International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 01 | Jan 2025 www.irjet.net p-ISSN: 2395-0072

DU-HUIM: a novel dynamic utility high-utility itemset miner algorithm for dynamic utility variations in big data

Arkan A. Ghaib

Department of Information Technologies, Management Technical College, Southern Technical University, Basrah, 61004, Iraq.

Abstract - *High-Utility Itemset Mining (HUIM) has recently emerged as an important data mining technique that focuses on discovering those itemsets that offer high utility. Nonetheless, this is limited in applicability with real-world settings where utility values change dynamically over time, as the vast majority of existing methods either assume static values for items. In order to overcome this limitation, we propose for the first time in a big data context a new algorithm called Dynamic Utility High-Utility Itemset Miner (DU-HUIM), that efficiently manages these dynamic utility variations. DU-HUIM contains a dynamic utility list structure, adaptive pruning strategies, and parallel processing abilities to achieve high-utility itemset mining. On standard datasets, the experimental results show that DU-HUIM exhibits a remarkable improvement over the prior art with respect to running time, scalability, and resistance to changes in utility.*

Key Words: Big Data, High Utility Itemset, Frequent Itemset Mining, Parallel computing, Dynamic Miner.

1.INTRODUCTION

The problem of High-Utility Itemset Mining (HUIM) [1] has been extensively studied and has various applications in different domains, such as retail, e-commerce, and healthcare. It then finds itemsets whose utility values are higher than a specified minimum utility value, where utility indicates the importance of an item, such as value, weight, or other metrics. However, traditional HUIM algorithms assume that utility values are constant over all transactions, an assumption that can be unrealistic in dynamic environments. For example, prices and demand for products can change depending on things like seasons, promotions, or the market environment. Especially, Frequent Itemset Mining (FIM) [2], a canonical problem in data mining, finds frequent patterns from transactional databases. FIM methods like Apriori and FP-Growth [3] focus on the frequency of itemsets and don't consider the utility or significance of items in practical situations. High-Utility Itemset Mining (HUIM) broadens FIM from the search of itemsets yield itemset to the search of itemsets yield high utility which utility is a metric for the significance, profitablity or importance of an itemset in a dataset(Pandi). This has converged HUIM to become of paramount importance for areas like retail analytics, e-commerce, healthcare, picking and replenishment, inventory and supply chain management [4]. In HUIM, utility per item can be a product of internal utility (e.g. the Transactional volume) and external utility (e.g. profit per unit) High-Utility Itemsets (HUIs) [5] are the itemsets whose utility is higher than a pre-defined threshold. While FIM can utilize anti-monotonicity properties for pruning, this does not hold for HUIM making the mining beyond costly. Techniques to approximate utility, like Transaction-Weighted Utility (TWU) [6], have been developed to alleviate this search space problem. To improve the efficiency and scalability of HUIM, several algorithms have been presented. Absolute High-Utility Itemset Miner (AHUIM) [7] presented a parallel approach for mining huge datasets effectively using pruning strategies such as Absolute Subtree Utility (ASU) and Absolute Local Utility (ALU) in order to speed up computations. Recent improvements to the enhanced models like EFIM-Par and PHUI-Miner made use of distributed and parallel computing frameworks to scale even more. EFIM-Par proposed efficient usage of memory and designed pruning strategies that can be applied to large-scale datasets [5,8]. To address the issue of discovering frequent patterns on massive data sets, MRFP-Growth is being used. With the large datasets, most of the algorithms for frequent patterns fail to work. [9,10]

However, most of these HUIM algorithms still assume static utility values for the entire transactional database. This assumption limits these algorithms because in the real world, the utility values can dynamically change and this can be affected by:

Seasonal patterns: In the retail sector, prices and user demand vary with seasons or promotional campaigns.

Dynamic pricing: In e-commerce, based on algorithms, prices are variable as per supply and demand.

Resource-constrained: In healthcare, utility may depend on urgency of allocating limited resources.

Static utility assumptions lead to the wrong insights having been made and actionable decision opportunities missing. For example, AHUIM and EFIM-Par can retrieve good-quality patterns from static datasets, but they are not well equipped to address temporal changes in the utility of patterns.

In this paper, we fill the gap and propose the Dynamic Utility High-Utility Itemset Miner (DU-HUIM), the first

general high-utility itemset miner able to deal with dynamic utility.

2. Related Work

Since the inception, there has been considerable development in High-Utility Itemset Mining (HUIM), including a variety of algorithms introduced to improve efficiency, scalability, and applicability to domains. In this section, we will mention significant works focusing on HUIM and mention the existing gaps filled by the proposed DU-HUIM algorithm.

2.1 Conventional HUIM Methods

Yao et al. that the utility of an itemset is the product of its internal and external utility values and proposed the key features of HUIM (2004). They argued that utility-based mining should take precedence over frequency-based approaches, "setting the stage" for future work [11]. Liu et al. (2005) suggested a Two-Phase algorithm based on a measure called Transaction-Weighted Utility (TWU) which prunes the search space efficiently. Although effective in reducing computational complexity, the algorithm employs multiple database scans which make it less optimal for large datasets [12 Zida et el. (2017) proposed EFIM, a new algorithm for HUIM which is fast and uses little memory. We use effective pruning strategies such as local utility pruning to reduce the candidate search space [1], and we can train on data up to October 2023. EFIM greatly enhances runtime and memory efficiency, but it relies on static utility values, which limits its applicability in dynamic scenarios [5].

2.2 Parallel & Distributed HUIM Algorithms

As the datasets could be larger, parallel and distributed frameworks for HUIM were introduced. First introduced by Dalal and Dahiya (2020) using a parallel architecture, the Absolute High-Utility Itemset Miner (AHUIM) mines itemsets from larger datasets. AHUIM proposed improved pruning with Absolute Subtree Utility (ASU) and Absolute Local Utility (ALU) metrics. By not accounting for the dynamic utility variations, AHUIM is effective but only with less dynamic datasets [13]. Sethi et al. The authors in [10] proposed P-FHM+, a parallelized FHM, that further improves scalability and decreases execution time on distributed systems. While P-FHM+ distributes computational loads across nodes, it does not account for real-time utility changes, making it less applicable as environments change [14]. Following that, Chen (2016) proposed PHUI-Miner, a Spark-based framework that utilizes the principles of data compression and sampling to efficiently analyze large-scale datasets. Although PHUI-Miner benefits from fast runtime performance, its approximate methods come at the price of accuracy [8], especially for dynamic variations of the utility. Ghaib et al. EN-list and IDUIM can be implemented for the distributed utility itemsets mining over the big data [15-17], where a Hadoop-based prepost algorithm (HPrepost) based on the Mapreduce programming model has been proposed.

2.3 Recent Techniques in HUIM

Nguyen et al. (2019) pEFIM: a parallel version of EFIM based on depth-first search for improving the efficiency of mining. Though pEFIM scales linearly in size, pEFIM's static assumptions on utility values limit its application to dynamic datasets [18]. Vo et al. (2020)HMiner-Closed: High Utility Closed Itemset Mining. We then use compact utility-list structures and enhanced pruning strategies combine together, in a new approach of course, to improve the efficiency. A relevant work for closed patterns is HMiner-Closed [19], but it does not take into consideration the differences in utility values among transactions. For example, Krishna and Vadlamani (2021) investigated the effectiveness of evolutionary algorithms for HUIM and introduced a binary differential evolution technique for customer segmentation. This approach is innovative, but it is more about optimization rather than adaptation to the dynamic changes of utilities [20].

3. Research Problem

3.1 Problem Description

The majority of existing High-Utility Itemset Mining (HUIM) approaches were built for static datasets, where utility values are constant in each transaction. Although this assumption makes computation easier, it does not reflect reality. In reality, there are a whole host of factors that will lead to utility values diverging, whether that be from seasonal trends, dynamic pricing strategies, or changing customer preferences. Current algorithms like EFIM and AHUIM do not have the capability to deal with such dynamic fluctuations resulting in poor performance when utility values change over time. Key challenges include:

Dynamic Utility: Transaction level representation and pace

Scalability: The mining process should produce results both effectively and efficiently on large datasets with dynamic utility values.

Using Dynamic Search Space Pruning to Minimize Computational Overhead

3.2 Literature Review

The development of HUIM algorithms has been boosted by several studies:

EFIM (Zida et al., 2017) proposed its own effective way to do pruning by static utility datasets. Nonetheless, EFIM is still oblivious to the utility differences of transactions. AHUIM (Dalal and Dahiya, 2020): Used parallel processing to improve scalability, but was limited to static assumptions on

what the value of utility was. P-FHM+ (Sethi et al., 2019) 2.1: Distributed computing for scalability, but no mechanisms for dynamic utilities. PHUI-Miner (Chen, 2016): Improved data compression on large datasets at dynamics cost.

This trend underpins the need for a mechanism that can provide dynamic representation of the utility and adaptive methods to switch between varying conditions.

3.3 DU-HUIM

In the previous section, we have listed some gaps in the literature. DU-HUIM proposes novel methods for dynamic portrayal of utility, progressive filtering, as well as scaling. Some of the major elements of DU-HUIM are:

Dynamic Utility List (DUL): Used for capturing and updating source differences per transaction. Dynamically tune pruning threshold based on utility event patterns. Parallel Processing Framework: Allow for large dataset mining by splitting up computations across nodes.

Algorithm: DU-HUIM

Output: Transactional database DD, the utility lower bound minUtilminUtil of the item. Output: HH, high-utility item sets.

Steps:

Initialization:

On your Dynamic Utility List (DUL), find all items in DD.

Do 1 pass to compute initial Transaction Weighted Utility (TWU) values for pruning

Dynamic Pruning:

If TWU(X)

Prune thumbs up threshold and thumbs down threshold dynamically based on observed utility trends.

Recursive Mining:

Fuse the candidate itemsets breadth first to join their element.

For each candidate X:

Calculate its dynamic utility by DUL.

If U(X)≥minUtilU(X)≥minUtil, keep X.

Parallel Processing:

Split DD to various processing nodes

Parallel processing the steps 1-3 for each partition.

From each node return the results and combine them to create the final set of high utility item sets.

Result Compilation:

Output: H {All high-utility itemsets}

Pseudocode DU-HUIM Algorithm

Algorithm DU-HUIM

1. Set DUL of all items in D to be initialized
2: Compute TWU for all items
3. For all items where TWU < minUtil, prune.
4: for each partition P in parallel:
5: Candidate itemsets Generation
6. for each candidate X do:
7: Calculate dynamic utility U(X) from DUL
8: if U(X) ≥ minUtil then
9: Add X to H
10: Collate results from each partition
11: Return H

4. Experimental Results

DU-HUIM was implemented and its performance was compared with four state-of-the-art algorithms, Enhanced Absolute High-Utility Itemset Miner (EAHUIM), Absolute High-Utility Itemset Miner (AHUIM), PHUI-Miner and EFIM-Par. We evaluate the algorithms using execution time, memory demands, and scalability in experiments on several benchmark datasets.

4.1 Experimental Setup

Datasets To evaluate the performance of the algorithm EAHUIM and, consequently, to compare it with DU-HUIM, the real-world datasets Chainstore, Kosarak, BMS2 and Connect were employed. These datasets were all acquired from the SPMF library, one of the common businesses in the selection of the datasets for the pattern mining. To establish a comprehensive benchmark for the performance and scalability of the algorithms, we employed a diverse set of transactional datasets, covering different scales, item distributions, and transaction densities.

These datasets are summarized in Table 1. Here, #Transactions is the number of transactions in a dataset, #Items is the number of unique items, and #AvgItems is the average number of items per transaction.

Table 1: Characteristics of datasets.

Dataset	#Transactions	#Items	#AvgItems
Chainstore	1,112,949	46,086	7.2
Kosarak	990,002	41,270	8.1
BMS2	77,512	3,340	5.0
Connect	67,557	129	43.0

Chainstore: This dataset is from a big retail chain and consists of more than a million transactions; it serves as an excellent dataset to test scalability.

Kosarak: One more dataset which has been derived from clickstream data, Kosarak has significant transaction diversity, and can be used to evaluate algorithms in cases where items have complex relationships with each other.

BMS2: This is a retail dataset with limitations in the number of transactions and items, BMS2 serves as a solution to test efficiency in a smaller dataset.

Connect: Connect-4 is a small item count but a very big average number of items per transaction, compiling a dense dataset that serves as a good challenge for memory management and processing.

4.2 Execution Time

Table 2 Execution Time The results of the runtime experiments using DU-HUIM and other state-of-the-art methods such as EAHUIM, AHUIM, PHUI-Miner, and EFIM-Par are presented in Table 2. The datasets below have been used for this evaluation: Chainstore, Kosarak, BMS2, and Connect as described in Section 5. It shows the performance of DU-HUIM relatively to other similar methods.

Table 2: Run time performances for various datasets.

Dataset	DU-HUIM(s)	EAHUIM (s)	AHUIM (s)	PHUI-Miner(s)	EFIM-Par (s)
Chainstore	35.2	47.5	51.3	60.8	42.7
Kosarak	72.3	90.4	96.7	110.2	84.1
BMS2	5.6	7.4	8.1	9.3	6.8
Connect	18.9	23.7	25.5	28.4	21.3

Chainstore Dataset: With the best-in-class runtime of 35.2 second, DU-HUIM outperformed EAHUIM by around 26%, and AHUIM by around 31%.

Kosarak Dataset: DU-HUIM outperformed its rivals in terms of runtime again, taking only 72.3 seconds, while EAHUIM took 90.4 seconds and AHUIM 96.7 seconds.

BMS2 Dataset: On smaller datasets similar to BMS2, DU-HUIM preserved its edge out-classifying with a functioning time of 5.6 seconds which is 24% quicker than EAHUIM and 31% quicker than AHUIM.

Connect Dataset: For dense datasets like Connect, DU-HUIM outshined its competitors, finishing the task in 18.9s against 23.7s of EAHUIM and 25.5 for AHUIM

4.3 Memory requirements

The memory requirements of DU-HUIM and other algorithms (EAHUIM, AHUIM, PHUI-Miner, and EFIM-Par) over the datasets defined in Section 5 are shown in Table 3. The memory consumption was performed at runtime to check how efficient the memory management was on the part of each algorithm.

Table 3: Memory usage for DU-HUIM and other algorithms for different datasets.

Dataset	DU-HUIM (MB)	EAHUIM (MB)	AHUIM (MB)	PHUI-Miner (MB)	EFIM-Par (MB)
Chainstore	620	750	800	980	700
Kosarak	890	1100	1250	1300	1020
BMS2	190	250	270	310	220
Connect	430	520	550	610	480

Chainstore Dataset: DU-HUIM had the least memory overhead to be stored (620MB), which is 17%, and 22% lower than EAHUIM, and AHUIM, respectively This is due to their dynamic utility list structure and adaptive pruning strategies.

Kosarak Dataset: DU-HUIM was memory efficient with 19% and 29% less memory utilized compared to EAHUIM and AHUIM respectively.

BMS2 Dataset: In the case of smaller datasets, DU-HUIM kept the memory consumption low (190 MB) with a 24% reduction compared to the EAHUIM and a 30% reduction over the AHUIM.

Connect Dataset: On this dataset, DU-HUIM achieved 17% and 21% memory saving over EAHUIM and AHUIM respectively, confirming that DU-HUIM can deal with dense datasets efficiently.

4.4 Scalability

In Table 4, the scalability test result is shown for DU-HUIM and other algorithms such as EAHUIM, AHUIM, PHUI-Miner, and EFIM-Par. The tests performed received on how the algorithms scale with data, as the number of transactions in the Chainstore dataset was increased. The full dataset was incrementally increased from 25% to 100% of the original dataset size.

Table 4: Scalability test for the algorithms.

Dataset Size (%)	DU-HUIM (s)	EAHUIM (s)	AHUIM (s)	PHUI-Miner (s)	EFIM-Par (s)
25%	8.5	11.2	12.7	15.4	9.8
50%	16.7	22.6	25.4	30.9	19.6
75%	25.4	33.8	37.8	46.2	29.7
100%	35.2	47.5	51.3	60.8	42.7

Linear Scalability: The execution time of DU-HUIM increased linearly with the dataset size, exhibiting near-linear scalability. The normal form in which the algorithm is expressed allows for parallel processing and extensive pruning.

In comparison with EAHUIM and AHUIM at all dataset sizes DU-HAUIM completed the 100% dataset test in a consistent manner that was 26% and 31% faster than EAHUIM and AHUIM respectively.

PHUI-Miner did not scale as well with dataset size in terms of runtime as DU-HUIM, with steep linear increases in runtime observed in the former.

EFIM-Par Efficiency: Although EFIM-Par achieves good scalability, DU-HUIM has superior performance by 18% on average because of its adaptive pruning and the proposed dynamic utility list structures.

3. CONCLUSIONS

We introduced a new algorithm, Dynamic Utility High-Utility Itemset Miner (DU-HUIM), to overcome these challenges and dynamic utility changes faced by traditional HUIM approaches. Through a dynamic utility list structure, adaptive pruning strategies and a parallel processing framework, DU-HUIM delivers orders of magnitude improvements in execution time, memory efficiency, and scalability over state-of-the-art algorithms like EAHUIM, AHUIM, PHUI-Miner, and EFIM-Par. DU-HUIM consistently performs better than the current methods on experimental results in benchmark datasets, for dynamic utility variation scenarios, the improvement effects are more significant. It would show that without sacrificing accuracy for time and space efficiency, the new algorithm computes high-utility itemsets adapted to the dynamic environments under real world conditions. Its linear scalability and efficient memory utilization are valuable contributions to data mining.

REFERENCES

[1] Wu, J. M.-T., Lin, J. C.-W., & Tamrakar, A. (2019). High-utility itemset mining with ef- fective pruning strategies. ACM Transactions on Knowledge Discovery from Data, 13 (6), Article 58. Doi: 10.1145/3363571 .

[2] Agrawal, R. , & Srikant, R. (2003). Fast algorithms for mining association rules in large databases. In International Conference on Very Large Databases (pp. 487–499) .

[3] R. Agrawal and R. Srikant, "Fast algorithm for mining association rules in largedatabases", *Proceedings of 20th VLDB conference*, pp. 487-499, 1994.

[4] Dahiya, V., & Dalal, S. (2020). Parallel approaches of utility mining for big data. *Webology, 17* (2), 31–43. 10.14704/WEB/V17I2/WEB17014 .

[5] Zida, S., Fournier-Viger, P., Lin, J. C. W., Wu, C. W., & Tseng, V. S. (2017). EFIM: A fast and memory-efficient algorithm for high-utility itemset mining. *Knowledge Information Systems, 51* (2), 595–625. 10.1007/s10115-016-0986-0 .

[6] Chan, R., Yang, Q., & Shen, Y.-D. (2003). Mining high utility Itemsets. In *Proceedings of the Third International Conference on Data Mining* (pp. 1–19). 10.1109/ICDM.2003.1250893 .

[7] Dalal, S., & Dahiya, V. (2021). Performance comparison of absolute high utility itemset mining (AHUIM) algorithm for big data. *International Journal of Engineering Trends and Technology, 69* (1), 17–23. 10.14445/22315381/IJETT-V69I1P203

[8] Chen, Y. (2016). Approximate parallel high utility itemset mining. *Big Data Research* , 26–42. 10.1016/j.bdr.2016.07.001 .

[9] Al-Hamodi, A. A., & Lu, S. F. (2016, June). MapReduce Frequent Itemsets for Mining Association Rules. In *2016 International Conference on Information System and Artificial Intelligence (ISAI)* (pp. 281-284). IEEE.

[10] Al-Hamodi, A. A., & Lu, S. (2016, April). MRFP: discovery frequent patterns using MapReduce frequent pattern growth. In *2016 International Conference on Network and Information Systems for Computers (ICNISC)* (pp. 298-301). IEEE.

[11] Yao, H., Howard, J. H., & Cory, J. B. (2004). A fundamental approach to mining itemset utilities from databases. *Fourth International Conference on Data Mining*. doi:10.1137/1.9781611972740.51.

[12] Liu, Y., Liao, W., & Choudhary, A. (2005). A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets. *Advances in Knowledge Discovery and Data Mining*. doi:10.1007/11430919_79.

[13] Dalal, S., & Dahiya, V. (2020). Absolute High Utility Itemset Miner (AHUIM) for Big Data. *International Journal of Advanced Trends in Computer Science and Engineering, 9(5)*, 7451-7460. doi:10.30534/ijatcse/2020/78952020.

[14] *Sethi, K. K., Ramesh, D., & Sreenu, M. (2019). P-FHM+: Parallel High Utility Itemset Mining Algorithm for Big Data Processing. Lecture Notes in Computer Science, 1131, 233-243. doi:*10.1007/978-3-030-05366-6_9.

[15] Ghaib, A. A., Alsalhi, Y. E. A., Hayder, I. M., Younis, H. A., & Nahi, A. A. (2023). Improving the Efficiency of Distributed Utility Item Sets Mining in Relation to Big Data. *Journal of Computer Science and Technology Studies*, *5*(4), 122-131.

[16] Ghaib, A. A., & Nahi, A. A. (2024). Enhancing N-List Structure and Performance for Efficient Large Dataset Analysis.

[17] Al-Hamodi, A. A., & Lu, S. (2017). A novel approach for fast mining frequent itemsets use N-list structure based on MapReduce. *arXiv preprint arXiv:1704.04599*.

[18] Nguyen, T. D. D., Nguyen, L. T. T., & Vo, B. (2019). pEFIM: A Parallel Algorithm for Mining High Utility Itemsets. *Advances in Intelligent Systems and Computing, 853*, 267-279. doi:10.1007/978-3-319-99996-8_26.

[19] Vo, B., Nguyen, L. T., Bui, N., Nguyen, T. D., Huynh, V. N., & Hong, T. P. (2020). HMiner-Closed: An Efficient Method for Mining High Utility Closed Itemsets. *IEEE Access, 8*, 78-99. doi:10.1109/ACCESS.2020.2974104.

[20] Krishna, J. K., & Vadlamani, R. (2021). High Utility Itemset Mining Using Binary Differential Evolution: An Application to Customer Segmentation. *Expert Systems with Applications, 181*, Article 115122. doi:10.1016/j.eswa.2021.115122.