International Journal of Modern Computer Science (IJMCS) ISSN: 2320-7868 (Online)
Volume 7, Issue 3, June, 2019

TSM: TWU and Suffix-based High Utility Itemset Mining Algorithm

SonamPanwar
Department of Computer Science, UIET
Kurukshetra University
Kurukshetra, India
E-mail: panwarsonam12@gmail.com

Ajay Jangra
Department of Computer Science, UIET
Kurukshetra University
Kurukshetra, India
E-mail: er_jangra@yahoo.co.in

Abstract: Utility mining considers individual items different according to their respective utilities and mines items with high utilities. This mining task is commonly known as High Utility Itemset (HUI) mining. To do so, most available algorithms first generate large number of candidates among which many are eventually found to be low utility itemsets. Moreover, a large amount of time is consumed in join operations needed to generate larger itemsets from smaller ones. In this paper, we present a fast Hash-Map based algorithm named TSM that efficiently prunes low-utility itemsets using co-occurrence information. The overall impact is a considerable reduction in join operations and hence faster execution.

Keywords: Frequent itemset, high utility itemset, utility mining, mining algorithm

I. INTRODUCTION

Data mining approaches are used to mine hidden, interesting knowledge and facts from large number of transactions present in data-sets. Association rule mining (ARM) is the most commonly used mining approach to extract frequent itemsets and hidden co-relations amongst itemsets in the transactions and has great application in many real-life domains like business predictions, retail management, supply control, etc. [1].

Frequent itemsetmining(FIM) [2] provides the base for association rules mining (ARM), and is also useful in other mining tasks. FIM algorithms deal with DBs that denote the presence of an item in a transaction as a binary value without seeing its importance or utility (based on quantity, price or profit). This scenario doesn't reflect the real-life situation where items have their individual importance other than just frequency of occurrence in transactions. FIM algorithms utilize "downward closure property" or "anti-monotonic property" to reduce search space by pruning away unwanted itemsets as the property says that "any superset of an infrequent itemset is also infrequent".

Thus, while dealing with real world issues with the aim to maximize an organization's profit, utility plays an essential role. Utility mining provides solution to above discussed issues and is significant in many practical domains [3]. Utility measure tells about significance of individual items present in the DB other than its presence. Thus, utility mining aims at finding high-utility itemsets that contributes to a large part of overall utility and hence called High Utility Itemset Mining (HUIM). The problem with utility mining approaches is due to the fact that it doesn't follows "downward closure property". The heuristics proposed in [3] known as MEU (Mining using Expected Utility) helps to decide whether to include an itemset as candidate for next phases of mining. But this model generally leads to "over-estimation" at starting causing issues related to memory requirement and execution time of most algorithms. Thus, challenges with utility mining can be summarized as:

- Discovering complete set of high utility items
- Reducing the over-estimation of utilities for candidate set
- Reducing execution time of algorithms when transaction size increases

In an effort to deal with these problems many algorithms have been proposed as in [4], [5], [6] and [7]. Among these, HUI-Miner being the most popular one, mines HUIs in single phase but it need to use costly join operations affecting execution time of the algorithm [7].

In a step towards improving with the execution time of the HUI-Miner, we propose an efficient algorithm named TSM (TWU and Suffix based HUI Mining) algorithm to mine HUIs using hash-maps to store TWU information about the suffix. As a proof, experiments with dense, synthetic datasets have been performed to show the considerable improvement in working of previous algorithm in terms of execution time. The rest of the paper is organized as follows. The Section 2 involves problem definition and an insight into related works. Section 3 presents proposed solution and the related performance results will be involved in Section 4. The overall work is summoned in Section 5.

II. Background

This section involves major issues related to HUI mining and then present related works done to solve handle these issues.

A. Problem Definition

Let us take an example of a DB in Figure1. having six transactions in transaction table and five items in utility table. The support of each item can be calculated simply by looking at the total number of transactions in which the item is present. Thus, to calculate support we don't need utility table and last column of transaction table.

As discussed earlier, support count follows anti-monotonic property and thus, support of itemsets {p}, {pq}, {pqr}, {pqrs} and {pqrst} are respectively 3, 2, 2, 2, 1 i.e. either decreasing or unchanged. But this is not true in case of utility mining as {p} has utility 5, {pq} is 9 and {pqrst} is 24. So, we

Table1. Transaction table

TID	Transaction	Utility
T1	p, q, r, s	1, 2, 1, 1
T2	p, r, s, t	2, 2, 3, 1
T3	q, r, s, t	1, 2, 1, 2
T4	q, s, t	1, 1, 1
T5	p, q, r, s, t	2, 1, 3, 1, 2
T6	r, s, t	3, 2, 3

Item	p	q	r	s	t
Utility	1	2	4	2	3

Table2. Utility table

can clearly see that utility mining, which will be discussed in detail later, don't follow anti-monotonic property. This makes it difficult to use FIM pruning approach in HUIM. It aims at finding itemsets whose utility count satisfies a user-specified, minimum utility threshold i.e. min_util [4]. For example, if we set min_util = 20, then from above shown example {p} and {pq} are low-utility itemsets wheras {pqrst} is a high utility itemset.

Let I be the set of unique items that occur in any transaction T of a DB, i.e. DB={T1, T2, …, Tn} where T={I1, I2, …, Im}. An itemset I having K items is called K-itemset. The formal definitions associated with utility mining problem for such scenario are as follows:

Definition 1: Itemset: I = {I_1, I_2,…..,I_m} is the set of unique items that occurred in the transactions of DB

Definition 2: Database: DB = {T_1, T_2,….,T_n} is a set of transactions such that Ti⊂ I.

Definition 3: Internal Utility: IU(I_j, T_k) represents the number of occurrences of item $I_j$in transaction T_k.. For example, IU(q,T3) = 1.

Definition 4: External Utility: EU(I_j) represents utility value of unit quantity of item I_j. For example, EU(q) = 2.

Definition 5: Utility of an item I_j: U(I_j, T_k) = IU(I_j, T_k) × EU(I_j) represents the measure of utility of item $I_j$in transaction T_k. For example, U(r, T3) = 2 × 4 = 8.

Definition 6: Utility of an itemset I in transaction T_k: U(I, T_k) = Σ[U(I_j, T_k)] such that I_j∈ I ˄ I ⊂T_k, i.e. sum of utilities of all the items of itemset I where I is a subset of transaction T.

For example, U({ps}, T1) = U{p, T1} + U(s, T1) = 1×1 + 1×2 = 3.

Definition 7: Utility of an itemset I in DB: U(I) = Σ [U(I, T_k)] such that T_k∈ S(I) where S(I) is the set of transactions containing itemset I, i.e. sum of utilities of itemset I in all the transactions where I has occurred.For example, U({pq}) = U({pq}, T1) + U({pq}, T3) = 5 + 4 = 9.

Definition 8: Utility of a transaction T_k: TU(T_k) = Σ[U(I_j, T_k)] such that I_j∈ T_k. For example, TU(T4) = 2+2+3 = 7. Figure2. shows TUs of all the transactions of our running example.

Definition 9: Utility of DB: It represents the sum of transaction utilities of all the transactions present in database.

For example, U(DB) = 11+19+18+7+24+25 = 104

Table3. TU of current DB example

TID	T1	T2	T3	T4	T5	T6
TU	11	19	18	7	24	25

Definition 10: Problem Definition: An itemset I is considered HUI when U(I) ≥ Min_util where Min_util is the user-specified utility threshold. The problem of HUI mining is to find complete set of HUIs.

For example, in our example, if Min_util = 20, then one of the HUI will be {pqr} with utility 25, whereas {pq} is a low-utility itemset with utility 9.

B. Related Work

It is obvious that utility don't follows monotonic or anti-monotonic property and hence there is a need for different pruning strategy in HUI mining than that of FIM. Prior to HUI mining, some algorithms as in [8-11] were available as a variant of this problem that considers EU = 1 for every item. For the first time, the concept of HUI mining was formally put forward in [12]. As a step towards finding some new pruning strategy for HUI mining, a new property named TWU was presented in [13].

Definition 11: Transaction Weighted Utility of an itemset I: TWU(I) = Σ[TU(T_k)] such that $T_k \in S(I)$ where S(I) is the set of transactions containing itemset I, i.e. the sum of TU of transactions in which itemset has occurred.

For example, TWU(p) = TU(T1) + TU(T2) + TU(T5) = 11+19+24 = 54.

Table4.TWU of 1-itemset of example DB

Item	p	q	r	s	t
TWU	54	60	97	104	93

This definition is used in HUIM to prune itemsets based on following properties:

- **Property 1:** TWU(I) ≥ U(I), called *Over-estimation.*
- **Property 2:**TWU(A) ≥ TWU(B) if A ⊂ B, called *TW Downward Closure property*.
- **Property 3:**TWU(A) ≥ Min_util then A is a HUI. If it is not then using Property 2 we can also say all its supersets will also be low-utility itemsets. This is called *Pruning property.*

These properties forms the ground for many famous HUI mining algorithms such as Two-Phase [14], IHUP [15], UP-Growth [16], UP-Growth+ [17], FP-Growth [18], etc. Among these, algorithms based on Two-Phase working suffers from the problem of level-wise repetitive DB scans for finding HUIs. The tree based approaches such as FP-Growth and UP-Growth outperform two-phase algorithms as these stores utility information of DB in prefix-trees and reduces the DB scans and candidate set as compared to two-phase approaches. HUI-Miner [7] proves to be the present best algorithm among HUI mining algorithms. It proposed a vertical format based structure to store the utility information of each itemset and performs a depth-first search to mine HUIs. The structure called Utility List (UL), performs join operations to compute the utility of patterns from their subsets.

Definition 12: Utility List (UL): For a given lexicographic ordering of items in an itemset, its UL is a set of tuples of the form (tid, iutil, rutil) for each transaction in which itemset has appeared in DB. The 'iutl' represents utility of itemset in Tid and 'rutil' represents the utilities of the items appearing after this itemset in the same transaction.For example, UL of {p} is as follows:

Table5. UL of {p}

tid	iutil	rutil
T1	1	10
T2	2	17
T5	2	22

HUI-Miner requires single scan to create UL of all 1-itemsets and then perform costly join operations to find larger itemsets. It uses following properties for pruning search domain:

- Sum of ituils: For a given itemset, if Σ(iutils) ≥ Min_util, it is a HUI.
- Sum of iutils and rutils: If Σ(iutils+rutils) ≥ Min_util, then only we need to check further it's extensions otherwise stop. This acts as a pruning property.

Although HUI-Miner is a proficient algorithm, the join operation proves to be costly in terms of execution time. The next section will present our approach to improve execution of HUI miner by reducing the number of joins needed to prune low-utility itemsets.

III. PROPOSED SOLUTION

The HUI-miner just uses ULs and join operations to search larger itemsets using these ULs without utilizing previous information for repeating itemsets. In this section, we present our solution, the TSM algorithm.

A. *Fast Hash MAP (FMAP):*To make the execution faster for the existing HUI-Miner, we tried to store the TWU information in Hash maps and later use this to reduce join counts while searching larger patterns. The overall working using this structure is given below:

1. In starting scan of DB, hasp map are created to store the TWU value of each item present in the DB.
2. For items having TWU ≥ Min_util, utility list will be created. Else item will not be considered for further mining process.
3. All items passing step 2 will be considered to generate another hash maps containing co-occurrence information. This is done by again scanning DB and computing newTWU values.

Let's take Min_util = 20 for our running example. As shown in Fig. 3, every item has TWU ≥ 20. So, hash maps will be created for these items as follows:

HM<p, 54>, HM<q, 60>, HM<r, 97>, HM<s, 104> and HM<t, 93>.

Figure1. Hash Maps created for 1-itemsets in first DB scan

In second DB scan, TWU values for every pair of 2-itemset will be computed, so that this information can be utilized further to prune larger itemsets quickly. So, after second scan, new hash maps will be formed, let's name them Fast hash maps. Fast hash maps will be created for every following pair:

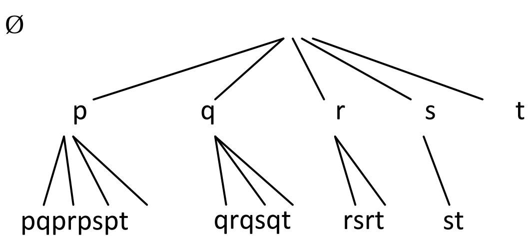

Figure2. Enumeration tree

FH<p, <q, 35>>, FH<p, <r, 54>>, FH<p, <s, 54>>,
FH<p, <t, 43>>
FH<q, <r, 53>>, FH<q, <s, 60>>, FH<q, <t, 49>>
FH<r, <s, 97>>, FH<r, <t, 86>>
FH<s, <t, 104>>

Figure3. Hash-maps storing TWU information of co-occurring suffix

B. TSM (TWU and Suffix based HUI Mining) Algorithm:

After completion of above mentioned steps, the algorithm recursively searches larger patterns using the ULs, hash maps and fast hash maps generated till now. It works in following manner:

1. Using 1-itemsets, ULs of 2-itemsets will be created by directly merging the ULs of relevant 1-itemsets. For example, UL{pq} will be created by merging UL{p} and UL{q}. The steps for creating new UL is similar as given in HUI-Miner, hence not explained here.
2. For creating higher order itemsets, the extensions of 2-itemsets are explored. The key idea here is to directly prune away those extensions which don't satisfy Min_util criteria according to fast hash maps. Along with this, pruning strategy of HUI_Miner is also used. For example, if we wish to find itemset {pqr} then we will need to utilize information about extensions of {p} i.e. {pq} and {pr}. So, leaving out the common prefix if the new itemset i.e. {qr} is found to be having TWU ≥ Min_util in Fast hash map, then only we will consider it for further mining and create its ULs. Thus, based on the TWU information of suffix attached to the common prefix, low utility itemsets are pruned.
3. After recursively checking all the extensions in this way, promising candidates i.e. HUIs will be mined in a faster way as compared to HUI-Miner because low utility itemsets are pruned in a better way without wasting time in creating their ULs.

The algorithm for searching promising extensions using hash maps is given below.

Explore Extensions Algorithm:

Input: current itemset (p), extensions of p (Exts_p), data structure storing Fast hash maps (FMAP), Min_util

Output: List of HUI

1. for each itemset X ∈Exts_p
2. if X.sumIUtils ≥ min_util
3. return X
4. if X.sumIUtils + X.RUtils ≥ min_util
5. for each itemset Y ∈Exts_p
6. if exists FMAP <x,<y, newTWU>> such that newTWU ≥ min_util {
7. pxy ← px∪py
8. pxy ← create(p,x,y)
9. Exts_X ← Exts_X∪pxy }
10. exploreExtensions(X, Exts_X, FMAP, min_util)

The Create function is used to create the ULs of larger itemsets in a similar way it is done in HUI-Miner [7]. So, the TSM for HUI mining can be presented as:

TSM Algorithm:

Input: Transaction database (DB), Min_util
Output: List of HUI

1. for each item I
2. scan DB to compute TWU and create HashMap<I,TWU>
3. if I.TWU ≥ min_util

4. do $I^+ \leftarrow$ I and create UL of I and HashMap<I,UL>
5. Sort I^+ in ascending order based on total ordering $\succ$
6. for each item I in I^+
7. scan DB to compute newTWU and create FMAP
8. exploreExtensions(I, I^+, FMAP, min_util)

IV. RESULTS

To evaluate the performance improvement of TSM algorithm as compared to HUI-Miner, we have done experiments with two synthetic datasets, namely Mushroom and Chess. The datasets can be obtained from [19]. The details of the two datasets are as follows:

Table6. Datasets used for experiments

Dataset	#Trans	#Items	Avg. Length	Size
Chess	3196	74	37	642Kb
Mushroom	8124	119	23	1064Kb

The TSM algorithm and HUI-Miner are compared for join counts, memory consumption and execution time. The observation can be summarized as follows:

1. **Join counts:** As a result of utilizing TWU information in an efficient way, a considerable reduction in the join counts while exploring search space is achieved.
2. **Execution time:** The TSM algorithm shows an improvement in the execution time as a result of reduction in the join counts. The comparison of above mentioned datasets is shown using in Fig. 7.
3. **Memory consumption:** Due to use of double hash-maps to store utility information of co-occurring items, the memory consumption of TSM algorithm is somewhat higher than the HUI-Miner.

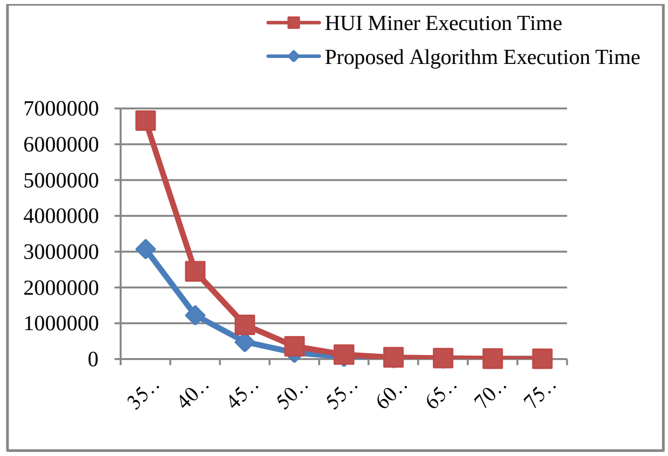

Figure4. Execution time comparison of Chess dataset[Horizontal axis: Min_Util, Vertical axis: Execution time(in ms)]

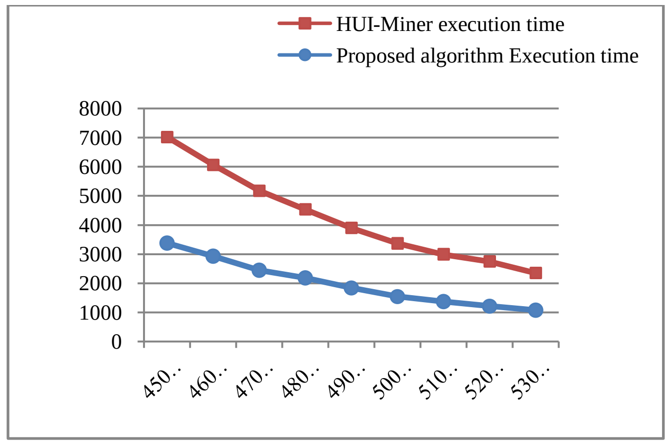

Figure5. Execution time comparison of Mushroom dataset[Horizontal axis: Min_Util, Vertical axis: Execution time(in ms)]

V. CONCLUSION

In this paper, the TSM algorithm has used double hash maps to efficiently store the TWU information that can be used in later stages of mining to efficiently prune the low-utility itemsets. As a result of introducing TWU based pruning for co-occurring, suffix items into HUI-Miner, the TSM algorithm shows join counts reduction and faster execution. Experimental results show that the TSM algorithm produces better results for dense datasets by consuming somewhat more memory.

REFERENCES

[1] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules. 20th VLDB Conference, 1994.

[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Database", Proc. ACM SIGMOD Intl. Conf, 1993, pp 207-216.

[3] Hong Yao, Howard J. Hamilton, and Cory J. Butz: A Foundational Approach to Mining Itemset Utilities from Databases. SDM (2004)

[4] Tseng, V. S., Shie, B.-E., Wu, C.-W., Yu., P. S., "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases", In: IEEE Trans. Knowl. Data Eng. 25(8), pp. 1772{1786 (2013)

[5] Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., Lee, Y.-K., " Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases", In IEEE Trans. Knowledge Data Eng. 21(12), pp. 1708{1721 (2009)

[6] Liu, Y., Liao, W., Choudhary, A., "A two-phase algorithm for fast discovery of high utility itemsets:, In: Proc. PAKDD 2005, pp. 689{695 (2005)

[7] Liu, M., Qu, J., "Mining High Utility Itemsets without Candidate Generation", In Proceedings of CIKM12, pp. 55{64 (2012)

[8] B. Barber and H. J. Hamilton, "Extracting share frequent itemsets with infrequent subsets", Data Mining and Knowledge Discovery, 7(2):153–185, 2003.

[9] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "A fast algorithm for mining share-frequent itemsets", In Proc. Asia-Pacific Web Conf., pages 417–428, 2005.

[10] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Efficient algorithms for mining share-frequent itemsets. In Proc. World Congress of Int'l. Fuzzy Systems Association, pages 534–539, 2005.

[11] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Direct candidates generation: A novel algorithm for discovering complete share-frequent itemsets", In Proc. Fuzzy Systems and Knowledge Discovery, pages 551–560, 2005.

[12] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases", In Proc. SIAM Int'l Conf. Data Mining, 2004

[13] Y. Liu, W.-K. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm", In Proc. Utility-Based Data Mining Workshop, pages 90–99, 2005.

[14] Liu, Y., Liao, W., Choudhary, A., "A two-phase algorithm for fast discovery of high utility itemsets", In: Proc. PAKDD 2005, pp. 689{695 (2005)

[15] Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., Lee, Y.-K., "Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases", In: IEEE Trans. Knowl. Data Eng. 21(12), pp. 1708{1721 (2009)

[16] Tseng, V. S., Shie, B.-E., Wu, C.-W., Yu., P. S., "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. In: IEEE Trans. Knowl. Data Eng. 25(8), pp. 1772{1786 (2013)

[17] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "Upgrowth: An efficient algorithm for high utility itemset mining", In Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, pages 253–262, 2010.

[18] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent pattern tree approach*", Data Mining and Knowledge Discovery, 8(1):53–87, 2004.

[19] Frequent Itemset Mining Dataset Repository. http://fimi.ua.ac.be/, 2012.