A Survey of High-utility Itemsets Mining

Haijun Yang*, Yonghua Lu and Bolan Zhang

Department of information and engineering, Lanzhou University of Finance and Economics, Lanzhou, China

*Corresponding author: yanghj@lzufe.edu.cn

Abstract. Data mining is of significance for finding useful information in massive data. Frequent itemsets mining (FIM) and high-utility itemsets mining (HUIM) are extremely common and wide application in research and real life. For one thing, HUIM algorithm focuses on utility, which is more practical. It can be used to find high profit goods, items with user’s preference, etc. For another, the difference between utility and frequency determines that HUIM and FIM algorithms are different. In order to introduce HUIM algorithms in the round, this paper showed typical HUIM algorithms for static data and stream data separately in section 2 and section 3. Meanwhile, section 2 partitioned algorithms based on candidates generation and threshold. Section 3 showed algorithms in terms of window model which is necessary to stream data mining. Lastly, this paper made a conclusion of referred HUIM algorithms and proposed some research prospects for this work.

1. Introduction
HUIM is developed on the basis of frequent itemset mining (FIM). In 2004, Yao et al.[1] firstly proposed the concept of high-utility itemsets and utility model. But they found the utility of itemsets doesn’t meet the downward closure property. As a result, Yao et al. didn’t propose a solution to HUIM. Utility model takes user’s preferences, item’s importance, profit and so on as utility. The utility generally contains two parts: internal utility and external utility. HUIM algorithm is generally improved by avoiding multiple database scans, reducing the candidates, improving pruning strategies, efficient data structures and so on. So far, efficient of HUIM algorithms has improved a lot. This paper was showed in 4 sections. Section 2 separately introduces HUIM algorithms for static data. Section 3 introduces HUIM algorithms for data stream. Section 4 draws a conclusion to introduce the challenges and prospective of HUIM.

2. HUIM Algorithms for Static Data
2.1. Two-phase Methods and One-phase Methods
According to the generation of candidates, there are two kinds of HUIM algorithms: two-phase method and one-phase method. The former will generate massive candidates, and in the second phase, it is necessary to verify them. For instance, Two-phase algorithm, IHUP, UP-Growth and so on. The latter directly calculates the actual utility of itemset. like HUIM, FHM, d²HUP, EFIM and so on. Since the utility of itemset does not meet the downward closure property, Two-phase algorithm[2] proposed transaction-weighted downward closure (TWDC) property in 2005 which realized HUIM firstly. Two-phase algorithm is level-wised. In the first phase, Two-phase algorithm selects candidates by TWDC property. Then in the second phase, it scans the database again to get the actual utility of each candidates. Its significant drawback is a huge amount of candidates.
To overcome the above drawback, Ahmed et.al.[3] proposed a tree structure named IHUP-tree, which store the name, support and TWU of item. When the database is scanned for the first time, the item is maintained in TWU descending order. Then, it is traversed from bottom to top. During the traversal, the candidates that is not less than the threshold is retained by iteration. For the second scan, the exact utility of all candidates are calculated, and high-utility itemsets can be found. IHUP uses a compressed tree structure to reduce memory consumption to a certain extent, but it also generates considerable candidates, and the iteration of sub-trees generation and tree node traversal will bring out greater time consumption. As a result, UP-Growth[4] introduced an improved UP-tree. UP-tree is more compressed because it only stores the item which meets the TWDC property. In order to reduce the candidates, UP-Growth uses four pruning strategies, DLU, DLN, DGU and DGN.

Numerous research demonstrate algorithms that apply tree structure usually generate a amount of candidates. Therefore, Liu et.al.[5] proposed a new algorithm HUI-Miner, applying a novel data structure, named utility list which store TID, iutil(utility of itemset) and rutil(remaining utility of itemset). HUI-Miner sorts items in TWU ascending order. For k-itemset, if the sum of its all iutil and rutil is not less than the threshold, it can be available to extended to (k+1)-itemset. Meanwhile, if the sum of iutil is not less than the threshold, it will be output as HUI. Through the iteration of above process, HUI-Miner can find all HUIs. Comparing with previous algorithms, HUI-Miner scans database twice and does not generate candidates, so the efficiency has improved further. In order to take advantage of the utility list, Fournier-Viger et.al.[6] applied a novel structure, named EUCS(Estimated Utility Co-occurrence Structure). EUCS maintains the TWU of all 2-itemsets. The algorithm does not construct utility lists for 2-itemsets whose TWU are less than the threshold. Krishnamoorthy et.al.[7] proposed an algorithm which uses revised sub-tree utility and local utility, stricter upper bound than remaining utility. With them, the search space can be further reduced. It also applies a novel array-based utility counting technique, named fast utility counting, to calculate these upper bounds. In addition, EFIM uses two techniques, high-utility database projection (HDP) and high-utility transaction merging (HTM), to reduce the complexity of space.

Though one phase algorithms are more efficient than two phase algorithms, utility list construction and junction have a limitation of time and memory consuming. ULB-Miner[8] was proposed using utility-list buffer structure which can store and retrieve utility list with time and memory reduction. Peng et.al[9] proposed a modified HUI-Miner, named mHUIMiner, which combined some thoughts from IHUP and HUI-Miner. This measure can avoid utility lists constructing of nonexistent itemsets in database. Experiment showed that mHUIMiner performed relatively well on sparse datasets.

Table 1. Comparison on traditional HUIM algorithms

| Algorithm   | Data structure | Theoretical basis | Features                                      |
|-------------|----------------|-------------------|------------------------------------------------|
| Two-phase   | -              | TWDC; width-first | A great amount of candidates; Multiple database scans |
| IHUP        | IHUP-Tree      | Pattern growth; depth-first | Many candidates; Twice database scanning |
| UP-Growth   | UP-Tree        | Pattern growth; 4 pruning strategies | Reducing the number of candidates; Twice database scanning |
| HUI-Miner   | Utility list   | Utility List; remaining utility | Without candidates; Twice database scanning |
| FHM         | Utility list   | EUCS              | Numerous calculation of itemsets; Performs better than HUI-Miner |
| EFIM        | Array          | sub-tree utility; local utility; HDP; HTM | 2-3 times faster, one-eighth space consumption |
| ULB-Miner   | Utility list   | EUCS; buffered utility lists | Time and memory consuming are both less than FHM. |
| mHUIMiner   | Utility list   | Tree structure; utility list | Outperforms in sparse datasets. |

2.2 HUIM Algorithms Based on Single and Multiple Threshold

In recent years, some scholars have realized that HUIM algorithms with a single threshold cannot meet
the practical demand. Therefore, multi-threshold HUIM algorithms are gradually proposed. In 2015, Lin et al. [10] introduced the problem of HUIM with multi-threshold and proposed two effective algorithms to solve this kind of problem, HUI-MMU and HUI-MMUTID. Their basic idea is to set a threshold for each item in the database. This paper introduces two new concepts and properties, LMU(Least Minimum Utility) and SDC(Sorted Downward Closure). HUI-MMU is a two-phase algorithm. To reduce time and space consumption of generating and verifying the candidates, an improved algorithm, HUI-MMUTID is proposed. The algorithm uses a TID search strategy, which uses a vertical structure to store the TID of the itemset, so that its utility of the itemset can be directly linked to its TID, reducing the number of database scans and improving the time efficiency of the algorithm. In 2016, the above authors added EUCS on the basis of HUI-MMUTID and proposed a new algorithm[11] which greatly improves time efficiency and space efficiency.

The three algorithms mentioned are two-phase algorithms. When the transaction length in the database is long and the threshold is small, these algorithms generate lots of candidates. Therefore, Gan et al. [12] proposed HIMU algorithm applying an efficient compressed tree structure, MIU-tree. Because TWDC is no longer applicable to this kind of problem, the paper introduced two improved attributes: GDC (Global Downward Closure) and CDC (Conditional Downward Closure).

The above algorithms sort the itemsets in a certain way, so user preferences will affect the results. Therefore, Krishnamoorthy et al. [13] proposed MHUI algorithm that does not impose a specific sorting method on the itemsets. The algorithm introduces four pruning measures, TWU-M, U-M, EUCS-M, LA-M. The results revealed that MHUI is more efficient than HIMU.

3. HUIM Algorithms for Data Stream
HUIM for data stream usually refers to four window models: landmark window, time decay window, sliding window, and inclined sliding window. The HUIM algorithms for data stream can also be divided into two-phase algorithm and one-phase algorithm, the former mainly includes THUI-Mine[14], HUI_W[15], wSWF[16], GUIDE[17], MHUI-BIT and MHU-TID[18], the latter mainly includes HUM-UT[19], vert-top-k[20] and HUISW[21].

THUI-Mine is the first algorithm to achieve HUIM on data streams applying sliding window model. A sliding window contains \(n\) batches of data, and each batch of data contains \(m\) transactions. When the first batch data arrives, the algorithm generates candidate 2-itemsets. When the second batch of data arrives, the algorithm will find all the candidate 2-itemsets in the first two batches until \(n\) batches of data are all processed. Finally, the database is scanned to mine all the HUIs. The algorithm saves the number, \(n_k\), of batches where the candidates firstly appears in the window in order to delete the oldest data. Meanwhile, the current TWU must be subtracted from its TWU in the \(n_k\) batch of data, and updated to \(n_{k-1}\). HUI_W uses an inclined sliding window, and divides the window into several parts. The user randomly sets the weight for each part to approximate the utility of \(k\)-itemset, and then generates the \((k+1)\)-itemset iteratively. w-SWF applies a similar method to THUI-Mine to generate candidate 2-itemsets, and then uses the Diffset strategy to calculate the utility of the HUIs. What is stored in the window is Binary vectors improves the computational efficiency of the algorithm. The framework of GUIDE is proposed, which has a compact MUI-trees and three methods based on different types of windows, GUIDELM, GUIDESW, GUIDETF. The idea of generating candidates are similar to Apriori algorithm. They respectively use BIT-vector and TID-lists structure to store the data in the sliding window. These two structures can retrieve the transaction where the itemset is located more quickly, thereby improving the time efficiency.

Feng et al. [20] proposed a UT-Tree structure in which items in the transaction are added in alphabetical order, and the leaf nodes store the utility of the items. HUM-UT calculates the utility of itemset according to the header table corresponding to the prefix tree of the global tree and the leaf nodes, and does not need to repeatedly scan the database. Vert-top-k follows the utility list structure. Through the utility and remaining utility of the itemset, the top-\(k\) high-utility itemsets are found without candidates generation and multiple database scanning. HUISW construct a structure called HUL-Tree. The tree nodes store not the TWU of itemset but the prefix utility values, which can make the minimum utility constraints more compact and reduce the search space.
Table 2. Summarization on HUIM algorithms for data stream

| Algorithm | Window                        | Data structure          | Theoretical basis                                                                 | Features                                      |
|-----------|-------------------------------|-------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------|
| THUI-Mine | Sliding window                | Utility list            | Partitioning the window                                                           | Runtime is two to three times faster.         |
| HUI_W     | Inclined sliding window       | Utility list            | Different weight for each partition.                                              |                                               |
| wSWF      | Sliding window                | Utility list            | Similar to THUI-mine for 2-item mining, and using Diffset strategy.               | The candidates are reduced by about 42%.     |
| GUIDE     | Landmark/sliding/inclined     | MUI-trees               | Similar to THUI-mine for 2-item mining, and using Diffset strategy.               | Compared with THUI-Mine, time has reduced more.|
| MHUI-BIT  | MHUI-TID sliding window       | Utility list/ LexTree-2HTU | Use Bit-vector, TIDlist structure and introduce LexTree-2HTU.                    | The consumption of time and memory is small. |
| HUM-UT    | sliding window                | UT-tree                 | Construct UT-tree, and construct utility list for each node.                      | Compared with HUPMS, there is no need to filter candidates. |
| Vert-top-k| sliding window                | Utility list            | A iList for each item, and the threshold for top-k itemsets increases with buffer. | Compared with T-HDUS, search space is reduced.|
| HUISW     | sliding window                | Utility list/ HUIL-tree | HUIL-tree is constructed which stores prefix utility of itemsets.                 | Compared with SHU-growth, the speed is twice faster in dense data streams. |

4. Conclusion
According to the above-mentioned HUIM algorithms, and considering the current challenges and problems which were presented in the research and practical application of data mining, this paper proposes several summaries as follow:
(1) Diversity of high-utility itemsets types. In addition to general high-utility itemsets, targeted algorithms for closed high-utility itemsets, maximum high-utility itemsets, top-k high-utility itemsets, complete high-utility itemsets, etc. should also be attached importance and improved further.
(2) Develop algorithms that specialize in processing data streams. As the most common data in today's society, it is meaningful to make research on data streams mining algorithms. In addition, since traditional HUIM algorithms cannot be directly applied to data streams, HUIM algorithms for data streams still face great challenges.
(3) The extension of HUIM applications in real life. The HUIM algorithms generally make experiments on public datasets, which can make reference to the selection of the minimum utility threshold. In order to meet the various demands of datasets in different fields, researchers should give more consideration to the diversity and differences in practical datasets, and propose more valuable HUIM algorithms.

Acknowledgments
Thanks to my supervisors, I got a lot of assistance and encouragement from them. And financial support from Natural Science Funding of Gansu Province(18JR3RA216) is significant.
References

[1] Yao Hong, Hamilton H. J. Mining itemset utilities from transaction databases [J]. Data & Knowledge Engineering, 2006, 59(3): 603-626.

[2] Liu Y, Liao W K, Choudhary A N. A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets[C]// Advances in Knowledge Discovery and Data Mining. Springer-Verlag, 2005:689-695.

[3] Ahmed C F, Tanbeer S K, Jeong B S, et al. Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases[J]. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(12):1708-1721.

[4] Lin C W, Hong T P, Lu W H. An effective tree structure for mining high utility itemsets. Expert Systems with Applications, 2011, 38(6):7419-7424.

[5] Liu M, Qu J. Mining high utility itemsets without candidate generation. In: Proceedings of the ACM International Conference on Information and Knowledge Management, Maui, HI, pp. 2012:55–64.

[6] Fournier-Viger P, Wu C W, Zida S, Tseng V S. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proceedings of the International symposium on methodologies for intelligent systems, Roskilde, Denmark. 2014: 83–92.

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

[8] Duong, Q.H., Fournier-Viger, P., Ramampiaro, H., Norvang, K. Dam, T.-L. (2017). Efficient High Utility Itemset Mining using Buffered Utility-Lists. Applied Intelligence, Springer

[9] Peng, A., Koh, Y. S., Riddle, P. mHUIMiner: A Fast High Utility Itemset Mining Algorithm for Sparse Datasets. PAKDD (2) 2017: 196-207.

[10] Lin C W, Gan W, Fournier-Viger P, Hong T P. Mining high utility itemsets with multiple minimum utility thresholds. In: Proceedings of the Eighth International Conference on Computer Science & Software Engineering. 2015: 9–17.

[11] Lin C W, Gan W S, Fournier-Viger P, et al. Efficient Mining of High-Utility Itemsets Using Multiple Minimum Utility Thresholds[J]. Knowledge-Based Systems, 2016, 113:100-115.

[12] Gan W S, Lin C W, et al. More Efficient Algorithms for Mining High-Utility Itemsets with Multiple Minimum Utility Thresholds [C]//Proc of 27th International Conference on Database and Expert Systems Applications. Switzerland: Springer International Publishing, 2016: 71-87.

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

[14] Chu C J, Tseng V S, Liang T. An efficient algorithm for mining temporal high utility itemsets from data streams[J]. Journal of Systems & Software, 2008, 81(7): 1105-1117.

[15] Tsai P S. Mining High Utility Itemsets in Data Streams Based on the Weighted Sliding Window Model[J]. International Journal of Data Mining & Knowledge Management Proc, 2014, 4(2): 13-28.

[16] Fouad M M, Mostafa M G M, Mashat A S, et al. wSWF: A Weighted Sliding Window Filtering Algorithm for Frequent Weighted Itemsets Mining [J]. International Journal of Innovative Computing Information & Control Ijcic, 2015, 11(4): 1421-1439.

[17] Shie B E, Yu P S, Tseng V S. Efficient algorithms for mining maximal high utility itemsets from data streams with different models [J]. Expert Systems with Applications, 2012, 39(17): 12947-12960.

[18] Li H F, Huang H Y, Lee S Y. Fast and memory efficient mining of high-utility itemsets from data streams: with and without negative item profits[J]. Knowledge&Information Systems, 2011, 28(3): 495-522.

[19] Feng Lin, Wang Le, Jin Bo. UT-Tree: Efficient mining of high utility itemsets from data streams[J]. Intelligent Data Analysis, 2013, 17(4): 585-602.

[20] Dawar S, Sharma V, Goyal V. Mining top-k high-utility itemsets from a data stream under sliding window model [J]. Applied Intelligence, 2017, 47(4): 1-16.

[21] Guo Shiming, Gao Hong. An efficient algorithm for mining high utility itemsets from data Streams based on sliding window techniques [J]. Journal of Harbin Engineering University,
2018, 39(4): 721-729.