Parallel Algorithm to Efficiently Mine High Utility Itemset

Eduardus Hardika Sandy Atmaja✉[1,2][0000-0003-1739-0116] and Kavita Sonawane[1][0000-0003-0865-6760]

[1] St. Francis Institute of Technology, Mumbai, India
eduardus@student.sfit.ac.in, kavitasonawane@sfit.ac.in
[2] Sanata Dharma University, Yogyakarta, Indonesia

Abstract. Finding high utility itemset (HUI) from transactional databases like customer transaction data is not an easy task. The generated HUI should represent the real world condition which contains set of items that likely to be purchased together and also has high profit. Many algorithms have been introduced to overcome that issue. Parallelism is one of the good architectures that can be adapted for mining algorithms and it is rarely used for HUI Mining. The basic idea that can be adapted is by dividing the mining process based on sub search spaces in concurrent time to boost the performance. This paper proposed two new frameworks that adapted parallel mining namely CLB and PLB which are extension of ULB-Miner algorithm. CLB is hybrid algorithm from CTU-PRO and ULB-Miner, and PLB is proposed because of inefficient process in CLB. From the experimental evaluation, PLB outperforms ULB-Miner in all cases regarding execution time.

Keywords: High utility itemset · Data mining · Parallel processing · CLB · PLB

1 Introduction

Market basket analysis is a common instance of frequent itemset mining (FIM) which specifically deal with transactional databases such as customer purchase history. FIM helps to reveal products that are likely to be purchased together by analyzing customer buying habit. This technique commonly used by retailers to increase sales. They can optimize product bundling, placement, recommendation, and offer special deals to get customer satisfaction. Agrawal and Srikant [1] introduced FIM along with apriori as the algorithm that can efficiently discover interesting patterns. There are several algorithms that were introduced afterwards [2-4]. FIM uses minimum support threshold to identify which itemset is interesting or has high frequency in the database. This condition makes the interesting patterns become less meaningful in real world situation. For example, tea and sugar are items that are likely to be purchased together but they give less profit when compared to washing machine.

To solve problem in FIM algorithms, high utility itemset mining (HUIM) was introduced. To find the profitable patterns, HUIM uses additional parameter such as quantity and profit. HUIM uses minimum utility threshold to decide whether an itemset is interesting or not. In HUIM, every item is not seen as equal like in FIM because every item has their own weight. That is the reason why HUIM is more complex thus designing efficient algorithm is very important. HUIM has been studied by many researchers, tree structure [6, 9, 10, 17] and utility list based structure [7, 11-14] are the most interesting extensions because it has efficient data structure and also has efficient mining algorithm. Hybrid implementation [15, 16] is also done to combine advantages from different data structures to improve the performance. Moreover computation technology such as multicore processor can be used as a powerful infrastructure to execute many tasks in a concurrent time that leads to efficient mining process [17-19].

In this paper, two frameworks are proposed for HUIM. The first framework is by combining utility list and tree based algorithm to get advantages of them and applying parallel processing to the mining process. The second framework is by creating new parallelism framework to improve efficiency of ULB-Miner [7] algorithm.

2 Literature Review

FIM has been studied by many researchers, Apriori [1] is the first algorithm to solve FIM that introduces downward closure property. It means that for every itemset that has support below the minimum support threshold, all of its superset is infrequent, thus can be removed from the search space. This Apriori idea greatly reduces the search space by removing unnecessary candidates. UMining and UMining_H [5] is extension of Apriori to solve HUIM. It is designed more efficient than apriori but it may discover incomplete HUIs, moreover it also does not satisfy downward closure property. Two Phase [8] is then introduced that applies downward closure property. Transaction weighted utilization (TWU) is the upper-bound to identify the unpromising item by applying transaction weighted downward closure (TWDC) strategy. All itemsets that has TWU below the minimum utility threshold can be pruned, thus a complete HUIs can be discovered. This concept is used by many tree based algorithms [6, 9, 10]. CTU-PRO[6] has efficient tree structure called CUP-Tree. It is more compressed than the other above trees structures that makes the algorithm efficient.

The main problem in tree based algorithm is that when the number of transaction is large then it becomes inefficient because more memory and time are needed to build and mine the tree. Utility list based algorithm is then introduced. HUI-Miner [11] is the first algorithm inspired by ECLAT [2]. The mining process works by combining itemsets within the utility list. This combination procedure is costly when there are too many items. FHM [12] solved the problem by developing EUCS (Estimated Utility Co-Occurrence Structure) and EUCP (Estimated Utility Co-occurrence Pruning) as the pruning strategy. This concept is used by many researchers to efficiently prune the search space. Some other utility list based algorithms are [7, 13, 14], ULB-Miner [7] has utility list buffer (ULB) that works like memory pipeline which reuses data segments whenever it is no longer needed thus makes efficient in memory consumption.

mHUIMiner [15] and UFH [16] are hybrid algorithms that combine tree and utility list based algorithms. The tree structure is used to generate candidate itemset and helps to prune the low utility itemsets and the mining process is done by utility list based. With this strategy, the search space can be reduced so it can reduce the memory usage and running time. Parallelism can be used to improve the performance of HUIM algorithms. Since the search space is independent then parallel mining is possible to do. The basic idea is by dividing the search spaces that can be mined concurrently. CHUI-Mine [17], pEFIM [18] and MCH-Miner [19] are algorithms that implemented parallel processing to mine HUI. pEFIM which is extension from EFIM [20] and MCH-Miner which is extension from iMEFIM [21] are the efficient two that apply simple load balancing to manage the resources. This strategy is efficient to boost the original algorithm performance.

3 Problem Statement

From a dataset, let $I = \{i_{1,} i_{2,} i_{3,} \dots i_n\}$, where I is list of distinct items from dataset. Let $D = \{T_1, T_2, T_3, \dots T_n\}$, where D is list of transactions from database. Each transaction has an id called T_{id}. Each item i in T_i has a quantity that represents how many i appeared in T_i. Every item $i \in I$ is related to a profit $p(i)$. Table 1 shows example of D and Table 2 shows example of profit.

Problem Statement : Let D is transactional database and $minutil$ is minimum threshold of utility given by the user, then the problem in HUIM is that performs efficient mining process to reveal itemsets that have utility higher than $minutil$.

Table 1. Example of database

Tid	Transaction (item:quantity)	tu
T_1	(a:4), (b:1), (c:2)	9
T_2	(c:1), (d:3)	13
T_3	(a:2), (d:3)	14
T_4	(a:2), (b:3), (c:2), (d:1)	17
T_5	(b:1), (c:5), (e:3)	14

Table 2. Example of profit

Item	Profit
a	1
b	3
c	1
d	4
e	2

4 Proposed Algorithm

This research work presents two strategies being proposed and experimented namely hybrid framework termed as CLB and parallel framework termed as PLB. The pseudocode for CLB and PLB is given in Algorithm 1 and Algorithm 2.

4.1 Hybrid Algorithm : CLB (ULB-Miner with CTU-PRO)

The hybrid idea of CLB is by modifying CUP-Tree [6] and combining with ULB-Miner [7]. The proposed data structure namely CLB-Tree, a modified version of CUP-Tree is constructed to maintain the database, reduce transactions, and manage parallel mining. The CLB-Tree construction is similar to CUP-Tree, the differences are 1) Every node maintains items, TWU and utilities from similar transactions. It supports parallel processing by saving time to traverse all descendant nodes 2) Every node contains calculated utilities so that there is no need to do repeated calculations. The CLB mining process occurs during the tree construction with a specific condition based on step 3c and 3d [17] in Algorithm 1.

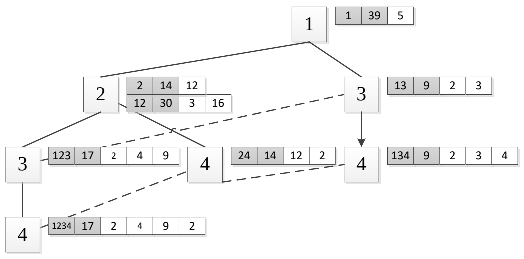


Fig. 1. CLB-Tree after inserting T_4

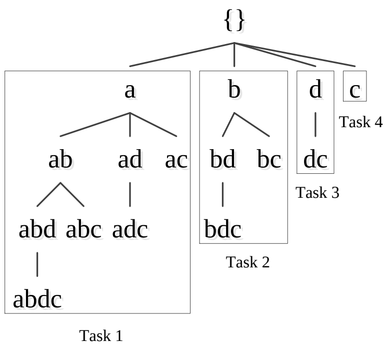


Fig. 2. Enumeration tree distribution

Fig. 1 illustrates CLB-Tree after inserting T_4. From our running example, T_1 is mapped into {(1:2), (3:3), (4:4)} and inserted into the tree. The count of these items are deducted by 1 so that the new set of count is $Count$ = {3, 3, 2, 2}. T_2, T_3, T_4 are inserted in the same way and $Count$ = {1, 0, 1, 0}. Count of item {d} and {a} are 0 so we can do local mining immediately using concurrent process for prefix 2 and 4.

Algorithm 1: CLB | **Algorithm 2**: PLB

Input : database D, minimum utility threshold minUtil
Output : high utility itemset $HUIs$

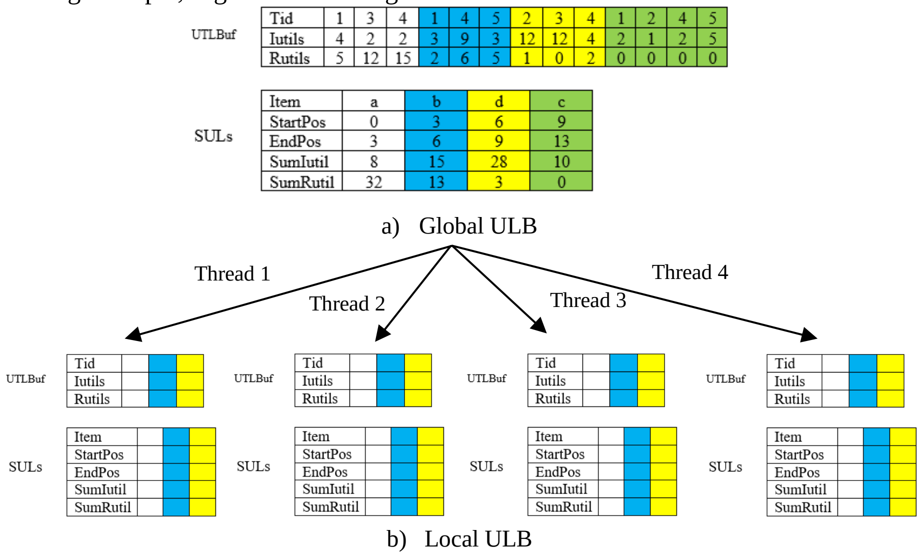
```
Step 1:Construct global item table
Step 2:Initialize CLB-Tree
Step 3:For each t ∈ D
a. Revise and map t
b. Insert t and update
   items, TWU and
   utilities for each node
   i ∈ t
c. Reduce i.count ∈ t in
   global item table by 1
d. If i.count==0 then
   retrieve all node i and
   do parallel mining
   using ULB-Miner
Step 4:Collect all HUIs
```

```
Step 1:Scan D to find
promising items I where
TWU(i∈I) ≥ minUtil and sort
I into TWU ascending order
Step 2:Scan and revise D to
construct global ULB
Step 3:Set n=number of CPU
cores and create n empty
local ULB
Step 4:For each task i ∈ I
do parallel:
a. If no local ULB available
   then wait. Else,
b. Assign and mine task i
   using available local ULB
c. Release local ULB and
   notify main thread
Step 5:Collect all HUIs
```

The next proposed strategy is that we distribute the search space based on prefix tree, overall step 3 in Algorithm 1 explain this condition. Fig. 2 shows enumeration tree for the search space. For local mining prefix 2 which is item {d}, the mining process only does task 3. Similar to prefix 4 which is item {a}, it only does task 1. The last strategy is by ordering the items. CLB-Tree uses TWU descending order but ULB-Miner uses TWU ascending order. This leads to incorrect calculation of remaining utilities. The order of the items is incorrect and the distribution of the data from CLB-Tree is also incorrect. Then, items in CLB-Tree is sorted into TWU descending order and for same TWU value it is lexicographically sorted into descending order. In ULB-Miner, we sort them into ascending order. With this strategy the order of the data is correct either for ascending or descending task.

4.2 Extended ULB-Miner : PLB (ULB-Miner with Parallel Framework)

PLB is proposed to improve performance of ULB-Miner [7] by applying parallel processing along with load balancing and more efficient memory management. Step 2 in Algorithm 2 is the same process as ULB-Miner to build global ULB. Based on the running example, Fig. 3a shows the global ULB.



UTLBuf

Tid	1	3	4	1	4	5	2	3	4	1	2	4	5
Iutils	4	2	2	3	9	3	12	12	4	2	1	2	5
Rutils	5	12	15	2	6	5	1	0	2	0	0	0	0

SULs

Item	a	b	d	c
StartPos	0	3	6	9
EndPos	3	6	9	13
SumIutil	8	15	28	10
SumRutil	32	13	3	0

Fig. 3. PLB mining strategy

Instead of exploring the search space sequentially like the original ULB-Miner, PLB explores it in parallel guided by enumeration tree in Fig. 2. The mining process in ULB-Miner uses the ULB as a memory pipeline. The explored data is inserted as a new data segment in the last index+1 of UTLBuf and SULs from main ULB. This data segment can be reused when it is no longer needed. This strategy cannot be applied to PLB because it causes collision and synchronization problem between threads. PLB introduces new strategy by applying load balancing and more efficient memory management. PLB limits the number of task pool based on the number of available processors [19]. It also creates and limits the number of local ULB to improve the memory efficiency during exploration. Step 3 and 4 in Algorithm 2 explain this strategy. Fig. 3a and 3b also show illustration of the above explanation. For example, task 1 (item a) is explored by thread 1, task 2, 3 and 4 are explored by thread 2, 3 and 4 respectively. Suppose we have extra item {e} that is executed through task 5, then it waits until any thread finished. Suppose thread 4 is finished and released its local ULB then task 5 is executed and reuses local ULB from task 4.

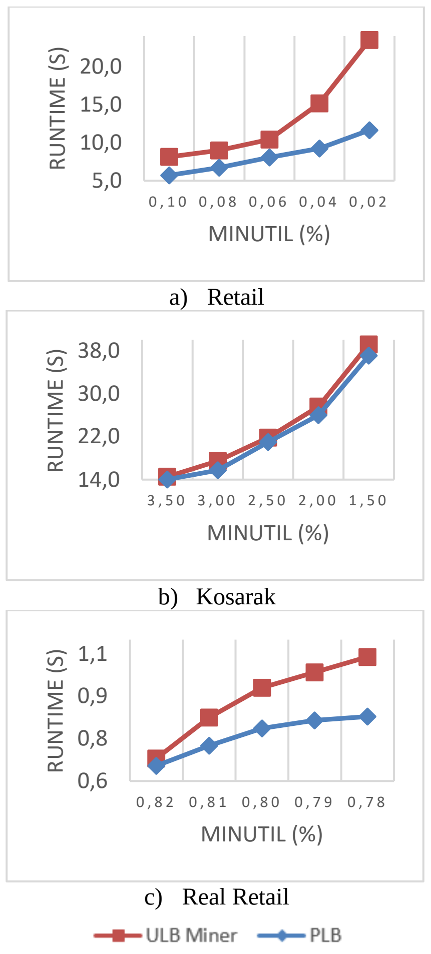


Fig. 4. Execution time for 3 datasets

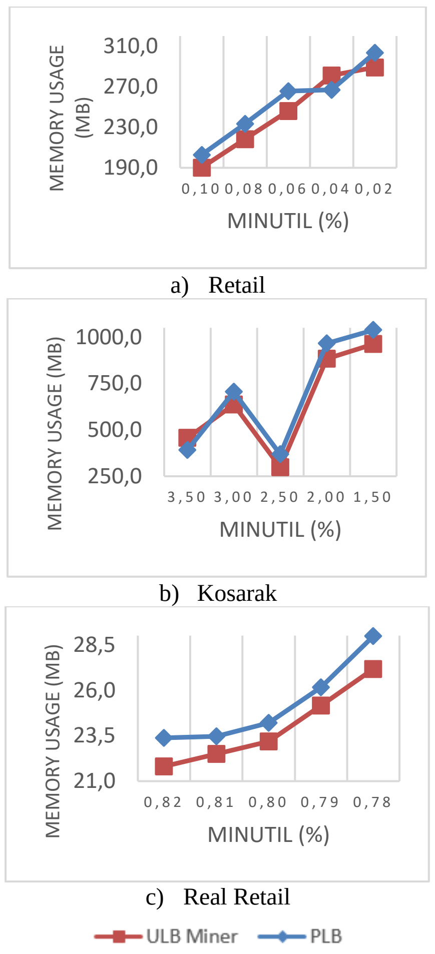


Fig. 5. Memory consumption for 3 datasets

5 Results and Discussion

5.1 Experimental Setup

The experiments used three datasets, two standard benchmark structured datasets retail and kosarak available in SMPF website [22] and a real world customer transaction dataset namely real retail from a store in Malang Indonesia. Retail and kosarak were chosen because these two datasets are complex with many transactions and many distinct items. Table 3 shows the datasets in details. The algorithms were evaluated based on minimum utility, memory usage and the running time. For each algorithm, it was executed using the same minimum utility varied from 0.02-0.1% for retail, 1,5-3,5% for kosarak, and 0,78-0.82% for real retail, then the results are compared. For the implementation, Java Netbeans IDE 8.2 was used. The experiments were performed on a personal computer equipped with 4th generation Intel Core i5-4210U processor @1.7 GHz (4 logical CPUs), DDR3 8 GB RAM, and running Windows 10 Pro.

Table 3. Datasets details

No.	Dataset Name	Distinct Items	Transactions Count	Avg. Length
1	Retail	16.470	88.162	10
2	Kosarak	41.270	990.002	8
3	Real Retail	3.114	24.511	3

5.2 Results and Analysis

Table 4 shows comparison for CLB and PLB with ULB-Miner performances which clearly states that CLB is not contributing positively with the intended objectives. For large database it is inefficient because it needs more memory and time to build the tree, moreover creating many ULB for each local mining is inefficient. However the proposed parallel framework named PLB is performing quite better in terms of running time as compared to CLB and the existing ULB-Miner algorithm.

Table 4. Comparative analysis (ULB-Miner, CLB and PLB)

Dataset	ULB-Miner (Avg)		CLB (Avg)		PLB (Avg)	
	Runtime (s)	Memory (MB)	Runtime (s)	Memory (MB)	Runtime (s)	Memory (MB)
Retail	13,234	245,026	13,331	679,711	8,284	254,441
Kosarak	24,119	647,525	96,474	1.897,213	22,758	693,611
Real Retail	0,892	23,960	0,772	34,184	0,762	25,234

Observations: Best cases runtime (PLB-0,762 s) and memory usage(ULB-Miner-23,960 MB, PLB-25,234 MB**)**

Runtime.
The experiments were done by setting the maximum task pool number to 4 since the processor has 4 cores. Fig. 4 shows the results of the running time. It is clear that PLB has better performance than the ULB-Miner for all cases. The PLB running time is up to two times faster than ULB-Miner. For each dataset, let X be ULB-Miner runtime and Y be PLB runtime. For retail dataset $X = 1{,}5Y$, kosarak $X = 1{,}06Y$ and real retail $X = 1{,}2Y$ in average case. The results of retail dataset is the best and the

results of kosarak is not significant but still better than ULB-Miner. Different dataset has different results because there are some factors that affect the performance viz. distinct item count, transaction count and transaction length. The distinct item count is the most affecting factor because it increases the search space. From the overall results we can conclude that PLB can efficiently mine HUIs by applying parallel processing framework. It outperforms ULB-Miner which uses sequential process.

Memory Usage.
The experiments were also conducted to compare the memory usage. Fig. 5 shows the memory usage results. It is clear that in almost cases, ULB-Miner consumes less memory than PLB. It is reasonable because in parallel processing each thread need memory allocation to maintain their data. Instead of creating one local ULB for each sub search space, PLB allocates only 4 local ULB to maintain data from its sub search spaces. These local ULB are reusable, since every sub search space is independent and can be overwritten. This approach can minimize the memory usage. This is the reason why even PLB has higher memory usage but the difference is not significant.

6 Conclusion

We have proposed two new frameworks to extend the original ULB-Miner to efficiently mine HUI namely CLB and PLB. CLB has a great concept to combine tree based and utility list based algorithm but the performance is relatively poor even it can discover the correct HUI. The worst case is in large databases because it needs more memory and time to build the tree and creating individual ULB for all search spaces. PLB which is extension of ULB-Miner is proposed. Parallel processing, load balancing and more efficient memory management are implemented to improve the efficiency. The results show that PLB outperforms ULB-Miner by up to two times faster. The drawback of PLB is that it consumes more memory to store the data for each active thread. This problem can be investigated in the future work by finding a way to do efficient memory management for parallel processing.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: 20th Int. Conf. on Very Large Data Bases, pp. 487-499. Morgan Kaufmann, San Francisco (1994)
2. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. on Knowl. and Data Eng. **12**(3), 372–390 (2000). doi:10.1109/69.846291
3. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: 2000 ACM SIGMOD Int. Conf. on Management of Data, pp. 1-12. Association for Computing Machinery, New York (2000). doi:10.1145/335191.335372
4. Sucahyo, Y.G., Gopalan, R.P.: CT-PRO: a bottom-up non recursive frequent itemset mining algorithm using compressed fp-tree data structure. In: IEEE ICDM Workshop on Frequent Itemset Mining Implementations. (2004)
5. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. Data & Knowl. Eng. **59**(3), 603-626 (2006). doi:10.1016/j.datak.2005.10.004

6. Erwin, A., Gopalan, R.P., Achuthan, N.R.: A bottom-up projection based algorithm for mining high utility itemsets. In: 2nd Int. Workshop on Integrating Artificial Intelligence and Data Mining, pp. 3-11. Australian Computer Society, Australia (2007)
7. Duong, Q.H., Viger, P.F., Ramampiaro, H., Norvag, K., Dam, T.L.: Efficient high utility itemset mining using buffered utility-lists. Appl. Intell. **48**, 1859-1877 (2018). doi:10.1007/s10489-017-1057-2
8. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: 9th Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 689-695. Springer, Berlin (2005). doi:10.1007/11430919_79
9. Tseng, V.S., Shie, B.E., Wu, C.W., Yu, P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. on Knowl. and Data Eng. **25**(8), 1772-1786 (2013). doi:10.1109/TKDE.2012.59
10. Deng, Z.H.: An efficient structure for fast mining high utility itemset. Appl. Intell. **48**, 3161–3177 (2018). doi:10.1007/s10489-017-1130-x
11. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: 21st ACM Int. Conf. on Information and Knowl. Management, pp. 55-64. Association for Computing Machinery, New York (2012). doi:10.1145/2396761.2396773
12. Viger, P.F., Wu, C.W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: 21st Int. Symp. on Methodologies for Intell. Systems, pp. 83-92. Springer, Cham (2014). doi:10.1007/978-3-319-08326-1_9
13. Viger, P.F., Zhang, Y., Lin, J.C.W., Dinh, D.T., Le, H.B.: Mining correlated high-utility itemsets using various measures. Logic Journal of Interest Group in Pure and Appl. Logics (IGPL). **28**(1), 19-32 (2018). doi:10.1093/jigpal/jzz068
14. Wu, C.W., Viger, P.F., Gu, J.Y., Tseng, V.S.: Mining compact high utility itemsets without candidate generation. In: High-Utility Pattern Mining: Theory, Algorithms and Applications, pp. 279-302. Springer: Cham (2019). doi:10.1007/978-3-030-04921-8_11
15. Peng, A.Y., Koh, Y.S., Riddle, P.: mHUIMiner: A fast high utility itemset mining algorithm for sparse datasets. In: 21st Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 196-207. Springer, Cham (2017). doi:10.1007/978-3-319-57529-2_16
16. Dawar, S., Goyal, V., Bera, D.: A hybrid framework for mining high-utility itemsets in a sparse transaction database. Appl. Intell. **47**, 809-827 (2017). doi:10.1007/s10489-017-0932-1
17. Song, W., Liu, Y., Li, J.: Mining high utility itemsets by dynamically pruning the tree structure. Appl. Intell. **40**, 29-43 (2014). doi:10.1007/s10489-013-0443-7
18. Nguyen, T.D.D., Nguyen, L.T.T., B. Vo.: A parallel algorithm for mining high utility itemsets. In: Proc. 39th Int. Conf. on Information System Architecture and Tech., **853**, pp. 286–295. Springer, Cham (2019). doi:https://doi.org/10.1007/978-3-319-99996-8_26
19. Vo, B., Nguyen, L.T.T., Nguyen, T.D.D., Viger, P.F., Yun, U.: A multi-core approach to efficiently mining high-utility itemsets in dynamic profit databases. IEEE Access. **8**, 85890-85899 (2020). doi:10.1109/ACCESS.2020.2992729
20. Zida, S., Viger, P.F., Lin, J. C.W., Wu, C.W., Tseng, V.S.: EFIM: a fast and memory efficient algorithm for high-utility itemset mining. Knowl. and Information System. **51(2)**, 595-625 (2017). doi:10.1007/978-3-319-27060-9_44
21. Nguyen, L.T.T., Nguyen, P., Nguyen, T.D.D., Vo, B., Viger, P.F., Tseng, V.S.: Mining high-utility itemsets in dynamic profit databases. Knowl. Based Systems. **175**, 130-144 (2019). doi:https://doi.org/10.1016/j.knosys.2019.03.022
22. Viger, P.F.: SPMF : An open source data mining library (2008-2021). http://www.philippe-fournier-viger.com/spmf/