Fast Mining Algorithm for Frequent High-Utility Itemsets with Multiple Minimum Utility Thresholds Postprint

Authors: Wang Bin, Lü Ruirui, Fang Xinxiu, Ma Junjie

Date: 2018-10-11T00:00:00+00:00

Abstract

To address the problems of redundant computation and infrequent resulting itemsets in the Multi-Minimum Utility threshold High Utility itemset mining algorithm (MHUI), two novel efficient mining algorithms, FMHUI and SFMHUI, are proposed. The FMHUI algorithm utilizes previous computation results when calculating the minimum utility threshold for itemsets, thereby avoiding repeated comparisons among items. Additionally, an Extension Minimum Utility table (EMMU-table) is defined for item extension items to rapidly compute their minimum utility thresholds, which enhances operational efficiency. The SFMHUI algorithm incorporates support constraints on the basis of FMHUI, ensuring that the mined itemsets are both high-utility and frequent. The efficiency and feasibility of the proposed algorithms are validated through simulation experiments.

Full Text

Preamble

Fast Mining Algorithm for Frequent and High Utility Itemsets with Multiple Minimum Utility Thresholds

Wang Bin, Lyu Ruirui, Fang Xinxiu, Ma Junjie
(School of Information & Control Engineering, Qingdao Technological University, Qingdao, Shandong 266033, China)

Abstract: The mining algorithm for high utility itemsets with multiple minimum utility thresholds (MHUI) suffers from repeated calculations and produces itemsets that are not necessarily frequent. To address these issues, this paper proposes two novel fast mining algorithms: FMHUI and SFMHUI. The FMHUI algorithm leverages previous calculation results when computing the minimum

utility threshold of itemsets, thereby avoiding redundant comparisons between items. Additionally, it defines the Extended Minimum Utility threshold table (EMMU-table) for item extensions to rapidly calculate the minimum utility thresholds of extended items, significantly improving computational efficiency. The SFMHUI algorithm builds upon FMHUI by incorporating support constraints, ensuring that the mined itemsets are both high-utility and frequent. Simulation experiments validate the efficiency and feasibility of the proposed algorithms.

Keywords: frequent itemsets; high utility itemsets; support; multiple minimum utility thresholds

0 Introduction

High utility itemset mining, first introduced by Chan et al. [1–4] in 2003, has found extensive real-world applications in market basket analysis [4,5], website clickstream analysis [6], cross-marketing in retail stores, and healthcare and biomedical applications [5,7,8]. Early algorithms such as HUC-Prune [9], UP-Growth+ [10], MU-Growth [11], HUI-Miner [12], FHM [13], HUP-Miner [14], and EFIM [15] all employed a single utility threshold. However, this approach fails to account for the diversity among items, which is often inappropriate and unfair in practical scenarios. Consequently, mining high utility itemsets with multiple minimum utility thresholds has emerged as a more realistic solution that considers item differences by assigning a distinct minimum utility threshold to each item.

Existing multi-threshold algorithms include HUI-MMU, HUI-MMUTID, and HUI-MMUTE [16], which, like Apriori [17], generate excessive candidate itemsets and require multiple database scans. Another approach [18] utilizes a Multiple Item Utility tree (MIU-tree) and utility list structures in HIMU and HIMU-EUCP algorithms to avoid additional database scans and candidate generation. A common characteristic of these algorithms is that items are processed in ascending order of their minimum utility thresholds, which eliminates comparisons between item thresholds when computing the minimum utility threshold of an itemset. Krishnamoorthy [19] later proposed the MHUI (high utility itemsets with multiple minimum utility thresholds) algorithm, which employs a utility list structure and four pruning strategies to enhance efficiency. However, MHUI processes items in ascending order of transaction weighted utility (TWU) [18–20] and demonstrates that TWU-based ordering yields higher efficiency than minimum utility threshold-based ordering (the latter increases candidate itemset count despite avoiding threshold comparisons).

Nevertheless, the MHUI algorithm suffers from a critical drawback: when calculating the minimum utility threshold for each itemset and its extensions, it repeatedly performs comparisons among all item thresholds, leading to redundant computations that degrade mining efficiency. To address this limitation,

this paper proposes the FMHUI (fast mining high utility itemsets with multiple minimum utility thresholds) algorithm, which incorporates four pruning properties for improvement. Furthermore, while existing algorithms only guarantee that mined itemsets are high-utility, they cannot ensure frequency. Since both high utility itemset mining and frequent itemset mining hold significant importance in data mining [17,21], and considering their respective limitations—high-support itemsets may have low utility, and high-utility itemsets may have low support—this paper introduces support constraints. For instance, in the sample database (Tables 1 and 2), with a minimum support threshold of 2/6, itemset {a,f} has utility U({a,f})=18>15 (high-utility) but support sup({a,f})=1/6<2/6 (not frequent), while {a,d} is frequent but not high-utility. To resolve this, we propose the SFMHUI (FMHUI with support constraints) algorithm, which combines the efficiency improvements of FMHUI with support constraints to mine frequent and high utility itemsets.

1 Related Definitions and Problem Statement

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of m distinct items. Any $X \subseteq I$ is called an itemset. Each item $x_i \in T_j$ is characterized by its external utility $EU(x_i)$ and internal utility $IU(x_i, T_j)$. The database $D = \{T_1, T_2, ..., T_n\}$ contains n transactions. A k-itemset $X = \{x_1, x_2, ..., x_k\}$ where $X \subseteq I$ is called a K-itemset. A sample database D is shown in , with each item' s external utility $EU(x_i)$, minimum utility $MU(x_i)$, and transaction weighted utility (TWU) presented in .

Definition 1. The utility of item x_i in transaction T_j is denoted as $U(x_i, T_j)$ and defined as:

$$U(x_i, T_j) = EU(x_i) \times IU(x_i, T_j)$$

Definition 2. The utility of itemset X in transaction T_j is denoted as $U(X, T_j)$ and defined as:

$$U(X, T_j) = \sum_{x_i \in X \subseteq T_j} U(x_i, T_j)$$

Definition 3. The utility of itemset X in database D is denoted as $U(X)$ and defined as:

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

Definition 4. The utility of transaction T_j is denoted as $TU(T_j)$ and defined as:

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

Definition 5. The transaction weighted utility of itemset X is denoted as $TWU(X)$ and defined as:

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

Definition 6. The multiple minimum utility thresholds for all items are represented by MMU (multiple minimum utility) and defined as:

$$MMU = \{MU(i_1), MU(i_2), ..., MU(i_m)\}$$

where m is the number of distinct items. For example, for database D, $MMU = \{25, 30, 17, 28, 22, 15\}$.

Definition 7. The least minimum utility threshold among all items is denoted as LMU (least minimum utility threshold) and defined as:

$$LMU = \min\{MU(i_1), MU(i_2), ..., MU(i_m)\}$$

Definition 8. The minimum utility threshold of a k-itemset $X = \{x_1, x_2, ..., x_k\}$ is denoted as $MIU(X_k)$ and defined as:

$$MIU(X_k) = \min\{MU(x_1), MU(x_2), ..., MU(x_k)\}$$

Definition 9. Item ordering: All items are sorted in ascending order of their transaction weighted utility. As shown in , the reordered items for database D are presented in (note that item f is removed from subsequent use because $sup(f) = 1/6 < 2/6$).

Definition 10. The items following itemset X in transaction T_j are denoted as T_j/X, and their utility is called the remaining utility, denoted as $RU(T_j/X)$ and defined as:

$$RU(T_j/X) = \sum_{x_i \in T_j \wedge x_i \notin X} U(x_i, T_j)$$

For example, $RU(T_6/\{e, c\}) = U(b, T_6) + U(d, T_6) = 28$.

Definition 11. After sorting items in the database according to Definition 9, all items following item x_i are defined as extensions of x_i. The minimum utility threshold of item x_i and its extensions is denoted as $EMU(x_i)$. The extended minimum utility thresholds for all items are stored in the $EMMU$-table (extended multiple minimum utility threshold table for items), defined as:

$$EMMU\text{-table} = \{EMU(i_1), EMU(i_2), ..., EMU(i_m)\}$$

Notably, the $EMMU$-table is constructed in reverse order of the sorted items. For instance, with item order $a \prec e \prec c \prec b \prec d$, we first compute $EMU(d) = 28$, then $EMU(b) = \min\{EMU(d), MU(b)\} = 28$; similarly,

$EMU(c) = \min\{EMU(b), MU(c)\} = 17$, $EMU(e) = 17$, and $EMU(a) = 17$. Thus, the $EMMU$-table for is $\{17, 17, 17, 20, 20\}$.

Definition 12. Using itemset extensions, a global minimum utility threshold GMU (global minimum utility threshold) is introduced, defined as:

$$GMU(X) = \min\{MIU(X), EMU(x_k)\}$$

where x_k is the last item in itemset X. For example:

$$GMU(\{a, c\}) = \min\{MIU(\{a, c\}), EMU(c)\} = \min\{17, 17\} = 17$$

Definition 13. The support of itemset X, denoted as $sup(X)$, is the ratio of transactions containing X to the total number of transactions in the database.

Definition 14. The 2-itemset support matrix for all items in the database is represented by the estimated support co-occurrence structure $ESCS$, defined as:

$$ESCS = \{sup(\{x_i, x_j\}) \mid x_i, x_j \in I\}$$

This triangular matrix stores support values for all 2-itemsets, analogous to the estimated utility co-occurrence structure $EUCS$ [13,14].

Problem Statement: The objective of this paper is to mine all frequent and high utility itemsets ($FHUIs$) from the database, defined as:

$$FHUIs = \{X \mid X \subseteq I, sup(X) \geq minsup, U(X) \geq MIU(X)\}$$

---

2 SFMHUI Algorithm

Since SFMHUI is an extension of FMHUI with support constraints, this section focuses on the search space, pruning properties, and main workflow of SFMHUI. We first describe the data storage structure, which extends the utility list structure from [12–14,19] by incorporating support-related statistics (the total number of transaction IDs for each itemset). The initial (1-itemset) utility lists are shown in [Figure 1: see original paper].

2.1 Search Space

The search space for high utility itemset mining can be represented by a set-enumeration tree. For database D () with itemset $I = \{a, b, c, d, e\}$, all itemsets are represented by the set-enumeration tree shown in [Figure 2: see original paper]. SFMHUI employs depth-first search, enabling the minimum utility threshold of a $(k + 1)$-itemset to be computed using $GMU(X) = \min\{MIU(X), EMU(x_{k+1})\}$ rather than recomparing all item thresholds, thereby reducing redundant calculations and improving efficiency.

2.2 Main Pruning Strategies

SFMHUI mines frequent and high utility itemsets, requiring both support and utility constraints to be satisfied. We adapt pruning properties from high utility and frequent itemset mining [13,16] to develop SFMHUI' s pruning strategies:

Property 1 (Anti-monotonicity of Support). If $sup(X) < minsup$, then all supersets of X have support less than $minsup$ [17].

Property 2 (Item-Prune). If an item x_i has $TWU(x_i) < LMU$, then no itemset containing x_i can be a frequent high utility itemset.

Property 3 (SU-Prune). If an itemset X satisfies $U(X) + RU(X) < GMU(X)$, then X and all its supersets are not frequent high utility itemsets. The support constraint follows from Property 1.

Property 4 (EC-Prune). If $U(X) < GMU(X)$, then no superset of X is a frequent high utility itemset.

Property 5. Given two itemsets X and Y, if $X \cap Y = \emptyset$ and $GMU(X \cup Y) > EUCS(X, Y)$, then no superset of $X \cup Y$ is a frequent high utility itemset. Proof follows [19].

These four pruning strategies effectively improve the efficiency of frequent high utility itemset mining.

2.3 SFMHUI Algorithm Description

SFMHUI redefines the minimum utility threshold calculation by leveraging previous results to avoid recomputing thresholds for already-processed itemsets. The $EMMU$-table enables rapid determination of extension thresholds, improving GMU computation efficiency. Support constraints ensure the mining of frequent high utility itemsets.

Algorithm 1: SFMHUI (Main) *Input:* Transaction database D, minimum utility thresholds MMU for all items, minimum support threshold $minsup$
Output: Frequent high utility itemsets $FHUIs$

The algorithm scans the database twice. The first scan computes TWU and support sup for all 1-itemsets. The second scan (steps 5–6) performs: (a) pruning unpromising 1-itemsets using Item-Prune (Property 2) and recomputing transaction utilities TU; (b) constructing $EUCS$, $ESCS$ structures, and 1-itemset utility lists ([Figure 1: see original paper]). These utility lists then generate the itemset search tree and mine frequent high utility itemsets.

Algorithm 2: Explore-Search-Tree and **Algorithm 3: Construct-Utility-List** handle the search tree mining and utility list construction processes. Algorithm 2 primarily uses Properties 3 and 4 for pruning, while Algorithm 3 details utility list construction [12,14,18] and applies Property 5.

3 Experimental Analysis

Two simulation experiments validate the proposed algorithms' efficiency and feasibility: (1) FMHUI vs. MHUI, and (2) SFMHUI vs. SMHUI (SMHUI is MHUI with the same support constraints and frequent itemset pruning strategies as SFMHUI). The experimental environment: Windows 7 (64-bit), Eclipse IDE, Java programming language, 4 GB RAM, Intel i5 processor.

3.1 Experimental Design

Evaluations were conducted on the Accidents and Mushroom datasets from the SPMF library [23]. Dataset characteristics are shown in . Internal utilities were randomly generated as integers between 1-10, while external utilities followed a log-normal distribution between 0.01-10 [12-14]. Minimum thresholds for each item were assigned following [16,18,19]:

$$MU(x_i) = GLMU + \beta \times \max(EU)$$

where $GLMU$ is a user-defined global least minimum utility threshold and β is a constant. When $\beta = 0$, all items share the same threshold, reducing to the common single-threshold scenario.

3.2 Experimental Results

FMHUI vs. MHUI: [Figure 3: see original paper] and [Figure 4: see original paper] show the results. With constant β (500k for Accidents, 10k for Mushroom), FMHUI achieves average time efficiency improvements of 25.4% and 30.6% over MHUI. With constant $GLMU$ (12M for Accidents, 10k for Mushroom), improvements are 14% and 14.8%, respectively. These gains stem from FMHUI' s use of Definitions 8 and 11 to eliminate redundant calculations.

SFMHUI vs. SMHUI: [Figure 5: see original paper] and [Figure 6: see original paper] present the results, focusing on $GLMU$ with β fixed at 500k (Accidents) and 50k (Mushroom). [Figure 5: see original paper] varies minimum support ($minsup$) with $GLMU$ fixed at 14M and 100k. [Figure 6: see original paper] varies $GLMU$ with fixed support (0.46 for Accidents, 0.1 for Mushroom). SFMHUI consistently outperforms SMHUI, confirming the effectiveness of our improvements.

[Figure 7: see original paper] and [Figure 8: see original paper] analyze the number of frequent high utility itemsets ($FHUIs$) under different support thresholds. When $minsup = 0$ (no support constraint), the count is highest. As $minsup$ increases, $FHUI$ count decreases, validating SFMHUI' s feasibility. Support constraints yield more decision-relevant high utility itemsets.

4 Conclusion

The FMHUI algorithm resolves redundant computation and comparison issues in MHUI. The SFMHUI algorithm further addresses the non-frequency problem by adding support constraints to FMHUI. Both algorithms employ four pruning properties to enhance efficiency, validated through simulation experiments. Future work will focus on reducing memory consumption through parallel computing and exploring prefix utility tree-node list (PUN-list) structures to further improve efficiency.

References

[1] Chan R, Yang Qiang, Shen Yidong. Mining high utility itemsets [C]// Proc of IEEE International Conference on Data Mining. Washington DC: IEEE Computer Society, 2003: 19-29.

[2] Duong Q H, Liao B, Fournier-Viger P, et al. An efficient algorithm for mining the top-k, high utility itemsets, using novel threshold raising and pruning strategies [J]. Knowledge-Based Systems, 2016, 104: 106-122.

[3] Lin C, Fournier-Viger P, Gan W. FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits [J]. Knowledge-Based Systems, 2016, 111: 283-298.

[4] Tseng V S, Wu C, Shie B E, et al. UP-Growth: an efficient algorithm for high utility itemset mining [C]// Proc of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington DC: IEEE Computer Society, 2010: 253-262.

[5] Zida S, Fournier-Viger P, Lin C, et al. EFIM: a fast and memory efficient algorithm for high-utility itemset mining [J]. Knowledge & Information Systems, 2017, 51(2): 595-625.

[6] Thilagu M, Nadarajan R. Efficiently mining of effective Web traversal patterns with average utility [J]. Procedia Technology, 2012, 6(4): 444-451.

[7] Lee D, Park S H, Moon S. Utility-based association rule mining: a marketing solution for cross-selling [J]. Expert Systems with Applications, 2013, 40(7): 2715-2725.

[8] Lin C, Fournier-Viger P, Gan W. FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits [J]. Knowledge-Based Systems, 2016, 111: 283-298.

[9] Ahmed C F, Tanbeer S K, Jeong B S, et al. HUC-Prune: an efficient candidate pruning technique to mine high utility patterns [J]. Applied Intelligence, 2011, 34(2): 181-198.

[10] Tseng V S, Shie B E, Wu C, et al. Efficient algorithms for mining high utility itemsets from transactional databases [J]. IEEE Trans on Knowledge & Data Engineering, 2013, 25(8): 1772-1786.

[11] Yun U, Ryang H, Ryu K H. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates [J]. Expert Systems with Applications, 2014, 41(8): 3861-3878.

[12] Liu M, Qu J. Mining high utility itemsets without candidate generation [C]// Proc of ACM International Conference on Information and Knowledge Management. New York: ACM Press, 2012: 55-64.

[13] Fournier-Viger P, Wu C, Zida S, et al. FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning [C]// Proc of International Symposium on Methodologies for Intelligent Systems: Foundations of Intelligent Systems. Switzerland: Springer International Publishing, 2014: 83-92.

[14] Krishnamoorthy S. Pruning strategies for mining high utility itemsets [J]. Expert Systems with Applications, 2015, 42(5): 2371-2381.

[15] Zida S, Fournier-Viger P, Lin C, et al. EFIM: a highly efficient algorithm for high-utility itemset mining [C]// Proc of Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence and Soft Computing. Switzerland: Springer International Publishing, 2015: 530-543.

[16] Lin C, Gan W, Fournier-Viger P, et al. Efficient mining of high-utility itemsets using multiple minimum utility thresholds [J]. Knowledge-Based Systems, 2016, 113: 100-115.

[17] Han Jiawei, Kamber M, Pei Jian. Data mining concepts and techniques [M]. Ming Fan, Xiaofeng Meng, Trans. 3rd Edi. Beijing: China Machine Press, 2007: 157-170.

[18] Gan W, Lin C, Fournier-Viger P, et al. More efficient algorithms for mining high-utility itemsets with multiple minimum utility thresholds [C]// Proc of the 27th International Conference on Database and Expert Systems Applications. Switzerland: Springer International Publishing, 2016: 71-87.

[19] Krishnamoorthy S. Efficient mining of high utility itemsets with multiple minimum utility thresholds [J]. Engineering Applications of Artificial Intelligence, 2018, 69: 112-126.

[20] Ru B, He X. High utility itemsets mining algorithm of data stream with reducing candidate itemsets [J]. Application Research of Computers, 2017, 34(11): 3379-3383.

[21] Sahoo J, Das A K, Goswami A. An efficient approach for mining association rules from high utility itemsets [J]. Expert System with Applications, 2015, 42(13): 5754-5778.

[22] Lin C, Gan W, Fournier-Viger P, et al. Efficient algorithms for mining high-utility itemsets in uncertain databases [J]. Knowledge-Based Systems, 2016, 96:

171-187.

[23] Fournier-Viger P, Gomariz A, Soltani A, et al. SPMF: Open-Source data mining platform [EB/OL]. http://www.philippe-fournier-viger.com/spmf.

Note: Figure translations are in progress. See original paper for figures.