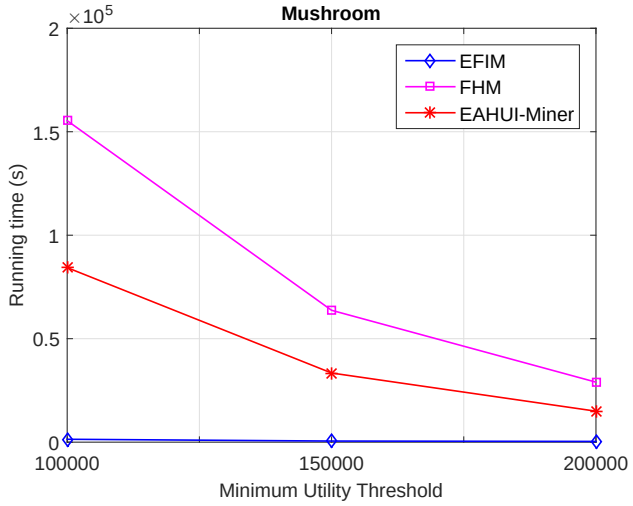

Figure 4. Execution times on Mushroom dataset.

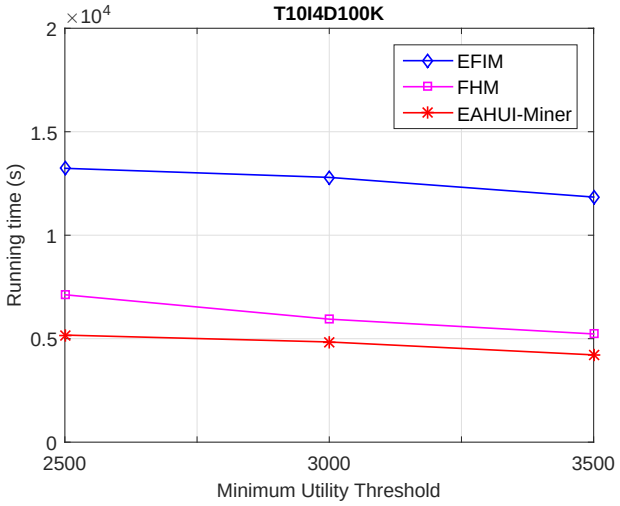

Figure 6. Execution times on T10I4D100K dataset.

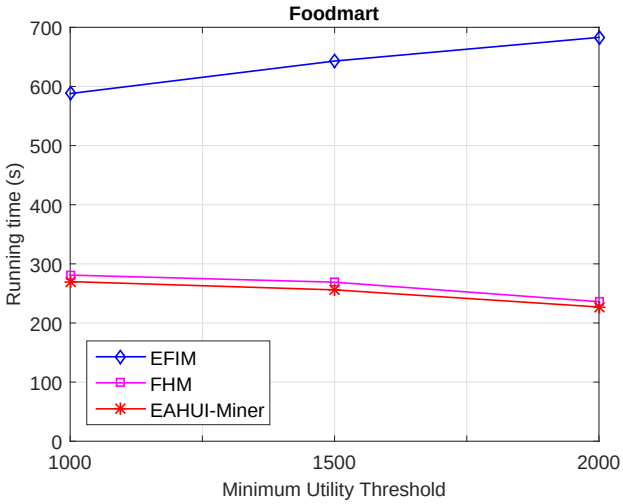

Figure 5. Execution times on Foodmart dataset.

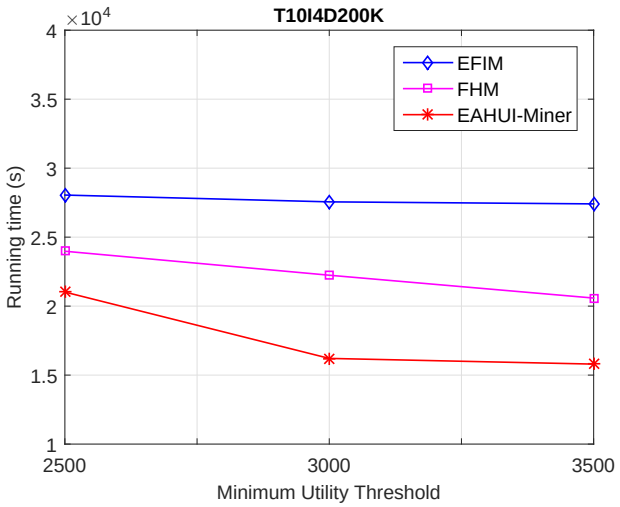

Figure 7. Execution times on T10I4D200K dataset.

2. Candidates and High Utility Itemsets

The experimental results of the EAHUI-Miner, PEAHUI-Miner, EFIM, and FHM algorithms are obtained with many minimum thresholds of utility and the same number of high utility itemsets.

Table V shows the number of candidates that the two algorithms generated. The results show that number of candidates generated by FHM is larger than that generated by EAHUI-Miner.

3. Running Time

Running times of the EFIM, FHM, and EAHUI-Miner algorithms are shown in Figures 4, 5, 6 and 7. The results show that EFIM is very fast on dense datasets with small size for the set $\mathcal{I}$, EAHUI-Miner and FHM are faster than EFIM on sparse large datasets with large size for $\mathcal{I}$ but there are fewer items in each transaction.

Figure 8 and 9 compare the running times of the EAHUI-Miner sequential algorithm and the PEAHUI-Miner parallel algorithm on the T10I4D100K and T10I4D200K datasets.

VI. Conclusions

In this paper, we have introduced the RTWU structure and two algorithms – EAHUI-Miner sequential algorithm and PEAHUI-Miner parallel algorithm. Experimental results have showed that the number of candidate sets that EAHUI-Miner generates is less than that of FHM. Moreover, the two proposed algorithms are also faster than two state-of-the-art algorithms (FHM and EFIM), with datasets containing thousands of items and huge numbers

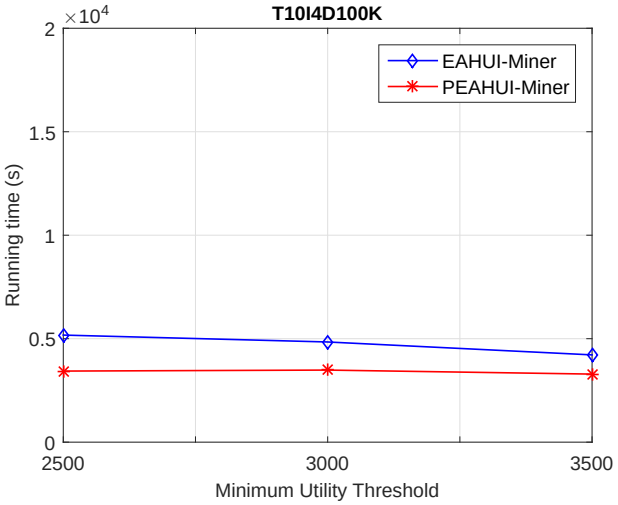

Figure 8. Execution times on T10I4D100K dataset.

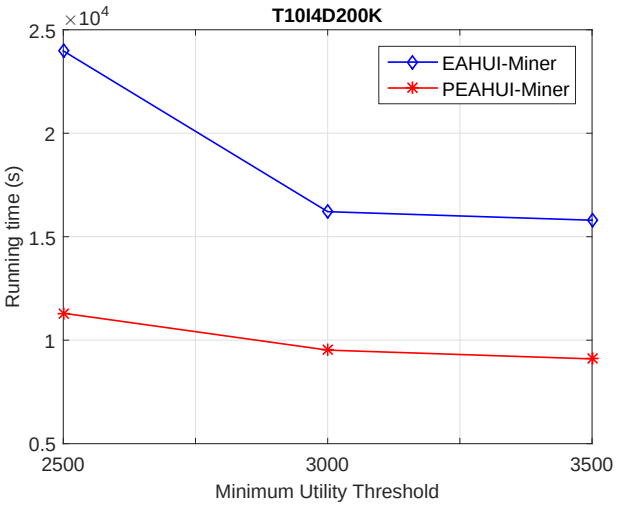

Figure 9. Execution times on T10I4D200K dataset.

of transactions (large databases), but there are fewer items in each transaction. In the future, we plan to test and implement the algorithms using Hadoop framework.

REFERENCES

[1] R. Agrawal, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 478–499.

[2] W. Song, Y. Liu, and J. Li, "Vertical mining for high utility itemsets," in *IEEE International Conference on Granular Computing (GrC)*. IEEE, 2012, pp. 429–434.

[3] 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.

[4] D. H. Phong and N. M. Hung, "Một mô hình hiệu quả khai phá tập mục lợi ích cao," *Research and Development on Information and Communication*, pp. 26–36, Jun. 2015.

[5] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "Up-growth: an efficient algorithm for high utility itemset mining," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 253–262.

[6] S. Dawar and V. Goyal, "Up-hist tree: An efficient data structure for mining high utility patterns from transaction databases," in *Proceedings of the 19th International Database Engineering & Applications Symposium*. ACM, 2015, pp. 56–61.

[7] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 55–64.

[8] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, "Efim: a highly efficient algorithm for high-utility itemset mining," in *Mexican International Conference on Artificial Intelligence*. Springer, 2015, pp. 530–546.

[9] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, "Fhm: faster high-utility itemset mining using estimated utility co-occurrence pruning," in *International symposium on methodologies for intelligent systems*. Springer, 2014, pp. 83–92.

[10] Y. Liu, W. keng Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2005, pp. 689–695.

[11] Philippe Fournier-Viger's site. http://www.philippe-fournier-viger.com.

Nguyen Manh Hung received the Ph.D. degree in Computer Science, Scientific Academy, Russian Federation in 2004. He is working in the Postgraduate Department - Military Technical Academy, Hanoi, Vietnam. His current research interests include: modern data structures, data mining, classification of the high-performance packet.

Dau Hai Phong received the Master degree in Information Technology, Military Technical Academy, Hanoi, Vietnam in 2008. He is working in Faculty of Mathematics and Computer Science, Thang Long University. His current research interests include: Data mining, Parallel computing, Distributed Database.