Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.017

IJCSMC, Vol. 7, Issue. 2, February 2018, pg.23 – 32

AN EFFICIENT ALGORITHM FOR MINING CLOSED HIGH UTILITY ITEMSET

V.G.Vijilesh[1], Dr. S. Hari Ganesh[2]

[1]Senior Lecturer, International School of Business & Technology Kampala, Uganda
Email: vijilesh.v.g@gmail.com

[2]Asst. Professor, Dept. of Computer Science, H.H. The Rajah's College, Pudukottai, Tamilnadu, India
Email: hariganesh17@gmail.com

Abstract: Mining of High utility itemsets refers to discovering sets of data items that have high utilities. In recent years the high utility itemsets mining has extensive attentions due to the wide applications in various domains like biomedicine and commerce. Extraction of high utility itemsets from database is very problematic task. The formulated high utility itemset degrades the efficiency of the mining process. The existing algorithms suffer the problem of producing a large amount of candidates, which degrades the mining performance in terms of time and space. The Closed high utility itemset proposed to achieve this goal. In this paper, a novel algorithm has been proposed to discover the high utility itemset without candidates generation. Experimental results show that the proposed algorithm is faster than the state-of-the-art algorithms

Keywords—High utility itemset, closed high utility itemset, candidates generations.

I. INTRODUCTION

High Utility Itemset Mining is a popular data mining task for discovering useful patterns in customer transaction databases [8]. It consists of discovering itemsets that yield a high utility (e.g. high profit), that is High Utility Itemsets [12]. Besides customer transaction analysis, HUIM also has applications in other domains such as click stream analysis and biomedicine [10,11]. HUIM can be viewed as an extension of the problem of Frequent Itemset Mining (FIM), where a weight (e.g unit profit) may be assigned to each item, and where purchase quantities of items in transactions are not restricted to binary values. HUIM is commonly observed as a tough problem, because the utility measure used in HUIM is neither monotonic nor anti-monotonic, unlike the support measure in FIM [1]. That is, the utility of an itemset may be greater, smaller or equal to the utility of its subsets. For this reason, efficient search space pruning techniques developed in FIM cannot be used in HUIM [6].

Several algorithms have been proposed for HUIM. However, an important limitation of traditional HUIM algorithms is that they often produce a huge amount of high-utility itemsets. Hence, it can be very time consuming for users to analyze the output of these algorithms [2]. Moreover, this makes HUIM algorithms suffer from long execution times and even fail to run due to huge memory consumption or lack of storage space [7]. To address this issue, it was recently proposed to mine a concise and lossless representation of all HUIs named closed high-utility itemsets (CHUIs). The concept of CHUI extends the concept of closed patterns from FIM. A CHUI is a HUI having no proper supersets that are HUIs and appear in the same number of transactions .This latter representation is interesting since it is lossless (it allows deriving all HUIs). Furthermore, it is also meaningful for real applications since it only discovers the largest HUIs that are common to groups of customers [3,11]. However, CHUI mining can be very computationally expensive [9]. In this paper, we address the need for a more efficient CHUI mining algorithm by proposing an algorithm named EFIM-Closed (Efficient high-utility Itemset Mining - Closed), based on the strict constraint that for each itemset in the search space, all operations for that itemset should be performed in linear time and space [4] .

The proposed algorithm efficiently find out closed high utility item sets uses closure jumping, forward closure checking and backward closure checking strategies. To reduce the cost of database scans, the proposed algorithm relies on two efficient techniques named High-utility Database Projection and High utility Transaction Merging [9,10]. Also, the proposed algorithm comprises two new upper-bounds on the utility of itemsets named sub-tree utility and local utility to effectively trim the search space, and an efficient Fast Utility Counting technique to compute them. An experimental study show that the proposed algorithm is much faster and consumes less memory than the existing.

II. PROBLEM STATEMENT

This section introduces the problem of closed high-utility itemset mining.

Let I be a finite set of items. An itemset X is a finite set of items such that $X \subseteq I$. A transaction database is a multiset of transactions $D = \{T_1, T_2, ..., T_n\}$ such that for each transaction T_c, $T_c \subseteq I$ and T_c has a unique identifier c called its TID (Transaction ID). Each item 'i' is associated with a

positive number p(i), called its external utility . Each item i appearing in a transaction T_c has a positive number q(i, Tc), called its internal utility. The utility of an item i in a transaction T_c is denoted as u(i, T_c) and defined as p(i) × q(i, T_c).

The utility of an itemset X in a transaction T_c is denoted as u(X, T_c) and defined as u(X, T_c) = $\sum_{i \in X} u(i, Tc)$ if X ⊆ T_c. The utility of an itemset X in a database is denoted as u(X) and defined as u(X) = $\sum_{T_c \in g(x)} u(X, T_c)$, where g(X) is the set of transactions containing X. The support of an itemset X in a database D is denoted as sup(X) and defined as $|g(X)|$.

An itemset X is called as a high-utility itemset if its utility is not less than a user-specified minimum threshold. Otherwise, X is a low-utility itemset. A HUI X is a closed high-utility itemset (CHUI) [11] iff there exists no HUI Y such that X ⊂ Y and sup(X) = sup(Y). The problem of closed high-utility itemset mining is to find out all closed high-utility itemsets, given a threshold set by the user .

III. RELATED WORK

The frequent itemset mining search space prune techniques cannot be used in HUIM because the utility measure is neither monotonic nor anti-monotonic [5]. Many HUIM algorithms avoid this problem by overestimating the utility of itemsets using the concept of Transaction-Weighted Utilization (TWU) measure defined as follows.

Transaction weighted utilization

To calculate the transaction utility of a transaction Tc is to sum the utilities of items from Tc in that transaction i.e. $T \cup (T_c)$ =$\sum_{x \in T_c} u(x, T_c)$. The following property of the TWU is commonly used in HUIM to prune the search space

Property 1 (Pruning using the TWU)

Let be an itemset X. If TWU(X) < minutil, then X is a low-utility itemset as well as all its supersets. Several HUIM algorithms utilize Property 1 to prune the search space. They operate in two phases. In the first phase, they identify candidate high-utility itemsets by calculating their TWUs. In the second phase, they scan the database to calculate the exact utility of all candidates to filter low-utility itemsets. Recently, algorithms that mine high-utility itemsets using a single phase were proposed to avoid the problem of candidate generation, and were shown to outperform previous algorithms. One-phase algorithms rely mainly on the concept of remaining utility to prune the search space.

Property 2 (Pruning using remaining utility)

Let X be an itemset. Let the extensions of X be the itemsets that can be obtained by appending an item i to X such that i x, ∀x ∈ X. The remaining utility upper-bound of an itemset X is defined as reu(X) = u(X) + re(X). If u(X) + reu(X) < minutil, then X is a low-utility itemset as well as all its extensions.

A crucial problem in HUIM is that the set of HUIs is often very large. To address this issue, it was proposed to mine the concise and representative subset of closed HUIs. But mining CHUIs

can be very computationally expensive. To address this issue, we next introduce a novel more efficient algorithm.

IV. THE PROPOSED ALGORITHM

The proposed algorithm is a highly efficient algorithm for closed HUI mining. It is a one phase algorithm designed using the strict design constraint that for each itemset in the search space, all operations for that itemset should be performed in linear time and space. This section is organized as follows.

4.1 The Search Space

Using set-enumeration tree the search space of all itemsets can be represented. For example, the set-enumeration tree of I = {a, b, c, d} for the lexicographical order is shown in Fig. 1. The proposed algorithm explores this search space using a depth-first search starting from the root (the empty set). During this depth-first search, for any itemset α, EFIM-Closed recursively appends one item at a time to α according to the order, to generate larger itemsets. In our implementation, the order is defined as the order of increasing TWU because it generally reduces the search space for HUIM. We next introduce definitions related to the depth-first search exploration of itemsets.

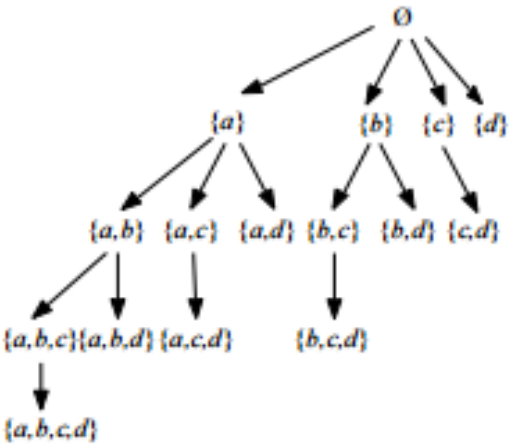

Figure 1: Set Enumeration Tree

Let E(α) denote the set of all items that can be used to extend α according to the depth-first search, that is E(α) = {z|z ∈ I ∧ z x, ∀x ∈ α}. An itemset Z is an extension of α (appears in a sub-tree of α in the set-enumeration tree) if Z = α ∪ W for an itemset W ∈ 2 E(α) such that W 6= ∅. An itemset Z is a single-item extension of α (is a child of α in the set enumeration tree) if Z = α ∪ {z} for an item z ∈ E(α).

4.2 Scanning the Database Efficiently

As we will later explain, EFIM-Closed performs database scans to calculate the utility of itemsets and upper-bounds on their utility. To reduce the cost of database scans, it is desirable to reduce the database size. In EFIM-Closed this is performed by two techniques.

High-utility Database Projection (HDP).

This technique is based on the observation that when an itemset α is considered during the depth-first search,all x€ E(α) can be ignored when scanning the database to calculate the utility of itemsets in the sub-tree of α, or upper-bounds on their utility. A database without these items is called a projected database.

Projected database

The projection of a transaction T using an itemset α is denoted as α-T and defined as α-T = {i|i ∈ T ∧ i ∈ E(α)}. The projection of a database D using an itemset α is denoted as α-D and defined as the multiset α-D = {α-T |T ∈ D ∧ α-T ≠ ∅}.

High-utility Transaction Merging (HTM)

To further reduce the cost of database scans, EFIM-Closed also introduce an efficient transaction merging technique named High-utilty Transaction Merging (HTM). HTM is based on the observation that transaction databases often contain identical transactions (transactions containing exactly the same items, but not necessarily the same internal utility values). The technique consists of replacing a set of identical transactions T r_1, T r_2, ...T r_m in a (projected) database α-D by a single new transaction TM = T r_1 = Tr_2 = ... = Tr_m where the quantity of each item i ∈ TM is defined as q(i, TM) = P k=1...m q(i, Tr_k).

4.3 Pruning Low-Utility Itemsets

To propose an efficient CHUI mining algorithm, a key problem is to design an effective mechanism for pruning low-utility itemsets in the search space. For this purpose, we introduce two new upper-bounds on the utility of itemsets. The subtree-utility and local utility upper-bounds. The first upperbound is defined as follows.

4.4 Pruning Non Closed HUIs

We now explain the techniques used by EFIM-closed to prune non closed HUIs. To find only CHUIs, a naive approach would be to keep all HUIs found until now into memory. Then, every time that a new HUI is found, the algorithm would compare the HUI with previously found HUIs to determine if (1) the new HUI is included in a previously found HUI or (2) if some previously found HUI(s) are included in the new HUI. The drawback of this approach is that it can consume a large amount of memory if the number of patterns is large, and it becomes very time consuming if a very large number of HUIs is found, because a very large number of comparisons would have to be performed. In this paper, we present new checking mechanisms that can determine if a HUI is closed without having to compare a new pattern with previously found patterns. It is inspired by a similar mechanism used in sequential pattern mining [13]. The mechanism is based on two separate checks, which we respectively name backward extension checking check and forward-extension checking, and are defined as follows.

4.5 The Algorithm

In this subsection, we present the proposed EFIM-Closed algorithm, which combines all the ideas presented in the previous subsections. The main procedure (Algorithm 1) takes as input a transaction database and the minutil threshold. The algorithm initially considers that the current itemset α is the empty set. The algorithm then scans the database once to calculate the local utility of each item w.r.t. α, using a utility-bin array. Then, the local utility of each item is compared with minutil to obtain the secondary items w.r.t to α that is items that should be considered in extensions of α. Then, these items are sorted by ascending order of TWU and that order is thereafter used as the order. The database is then scanned once to remove all items that are not secondary items w.r.t to α since they cannot be part of any high utility itemsets. If a transaction becomes empty, it is removed from the database. Then, the database is scanned again to sort transactions by the T order to allow O(nw) transaction merging, thereafter. Then, the algorithm scans the database again to calculate the sub-tree utility of each secondary item w.r.t. α, using a utility-bin array. Thereafter, the algorithm calls the recursive procedure Search to perform the depth first search starting from α.

The Search procedure (Algorithm 2) takes as parameters the current itemset to be extended α, the projected database, the primary and secondary items w.r.t α and the min-util threshold. The procedure performs a loop to consider each single-item extension of α of the form $\beta = \alpha \cup \{i\}$, where i is a primary item w.r.t α (since only these single-item extensions of α should be explored according to Theorem 1). For each such extension β, a database scan is performed to calculate the utility of β and at the same time construct the β projected database. Note that transaction merging is performed whilst the β projected database is constructed. If β has a backward extension, no extensions of β will be explored. Otherwise, the projected database of β is scanned to calculate the support, sub-tree and local utility w.r.t β of each item z that could extend β (the secondary items w.r.t to α), using three utility-bin arrays. This allows determining the primary and secondary items of β. If all items that can extend β have the same support as β, the closure jumping optimization is performed to directly output $\beta \cup S \{z\}$ if it is a HUI and prune the subtree of β. Otherwise, the Search procedure is recursively called with β to continue the depth-first search by extending β. If no extension of β have the same support as β and the utility of β is no less than min-util, β is output as a CHUI (by Property 3). Based on properties and theorems presented in previous sections, it can be seen that when EFIM-Closed terminates, all and only the CHUIs have been output.

Algorithm 1: The EFIM-Closed algorithm

Input : D is a transaction database, minuti is a user-specified threshold

Output: the set of high-utility itemsets

Step 1: $\alpha = \emptyset$;

Step 2: Calculate $lu(\alpha, i)$ for all items $i \in I$ by scanning D, using a utility-bin array;

Step 3: Secondary(α) = $\{i | i \in I \wedge lu(\alpha, i) \geq minutil\}$;

Step 4: Let be the total order of TWU ascending values on Secondary(α);

Step 5: Scan D to remove each item i 6∈ Secondary(α) from the transactions, and delete empty transactions;

Step 6: Sort transactions in D according to T ;

Step 7: Calculate the sub-tree utility $su(\alpha, i)$ of each item $i \in$ Secondary(α) by scanning D, using a utility-bin array;

Step 8: Primary(α) = $\{i | i \in$ Secondary(α) $\wedge su(\alpha, i) \geq minutil\}$;

Step 9: Search (α, D, P rimary(α), Secondary(α), minutil);

Complexity: The complexity of EFIM-Closed is briefly analyzed as follows. In terms of time, a O(nw log(nw)) sort is performed. This cost is negligible since it is performed only once. Then, to process each primary itemset α encountered during the depth-first search, EFIM-Closed performs database projection, transaction merging, backward/forward extension checking and upper-bound calculation in linear time and space (O(nw)). Thus, the time complexity of EFIMClosed is proportional to the number of itemsets in the search space, and it is linear for each itemset.

Algorithm 2: The Search procedure

Input : α: an itemset, α-D: the α projected database, Primary(α): the primary items of α, Secondary(α): the secondary items of α, the minutil threshold

Output: the set of high-utility itemsets that are extensions of α

Step 1: foreach item i ∈ P rimary(α) do
Step 2: β = α ∪ {i};
Step 3: Scan α-D to calculate u(β) and create β-D; // with transaction merging
Step 4: if β has no backward extension then
Step 5: Calculate sup(β, z), su(β, z) and lu(β, z) for all item z ∈ Secondary(α) by scanning β-D once, using three utility-bin arrays;
Step 6: if sup(β) = sup(β ∪ {z})∀z i ∧ z ∈ E(α) then
Output β ∪ S zi∧z∈E(α) {z} if it is a HUI; // closure jumping
Step 7: else
Primary(β) = {z ∈ Secondary(α)|su(β, z) ≥ minutil};
Step 10: Secondary(β) = {z ∈ Secondary(α)|lu(β, z) ≥ minutil};
Step 11: Search (β, β-D, P rimary(β), Secondary(β), minutil);
Step 12: if β has no forward extension and u(β) ≥ minutil then output β;
Step 13: end

V. EXPERIMENTAL RESULTS

The performed experiments to evaluate the performance of the proposed algorithm. The performance of EFIM-Closed was compared with the state-of-the-art CHUD algorithm. Experiments were performed using standard datasets used in the HUIM literature for evaluating HUIM algorithms, namely Accident, BMS, Chess, Connect, Foodmart and Mushroom. These datasets have varied characteristics representing the main types of databases (sparse, dense, long transactions). For these datasets, the number of transactions/number of distinct items/average transaction length are: Accident (340,183 / 468 / 33.8), BMS (59,601 / 497 / 4.8), Chess (3,196 / 75 / 37.0), Connect (67,557 / 129 / 43.0), Foodmart (4,141 / 1,559 / 4.4), Mushroom (8,124 / 119 / 23.0). Foodmart contains real external/internal utility values.

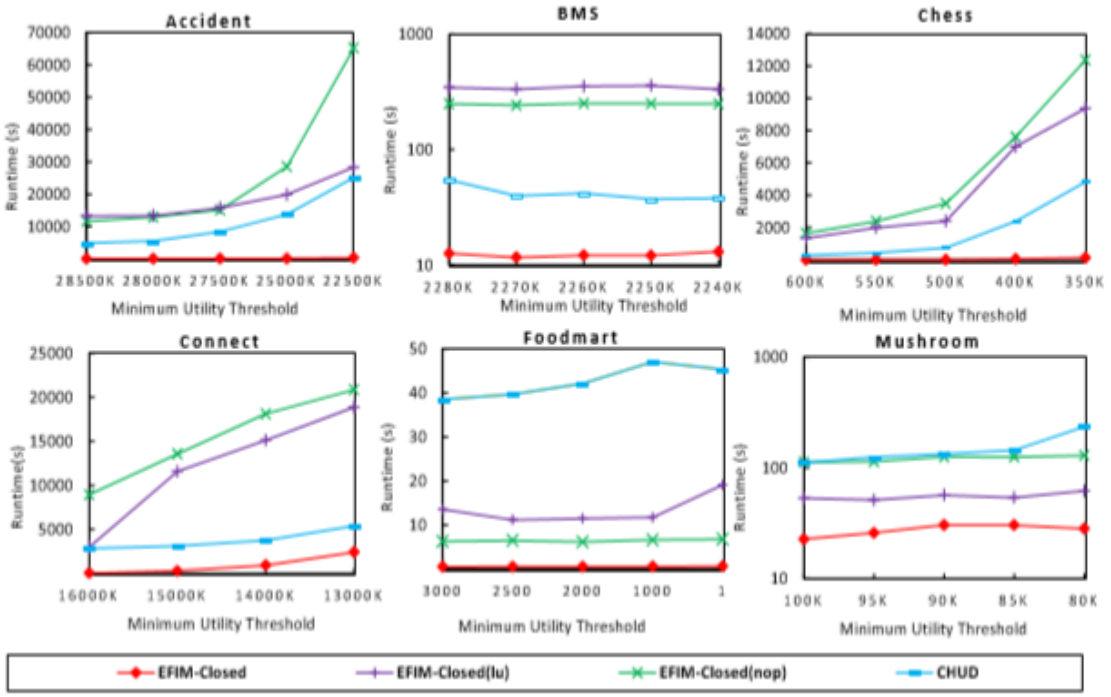

Figure 2: Execution Times on different datasets

VI. CONCLUSION

In this paper, an efficient algorithm proposed for closed high utility itemset mining. It relies on two new upper-bounds named sub-tree utility and local utility, and an array-based utility counting approach named Fast Utility Counting. Moreover, to reduce the cost of database scans, EFIM-Closed proposes two efficient techniques named High-utility Database Projection and High-utility Transaction Merging. Lastly, to discover only closed HUIs, three mechanisms are proposed forward closure checking, backward closure checking and closure jumping. Experimental results shows that EFIM-Closed can be faster and consumes less memory than the state-of-the-art CHUD algorithm.

REFERENCES

[1] C. Lin, T. Hong, G. Lan, J. Wong, and W. Lin, "Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases", Adv. Eng. Informat., vol. 29, no. 1, pp. 16–27, 2015.

[2] Agrawal , R., Srikant, R., "Fast algorithms for mining association rules in large databases". In: Proc. Int. Conf. Very Large Databases, pp. 487–499, (1994)

[3] H. Ryang and U. Yun, "Top-k high utility pattern mining with effective threshold raising Strategies", Knowledge Based Syst., vol. 76, pp. 109–126, 2015.

[4] H. Ryang, U. Yun and K. Ryu, "Discovering high utility itemsets with multiple minimum supports" , Intell. Data Anal., vol. 18, no. 6, pp. 1027–1047, 2014.

[5] UnilYun , HeungmoRyang ,Gangin Lee and HamidoFujita , "An efficient algorithm for mining high utility patterns from incremental databases with one database scan", Knowledge-Based Systems Volume 124, 15 May 2017, Pages 188-206.

[6] U. Yun and H. Ryang, "Incremental high utility pattern mining with static and dynamic databases", Appl. Intell., vol. 42, no. 2, pp. 323–352, 2015.

[7] Zida, S., Fournier Viger, P., "Efficient mining of high utility sequential rules", In: Proc. 11th Intern. Conf. Machine Learning and Data Mining (MLDM 2015), pp. 1–15 (2015)

[8] U. Yun and J. Kim, "A fast perturbation algorithm using tree structure for privacy preserving utility mining", Expert Syst. Appl., vol. 42, no. 3, pp. 1149–1165, 2015.

[9] Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., Lee, Y.-K., "Efficient tree structures for high-utility pattern mining in incremental databases", IEEE Trans. Knowl. Data Eng. 21(12), 1708–1721 (2009)

[10] Fournier-Viger, P., Zida, S. and Tseng V. S., "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning", In: Proc. 21st Intern. Symp. On Methodologies for Intell. Syst., pp. 83–92 (2014)

[11] G.-C. Lan, T.-P. Hong, and V. S. Tseng, "An efficient projection based indexing approach for mining high utility itemsets", Knowl. Inf. Syst, vol. 38, no. 1, pp. 85–107, 2014.

[12] Yun, U., Ryang, H., Ryu, K. H., "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates", Expert Syst. with Appl. 41(8), 3861–3878 (2014)

