International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

An Efficient High Utility Itemset Mining Approach using Predicted Utility Co-exist Pruning

Suresh B Patel[1], Sanjay M Shah[2], Mahendra N Patel[3]

Submitted: 10/09/2022 **Accepted:** 20/12/2022

Abstract: The traditional frequent item-set mining is most popular and widely used technique for mining of related items. It considers whether the item is present or absence in dataset. However, item quantity and its importance is need to be consider for some real-world problem such as identify profitable items from the customer transaction dataset in supermarket, discover valuable customer for business, in medical field identify the combination of symptoms that are more significant to diseases. High utility itemset mining considers item quantity and its importance. Many researches have been done on the high utility itemset mining. Among them, utility list-based methods are efficient as it does not generate the candidate set. However, drawback of such techniques is lot of expensive join operations on utility list which degrades the performance of algorithm by increasing the storage requirement and time for execution. We proposed Predicted Utility Co-Exist Structure known as PUCS to store the utility data and Predicted Utility Co-Exist Pruning known as PUCP to eliminate unnecessary utility list join operations. It improves the algorithm's performance. We experiment the proposed approach on standard real-life datasets and results are compared with existing methods. According to experimental result analysis, proposed PUCP-miner outperforms existing approaches concerning execution time and memory requirement. In terms of execution time, proposed approach achieves more than 20 % improvement and for memory consideration, proposed approach got 3% improvement compared to state of the art approaches.

Keywords: *Data Mining, Frequent Itemset, High Utility itemset, Utility List, Transaction Weighted Utility.*

1. Introduction

Data mining techniques such as mining of frequent itemset [2], mining of high utility itemset [3][4][5], are used to extract important patterns hidden inside the massive data [1]. Frequent item-set mining (FIM) is a demanding approach in real-life applications such as analyzing customer behavior, analyzing the symptoms contribution to disease, identifying valuable customers, etc. The FIM only consider the presence or absence of items within the transactions which is the major limitation of FIM [2][6]. Item quantities and its importance play a vital role in a variety of applications significantly for transaction databases [3][7][8][9]. High utility item-set mining (HUIM) highlights itemset that yields high profit/importance[3][5][10][11][12]. It considers the item's count and profit of the items. HUIM extracts the set of items whose utility is not lesser than the user-specific min utility threshold. The HUIM problem is substantially more difficult than FIM as the utility measures do not adhere to downward closer characteristic that is utilized to effectively trim the search field [5][8][10][12][13][14]. HUIM produces a huge number of candidate sets, which takes up high storage and time, in order to determine high utility itemsets[3][5] [9][15].Utilizing the variety of upper bound, existing methods trim the search space to relieve the expensive computational task. Still, some low utility candidate sets are remain as these upper bound are overestimation[5][12][16]. Recently, researchers proposed utility lists to locate high utility itemsets without creating candidate sets[10][13][14][17][18]. Utility list-based approaches for HUIM perform expensive join operations on utility list. The cost is measured in time for execution and storage demand. This study addresses these problems and contributes as below.

- Unique structure namely PUCS (Predicted Utility Co-exist Structure) which records information of utility list for co-exist items in dataset.
- A novel pruning technique, PUCP (Predicted utility co-exist pruning), is proposed that decreases join operations of utility list and improves the performance mining algorithms.
- Extensive experiments were performed on real datasets, and results are compared with various existing methods. Experimental results demonstrated that PUCP-Miner performed better in the matter of storage requirements and time for execution.

1.1. Problem Background

Table 1 & 2 shows $\{T_1, T_2 \ldots\ldots T_n\}$as set of transaction contains set of items $I = \{i_1, i_2, \ldots.., i_m\}$ with quantity $(q(i_1), q(i_2), \ldots..q(i_m))$ form a transaction database. T_{id} a transaction identifier shows

[1] *Gujarat Technological University, Ahmedabad-382424, India*
ORCID ID: 0000-0003-2861-8972
[2] *Government Engineering College, Rajkot- 360005, India*
ORCID ID: 0000-0002-4937-4527
[3] *Government Engineering College, Gandhinagar - 382028, India*
ORCID ID : 0000-0002-4342-1070
** Corresponding Author Email: patel_sureshkumar2@gtu.edu.in*

individual transaction in the database. The utility table shows the external utility of each item. The external utility is treated as an item's weight, importance, profit, etc.

Table 1. Transaction Database DB

T_{id}	*Transactions*
T_1	(b#1),(c#2),(e#1)
T_2	(f#3),(g#1)
T_3	(a#3),(b#4),(e#2),(g#1)
T_4	(a#1),(b#1),(d#1)
T_5	(a#1),(b#2),(c#3),(d#4),(e#5)

Table 2. External Utility

Item	a	b	c	d	e	f	g
External Utility	5	1	3	4	2	1	2

Definition 1: Item's Utility.

For an item i_k and T_j the utility value of i_k is u (i_k,T_j) = q(i_k,T_j)*p(i_k).Consider item a's utility in transaction T_3 is u(a, T_3)=p (a) *q (a, T_3) = 15.

Definition 2: Itemset 's Utility.

For the itemset P_x and transaction Tithe utility value of P_x is

$$u(Px, Tj) = \sum_{ik \in x \wedge ik \in Ti} u(ik, Ti) \quad (1)$$

The itemset P_x ={a,b}. P_x's utility is u(P_x,T_4)=u(b,T_4) + u (a,T_4)=7.

Definition 3: itemset's utility in database DB.

For the itemset P_x and database DB the P_x's utility in DB is

$$u(Px) = \sum_{ik \subset I \wedge x \in Ti} u(x, Ti) \quad (2)$$

Consider itemset Px = {a b} is a part of transactions T3,T4 and T5. The itemset Px's utility in database is u(Px) =u(Px, T3) +u(Px, T4)+u(Px, T5) = 19+6+7=32.

Definition 4: High Utility itemset.

The itemset P_x is high utility itemset, if it's utility value is higher than the user defined MinUtility threshold specified by user.

$$HUIset = \{P_x \mid P_x \subseteq I, u(P_x) \geq MinUtility\} \quad (3)$$

For the item set P_x = {a b} and MinUtility threshold is 30. The itemset P_x 's utility is 32 so P_x is consider as high utility.

Definition 5: HUIM Problem.

From the transaction database, finding all itemsets whose utility is larger than the user-specified MinUtility criterion.

Definition 6: Transaction's Utility

For the given transaction, the total of all the item's utility is called transaction's utility and is defined as

$$u(Ti) = \sum_{\forall ik \in Ti} u(ik, Ti) \quad (4)$$

Consider transaction T_2, u (T_2) =u (f, T_2)+u (g, T_2)=5.

Definition 7: Transaction Weighted Utility (TWU)

For the itemset, the TWU of itemset Px is the total of all transaction's utility in which itemset Px exist.

$$TWU(Px) = \sum_{x \in Ti} u(Ti). \quad (5)$$

Consider itemset Px = {a,b}, the TWU(Px) is the total of transaction's utility of T3,T4 and T5, so TWU(Px) is 77.

2. Related Work

Mining of Frequent Itemset is a problem of HUIM. Unlike FIM, utility measures in HUIM is not applicable to minimize space of searching as utility list can be equal, higher or lesser than its superset as well as subset [2][3][5][10][11][13][17][18][19]. Discovering the HUI is a costly task with regards to memory requirements and execution time[2][3][5][10][11][13][17][18] [19]. Most of the previous research focuses on the trim the search space by proposing various pruning measures [5][13][18] and the way to record the information of utility[10][12][17][18][19]. Recently the utility list based algorithms ware proposed that are cost effective[10][13][4][17][18]. In 2012, Liu and Qu introduced the first single-phase HUIM method, HUI-Miner, which does not needs candidate generation. The innovative list structure called utility list applied by HUI-Miner is where the utility data for the itemset is stored[10]. Although HUI-Miner is quicker than older techniques, it performs time-consuming utility list join operations[10]. To decrease the join count, Viger, Wu, Zida, and Tseng suggested FHM utilizing EUCP in 2014.Based on the item co-occurrence a novel pruning strategy called EUCP presented for reduction of the utility list join operations[18]. To prevent producing utility lists of item sets that don't exist in the database, Peng, Koh, and Riddle presented a tree structure called IHUP in 2017. They proposed an efficient algorithm incorporating the IHUP tree called mHUIMiner[13].The ULBMiner was proposed by Duong, Viger, ramampiaro, norvag, and dam in 2017. Authors introducing the itemset's memory reuse won't be further extended[17]. Qu, Liu, and Viger suggested a novel structure for utility list called Utility-List* in 2019[10]. Based on the observation, the output of ULB-Miner, FHM, HUI-Miner, and HUIMiner degraded due to ineffective Tids comparisons for joining the utility lists. The HUI-Miner* removes these ineffective comparisons by Utility-list* structure[20]. From the literature survey, it has been observed that researchers proposed various pruning measures to minimize the space for searching and data structures to reduce cost with regards to memory demand and execution time. The main limitation of the algorithms based on utility list is costly utility list join operations[10][13][14][17][18]. Even though numerous algorithms are proposed, there is a scope to decrease the cost of the join operations on utility list by reducing the number of comparisons, join count, etc. In this work, we Proposed novel pruning techniques PUCP uses PUCS to minimize the utility list join operations.

3. Proposed Method

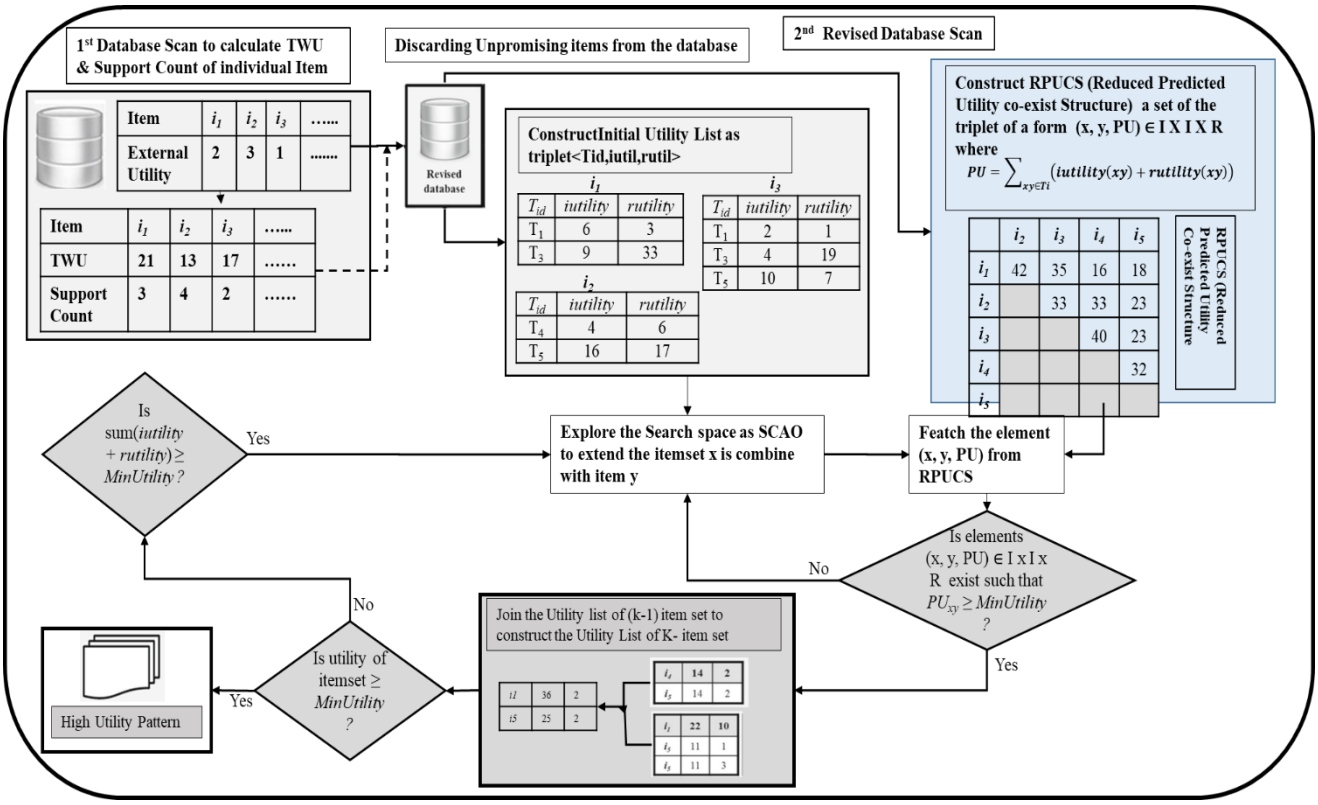


Fig 1. Flow of Proposed work

The utility list base approaches find the high utility itemset without generating a candidate set. Utility list-based HUIM methods initially read the dataset to determine support count and TWU for each individual item. Discard the unpromising items. Unpromising item can be considered as the item with less TWU value than MinUtility threshold. Consider the database in tables 1 & 2, the items f and g are unpromising, so they are discarded. Rearrange the transaction as per support count ascending order. It will lead to reducing the cost of mining procedure by decreasing count of comparisons needed to join utility list. The SCAO is c-d-e-a-b. The revised database is in table 3. Utility list-based HUIM algorithms apply the list structure to record the itemsets's utility. Utility list is a triplet <Tid,iutility,rutility>. Here Tid represents unique transaction id in which the itemset exists. iutility is a value of itemset's utility in a transaction. rutility represents sum of value of item's utility that come after itemset in a transaction to be considered.

Table 3. Revised Database

T_{id}	*Transactions*
T_1	(c#2),(e#1),(b#1)
T_3	(e#2),(a#3),(b#4)
T_4	(d#1), (a#1), (b#1)
T_5	(c#3),(d#4),(e#5), (a#1),(b#2)

Definition 8: All the items in Transaction T that come after itemset x where x⊆ T is denoted as T|x. Consider revised database as in table 3 T5 | de = {ab} and T3|a = {b}

Definition 9: Remaining utility.

The sum of all item's utility that follows itemset x in transaction Ti, is represented as rutility(x, Ti).

rutility(x ,T) = $\sum_{i\in(T|X)} u(i, T)$ where X⊆T

Consider itemset x = {de}, the rutility(x, T5) is u (ab, T5) =7.

Previous algorithms like HUI-Miner, mHUI-Miner, and ULB-Miner explore the search space that can represent as a set-enumeration tree. Then recursively extend the itemset by joining the utility list of a smaller itemset and then pruning the search space by using the following properties.

Property 1: If the sum of iutility and rutility value of the utility list of itemset x is less than minUtility threshold, then any extension of itemset x by appending item y comes after x as per order is not a high utility itemset[14].

These algorithms' performance degrades due to numerous costly utility list join operations. In this work, the proposed PUCS (Predicted utility co-exist structure) and PUCP (Predicted utility Co-exist pruning) minimize the number of utility list join operations. It eliminates the low utility itemset directly without performing the join operations. In the following part, we introduce our novel structure and pruning method.

3.1. The PUCS (Predicted Utility co-exist Structure)

Based on the item's coexistence analysis, we proposed a novel structure called PUCS as in fig. PUCS is defined as the triplet in

the form (x, y, PU)∈IXIX R. PU in the triplet is the Predicted utility of item-set xy. The PUCS constructed during the second time database scanning parallel with formation of the initial list of utility for the items.

$$PU = \sum_{xy \in Ti} (iutility(xy) + rutility(xy)) \quad (6)$$

Fig shows the PUCS of our example. The pruning condition is defined as, "if tuple (x, y, PU) does not exist such that PU ≥ minUtil, then itemset p= {xy} and superset are itemsets with low utility, so the itemset does not need to explore further".

	B	C	D	E
A	PU_{AB}	PU_{AC}	PU_{AD}	PU_{AE}
B		PU_{BC}	PU_{BD}	PU_{BE}
C			PU_{CD}	PU_{CE}
D				PU_{DE}
E				

Fig 2. PUCS Structure

	D	E	A	B
C	42	35	16	18
D		33	33	23
E			40	23
A				
B				

Fig 3. PUCS of the sample database

The PUCS was constructed during the second time database scanning with the construction of an initial utility list of items. Consider the element of PUCS for item c & d, the triplet <c, d, PUcd>. Here PUcd is the sum of iutility and rutility values of itemset {cd} that can be calculated initially at a time of database scanning. We also propose a unique strategy for pruning called PUCP (Predicted Utility Co-exist Pruning) to minimize the number of join operations that uses PUCS.

3.2. The PUCP (Predicted Utility Co-exist Pruning)

According to property 1, Previous algorithms like HUI-miner,mHUI-Miner, and ULB-Miner trim searching space, using the addition of iutility and rutility values of an itemset. For any itemset {xy}, these algorithms construct itemset {xy} 's utility list even though it is a low utility itemset. Then decide whether itemset {xy} should extend further based on sum of iutility and rutility values. These algorithms perform a number of costly utility list join operations for constructing low utility itemset.

Our proposed PUCP eliminates the joining operation for the low utility itemset. For constructing the utility list of itemset {xy} our proposed algorithm, PUCP-Miner, checks the element <x, y, PUxy> fromPUCS. Suppose such an element does not exist in the PUCS where PUxy ≥ MinUtility, itemset {xy} is discarded directly without constructing the utiliy list of itemset {xy}. As a result, it will minimize join operations of utility list.

Take the MinUtility threshold 40 as an example. To construct the itemset {cd} and its utility list, apply join operation on utility list of individual items c & d. According to PUCP, check the PUcd from the PUCS that is 42, so join operations has performed. While for construction of itemset {ce} the PUce from the PUCS is 35, less than MinUtility threshold, so there is no need to perform the join operation. By removing pointless join actions, this pruning method allows PUCP to dramatically reduce the amount of join operations.

Algorithm 1: Creating preliminary utility lists & Build RPUCS

Input: - Transaction Database DB, Minimum utility threshold minUtil

Output: - List of Utilitylist of each promising items LULs

1. Scan the DB
2. compute TWU of all items i
3. Calculate Support_Count of each item i
4. If TWU(i) <MinUtility Threshold Discard item i.
5. Rearrange the items in transaction as SCAO (Support Count Ascending Order)
6. Scan the database DB again
7. Create first list of utility for each favorable item.
8. Build the RPUCS (Reduced Predicted Utility Co-exist Structure)

Algorithm 2: Mining Algorithm

Input: -LUL - List of UtilityList of 1-itemset

Prev.UL- previous item's utiitylist initially it is empty,

MU – User specific threshold for Minimum utility

Output: - itemsets with high utility

```
1. For each element K in LUL do
2.    If TOTAL of K's iutil ≥ MinUtil then
3.       Add K & its previous itemsets in resultset
4. End If
5.    If TOTAL of all K's iutil&rutils ≥ MU then
6.    Create empty extUL;
7.       For each element L follow K in LUL do
8.          If ∃(K,L,PUKL) ∈ RPUCS such that
               PUKL ≥ MinUtility then
                  extUL=extUL+ join(Prev.UL,K,L)
            Endif
9.       Endfor
10.         Mining(K, extUL, MU)
11.   Endif
12.  Endfor
```

Algorithm 3: Join Algorithm

Input: - ULPrev- itemset Prev 's utility list.

ULK– itemset K's utility list,

ULL – itemset L's utility list

Output: - ULKL- itemset KL's utility list.

1. Initialize ULKL is NULL
2. Eoreach component Ek$\in$ ULK
3. if ∃ component EL $\in$ULL&&Ek.Tid == EL.Tid then
4. if ULPrev ≠ empty then
5. Search component EPrev∈ ULPrev have same Tid
6. Create new component Where EkL = <Ek.Tid, Ek.iutil+EL.iutil – Eprev.iutil,EL.rutil>
7. else
8. EkL= <Ek.Tid,Ek.iutil+EL.iutil – Eprev.iutil,EL.rutil>
9. Endif
10. Insert component EkLtoULKL
11. Endif
12. Endfor

return ULKL

4. Results & Discussion

In depth experiments have been performed on verity of real datasets with different MinUtility percentages. The author compared proposed PUCP-Miner with HUI-Miner, mHUI-Miner and ULB-Miner with regards to memory requirement and execution time. The java environment was used to implement each experiment. All the test were executed on a computer with 8GB RAM and 3.4GHz intel core i5 CPU. Standard real-time datasets were used in the experiments to record the algorithm's performance. Table 4 lists the dataset's properties and a full description.

4.1. Result Analysis

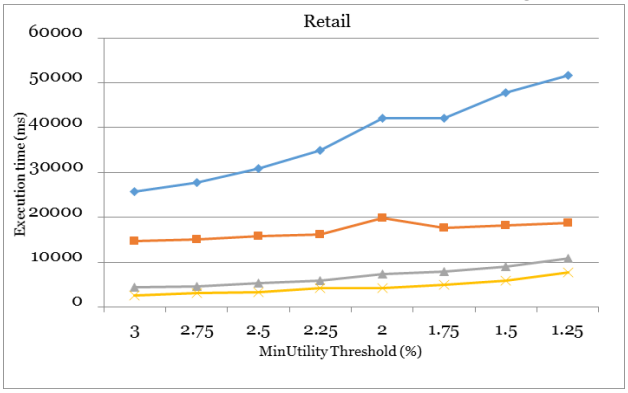
We have executed HUI-Mniner, mHUI-Miner, ULB-Miner, and PUCP-Miner on various real datasets with decreasing order of MinUtility threshold until it takes too long execution time or run out of memory. We noted the execution time and required memory for different datasets. The execution time for various datasets of above-mentioned algorithms is as shown in fig 4.

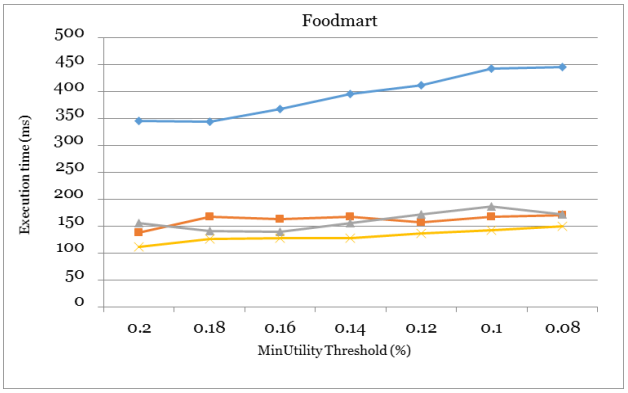
Table 4. Dataset characteristics

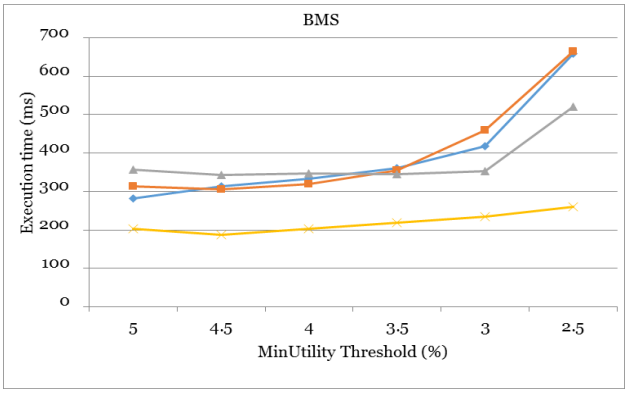
Sr.No	*Name of Dataset*	*#Transactions*	*#Items*	*Average Length*
1	Foodmart	4141	1559	4.4
2	Retail	88,162	16,470	10.3
3	BMS	59,602	497	2.51
4	Kosark	990002	41270	8.1000
5	ecommerce	14975	3468	11.71

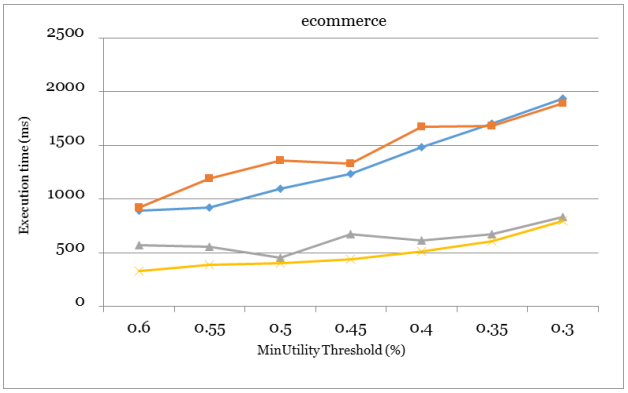
4.1.1. Execution Time Analysis

From the experimental result, it has been observed that on ecommerce dataset, Compared to HUI-Miner, mHUI-Miner, and ULB-Miner, suggested PUCP-Miner is almost 62% faster, 65% faster, and 20% faster, respectively. On the BMS dataset, PUCP-Miner takes nearly 45% less time than HUI-Miner, 46% less time than mHUI-Miner, and 42% less time than ULB-Miner. On the Foodmart dataset, almost PUCP-Miner is 67% quicker than HUI-Miner, 18% quicker than mHUI-Miner, and 18% quicker than ULB-Miner. On the retail dataset, proposed PUCP-Miner almost takes 88%, 74% and 35% less running time than HUI-Miner, mHUI-Miner and ULB-Miner respectively. Lastly, on the kosark dataset, PUCP-Miner is almost 10% faster than HUI-Miner, 14% faster than mHUI-Miner, and 19% faster than ULB-Miner. From the execution time analysis, proposed PUCP-Miner outperforms on retail, BMS and eCommerce datasets. While it is a satisfactory improvement in execution time on foodmart and kosark datasets.









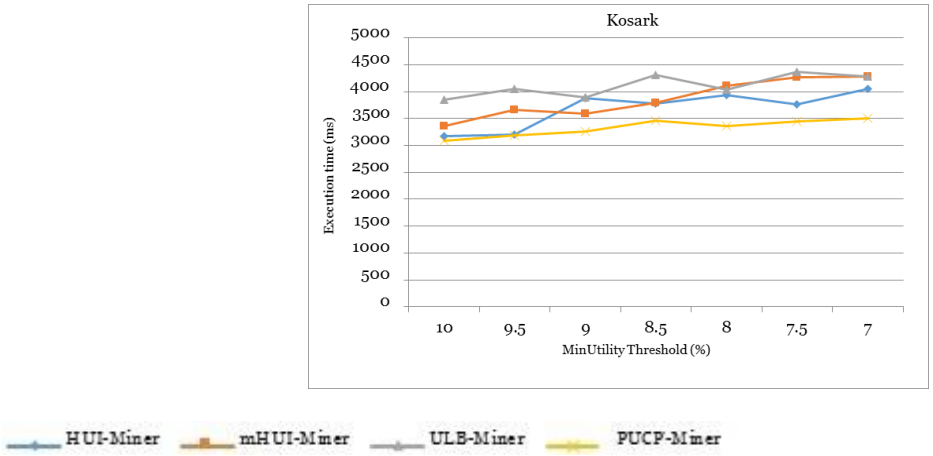


Fig 4. Execution Time comparisons

4.1.2. Memory Analysis

We recorded the required memory for executing the algorithms on various real-time datasets. Accorduing to the result analysis of experiments proposed PUCP-Miner takes almost 46% and 8% less memory than mHUIMiner and ULB-Miner, respectively, on ecommerce dataset. It takes nearly 3%, 12%, and 13% less memory than HUI-Miner, mHUIMiner, and ULB-Miner, respectively on BMS dataset. On the Foodmart dataset proposed, PUCP-Miner consumes 41% less memory than mHUI-Miner and ULB-Miner. On the retail dataset, PUCP-Miner consume 32% and 12 % less memory then mHUI-Miner and ULB-Miner respectively. Finally, on the kosark dataset PUCP-Miner algorithm consume more memory than HUI-Miner, mHUIMiner and ULB-Miner. From the result, it has been observed that HUI-Miner required less memory with the cost of execution time. It is very time-consuming.

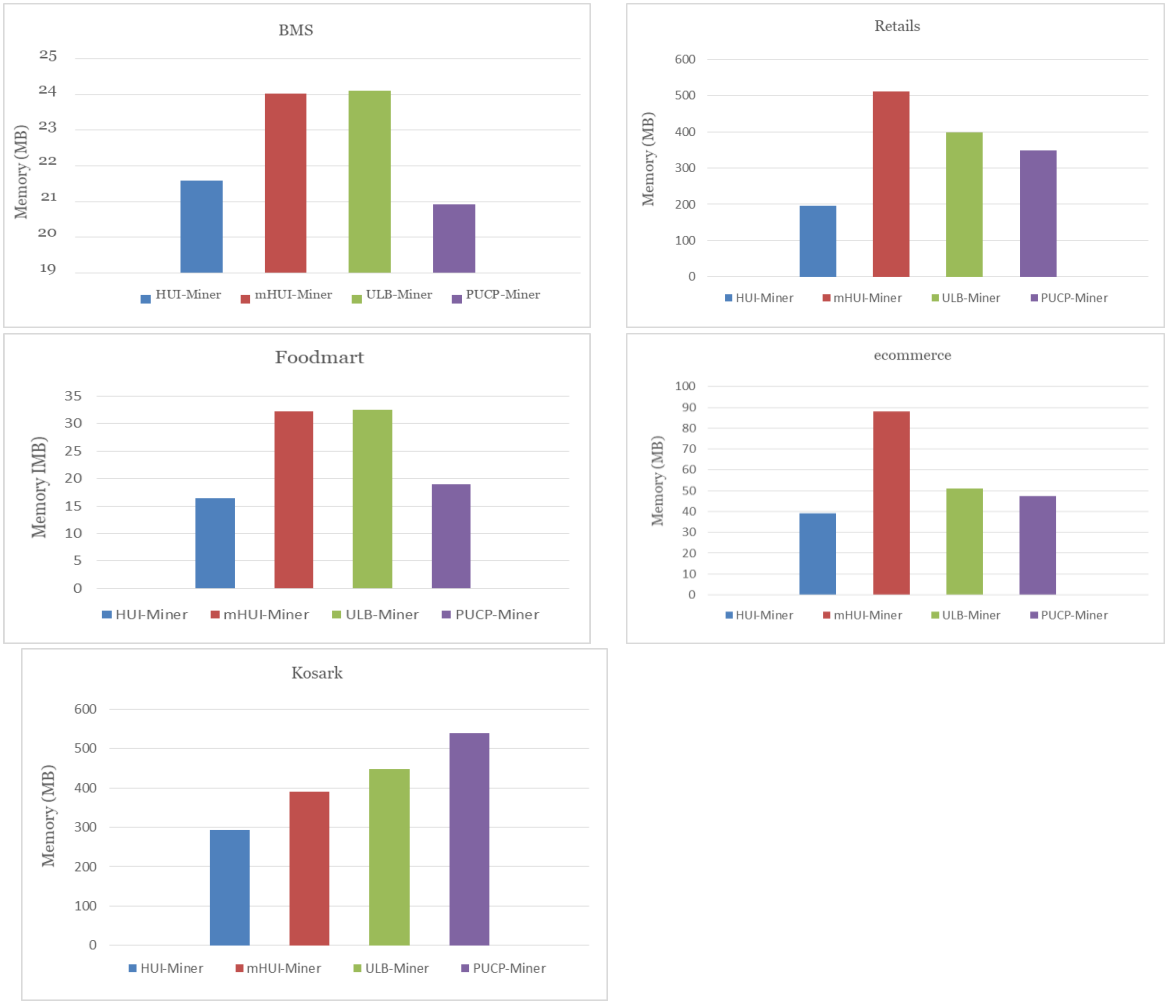


Fig 5. Memory Requirement comparisons

5. Conclusion

HUIM is a core of many real application for decision making. Approaches based on utility list for HUIM are recent as well as efficient. By reducing the amount of expensive utility list join operations, the performance of the mining methods based on utility list-High Utility itemset has increased. Proposed PUCP-Miner uses the novel pruning method PUCP to decrease utility list join counts. PUCP uses PUCS to eliminate the unnecessary join operations to construct the low utility itemset. Hence, according to the experiment's result analysis, it is proven that suggested PUCP-Miner is faster and memory efficient approach than various state-of-art methods i.e HUI-Miner, mHUI-miner and ULB-Miner. While it is lagged in memory utilization only with HUI-Miner, it has noticeable improvement in running time compared to HUI-Miner.

6. Future Work

In this work, only static dataset has been taken into consideration. But in a real sense, the transaction datasets are continuously updated. This work can be extended for dynamic and stream datasets. The itemset correlation can also be considered.

References

[1] D.-N. Le Ashour, Amira S., Nilanjan Dey, "Biological data mining: Techniques and applications,"Min. Multimed. Doc., vol. 1, no. 4, pp. 161–172, 2017.

[2] R. S. Agrawal, Rakesh, "Fast algorithms for mining association rules,"Proc. 20th int. conf. very large data bases, VLDB, vol. 1215, pp. 487–499, 1994.

[3] H. J. H. Yao, Hong, "Mining itemset utilities from transaction databases,"Data Knowl. Eng. 59, vol. 59, no. 3, pp. 603–626, 2006.

[4] Malla, S., M. J. . Meena, O. . Reddy. R, V. . Mahalakshmi, and A. . Balobaid. "A Study on Fish Classification Techniques Using Convolutional Neural Networks on Highly Challenged Underwater Images". International Journal on Recent and Innovation Trends in Computing and Communication, vol. 10, no. 4, Apr. 2022, pp. 01-09, doi:10.17762/ijritcc.v10i4.5524.

[5] A. M. Hu, Jianying, "High-utility pattern mining : A method for discovery of high-utility item sets,"Pattern Recognit., vol. 40, no. 11, pp. 3317–3324, 2007.

[6] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets,"Pacific-Asia Conf. Knowl. Discov. Data Mining, Springer, Berlin, Heidelb., pp. 689–695, 2005.

[7] J. Han, J. Pei, and Y. Yin, "Mining FrequentPatterns without Candidate Generation,"ACM sigmod Rec., vol. 1, no. 29, pp. 1–12, 2000.

[8] Kose, O., & Oktay, T. (2022). Hexarotor Yaw Flight Control with SPSA, PID Algorithm and Morphing. International Journal of Intelligent Systems and Applications in Engineering, 10(2), 216–221. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/1879

[9] Ho R. Ryang, Heungmo, Unil Yun, "Fast algorithm for high utility pattern mining with the sum of item quantities,"Intell. Data Anal., vol. 20, no. 2, pp. 395–415, 2016.

[10] V. Tseng, C. Wu, B. Shie, and P. Yu, "UP-Growth: an efficient algorithm for high utility itemset mining,"Discov. Data Min., pp. 253–262, 2010.

[11] Y. Shen, "Objective-Oriented Utility-Based Association Mining," in In 2002 IEEE International Conference on Data Mining, 2002. Proceedings, 2002, pp. 426–433.

[12] P. F.-V. Qu, Jun-Feng, Mengchi Liu, "Efficient Algorithms for High Utility Itemset Mining without Candidate Generation,"High-Utility Pattern Mining, Springer, Cham, 2019.

[13] Sally Fouad Shady. (2021). Approaches to Teaching a Biomaterials Laboratory Course Online. Journal of Online Engineering Education, 12(1), 01–05. Retrieved from http://onlineengineeringeducation.com/index.php/joee/article/view/43

[14] V. S. Tseng, B. Shie, C. Wu, and P. S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases,"IEEE Trans. Knowl. Data Eng., vol. 25, no. 8, pp. 1772–1786, 2012.

[15] W. Song, Y. Liu, and J. Li, "Mining high utility itemsets by dynamically pruning the tree structure,"Springer Sci. Media New York, pp. 29–43, 2014.

[16] Gill, D. R. . (2022). A Study of Framework of Behavioural Driven Development: Methodologies, Advantages, and Challenges. International Journal on Future Revolution in Computer Science &Amp; Communication Engineering, 8(2), 09–12. https://doi.org/10.17762/ijfrcsce.v8i2.2068

[17] A. Y. Peng, Y. S. K. B, and P. Riddle, "mHUIMiner : A Fast High Utility Itemset Mining Algorithm for Sparse Datasets,"Pacific-Asia Conf. Knowl. Discov. Data Mining, Springer, Cham, pp. 196–207, 2017.

[18] J. Liu, M Qu, "Mining High Utility Itemsets without Candidate Generation Categories and Subject Descriptors," in Proceedings of the 21st ACM international conference on Information and knowledge management, 2012, pp. 55–64.

[19] H. Yao, H. J. Hamilton, and C. J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases,"Proc. 2004 SIAM Int. Conf. Data Min., vol. Society fo, pp. 482–486, 2004.

[20] Li, Yu-Chiang, Jieh-Shan Yeh, "Isolated items discarding strategy for discovering high utility itemsets,"Data Knowl. Eng., vol. 64, no. 1, pp. 198–217, 2008.

[21] Q. D. Philippe, F. H. Ramampiaro, and K. Nørv, "Efficient high utility itemset mining using buffered utility-lists,"Appl. Intell., vol. 48, no. 7, pp. 1859–1877, 2018.

[22] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning,"Springer, Cham. pp. 83–92, 2014.

[23] V. Tseng, C. Wu, B. Shie, and P. Yu, "UP-Growth: an efficient algorithm for high utility itemset mining," in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, 2010, pp. 253–262.

[24] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation,"ACM Int. Conf. Proceeding Ser., pp. 55–64, 2012.