engineering proceedings

MDPI

Proceeding Paper

Efficient Algorithm for Mining Top-*k* High On-shelf Utility Itemsets with Positive/Negative Profits of Local/Global Minimum Count †

Ye-In Chang [1,*], Po-Chun Chuang [1], Yu-Hao Liao [1], Po-Yu Hu [1] and Ting-Wei Chen [1]

[1] dept. of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan; changyi@mail.cse.nsysu.edu.tw, zhungboqun@gmail.com, karta88821@gmail.com, HuPY@db.cse.nsysu.edu.tw, ChenTW@db.cse.nsysu.edu.tw

* Correspondence: changyi@cse.nsysu.edu.tw

† Presented at the 2025 IEEE 5th International Conference on Electronic Communications, Internet of Things and Big Data, New Taipei, Taiwan, 25–27 April 2025.

Abstract: High utility itemset mining (HUIM) utilizes the threshold value to extract HUI from the transactional database. However, it is difficult to define an optimal threshold value, since it depends on the domain knowledge of the application. Therefore, Top-k HUIM is used to solve the problem of setting a threshold. A user can define a k value, which represents the number of HUIs. Moreover, there exist itemsets occurring at a specific time interval, which can become HUI. Since the traditional HUIM algorithm does not consider the transaction with the time interval, the HUIM algorithm cannot be used directly. Therefore, high-on-shelf utility itemset mining (HOUIM) is used to address the above problem in this study. The proportion of the utility value of the item in all of the time intervals with the itemset is used for determining whether the itemset is HOUI or not. In the Top-k HOUIM, the KOSHU algorithm is used based on the data structure, ignoring the item with the negative profit in overestimating the utility of the itemset. The KOSHU algorithm needs less processing time. However, the KOSHU algorithm has to scan the database twice and sort the database once. Therefore, we developed an efficient algorithm based on the TIPN Table to mine Top-k HOUIs. The developed data structures include TIPN and MINC tables, IO Bitmap, and TIUL. In the TIPN table, we recorded positive items, positive utilities, negative items, and negative counts. The MINC table is used for storing the local/global counts of all of the items with negative profits. In the algorithm, we scanned the database only once. The developed algorithm is more efficient than the KOSHU algorithm.

Keywords: data mining; high on-shelf utility itemset mining; negative unit profits; static transac-tional database; top-k high utility itemset mining

Academic Editor: Firstname Lastname

Published: date

Citation: To be added by editorial staff during production.

1. Introduction

In the frequent weighted itemsets mining (FWIM) [10], the frequency and the weight of the item are considered. The high utility itemsets mining (HUIM) [7,8] has become popular recently. If the utility value of itemset *X* is not less than the minimum threshold value, then itemset *X* is a HUI. However, it is difficult to define an appropriate threshold value. To address the issue, the Top-k HUIM [8] has been proposed. The default threshold value

is set to 0, and then several threshold values are used, such as remaining item utility (RIU) and remaining utility constraint (RUC).

The traditional HUIM algorithm considers all of the transactions in the database on-shelf. The high on-shelf utility itemset mining (HOUIM) [3] has been proposed by considering the on-shelf time interval of the transaction to find itemsets with high utility in the specific time interval. All of the transactions are added with another field as a time interval, which indicates that the transaction is on-shelf as shown in Table 1. The related profit of items a, b, c, and d is 2, 4, 1, and 3, respectively.

Table 1. Database D1 with on-shelf time intervals

TID	Transaction	Interval
T_1	(a, 1)(b, 2)(c, 1)	1
T_2	(c, 3)(d, 2)	1
T_3	(b, 2)(c, 4)(d, 3)	2
T_4	(b, 6)(d, 2)	2

The total utility of a time interval h is denoted as TIU (h), which is calculated as the sum of the utility of transactions in which the time interval is h. The utility of itemsets in a certain time interval is equal to the summation of the utility of all itemsets in a certain interval. The result of the total utility of time interval 1 is 20 and that of time interval 2 is 51. The relative utility of the given itemset X is used for determining whether the itemset is a HOUI.

Singh *et al.* proposed the top-k high utility itemset mining (TKEH) algorithm [8] for mining the Top-k HUIs. Srikumar proposed the THUI algorithm [6] to mine the Top-k HUIs. Ashraf *et al.* proposed the TKN algorithm [2]. Later, HOUIM [4] is considered as the time interval. The redefined transaction-weighted utility (RTWU) [9] is a utility overestimated value that ignores items with negative profits. The KOSHU algorithm [4] is constructed based on the utility list structure to mine Top-k HOUIs. The KOSHU algorithm reduces the number of candidates during the mining process. However, the KOSHU algorithm needs to scan the database twice.

To reduce the number of database scans in the static database, we developed an efficient algorithm called TIPN-table-based to extract Top-k HOUIs. We applied the local and global concepts to deal with items with a negative profit. We also introduced the IO_Bitmap to record the occurrence of all of the items according to different time intervals. We proposed two pruning strategies based on the research results. The developed TIPN-Table-based algorithm is more efficient than the KOSHU algorithm.

2. Koshu Algorithm

The KOSHU algorithm [4] mines Top-k HOUIs. The KOSHU algorithm calculates RTWU values for all of the items and sorts them in the defending order. Then, they apply two threshold-increasing strategies. Moreover, they used a pruning strategy to effectively diminish the processing time. However, the KOSHU algorithm requires iterating the database twice to mine the Top-k HOUIs. Moreover, the utility list for each item was constructed by the KOSHU algorithm.

3. TIPN-table-based algorithm

We used an example database to illustrate the developed algorithm. We described variables, four data structures, three pruning strategies, two strategies for increasing the threshold value.

3.1. Example Database

We used an example database D2 to illustrate the algorithm (Table 2). Each item in the set I has its profit = [a:5, b:-2, c:3, d:-1, e:3, f :4]. Each transaction Tj in database D2 contains a unique identifier TID. Moreover, in each transaction Tj, we use a subset of items I with the related count and a related time interval.

Table 2. Example database D_2 with on-shelf time intervals

TID	Transaction	Interval
T_1	(b, 3)(d, 2)(f, 4)	1
T_2	(b, 2)(c, 7)(d, 4)(e, 5)	1
T_3	(a, 6)(c, 3)(d, 4)	2
T_4	(a, 4)(d, 2)(e, 2)	2
T_5	(b, 3)(c, 8)(d, 5)(e, 4)	3
T_6	(b, 1)(c, 6)(d, 3)	3

3.2. Figures, Tables, and Schemes

P_V al(i) and Q_V al(i) represent the profit and the quantity value of item i, respectively. Moreover, UT (i, Tj) is defined as the Utility of item i in Transaction Tj, which is calculated as the product of P_V al(i) and Q_V al(i). UT (X, Tj) represents the utility of itemset X in transaction Tj, which is calculated as the cumulative utility. TotalU (T) represents the total utility of transaction T, which is the sum of the utility of each item in transaction T. The total utility of each transaction in database D_2 is 8, 23, 35, 22, 21, and 13, for transactions T_1, T_2, T_3, T_4, T_5, and T_6, respectively. Each product is on the shelf at different time intervals. We defined the list of whole time intervals (TIList) as in database D_2. TotalTI(h) represents the utility of time interval h. The total utilities of all of the time intervals in database D_2 are [a, 59], [b, 83], [c, 106], [d, 142], [e, 77], and [f, 14]. The utility of itemset X in time interval h is denoted as UTI(X, h). The relative utility of the itemset is used for determining whether the itemset is an HOUI or not. RelativeU (X) represents the Relative Utility of the itemset, which is computed by the utility of itemset X divided by the total utility of each time interval that contains itemset X in the database.

3.3. Data Structures

In the proposed data structure, we stored the information of each transaction in the TIPN table. Negative items are ignored in the overestimated value of the itemset. We designed the NIMC table to keep the local minimum of each negative item according to different time intervals. Obviously, in mining top-k high on-shelf utility itemset, the threshold is initially set to 0 and it is increased during the mining process. We used the bitmap to store the status of the occurrence of each item according to different time intervals. We used the TIList to achieve the goal. The TIPN table has six columns as shown in Table 3. Each row in TIPN_Table contains a set of transactions at that time interval, and each transaction is classified as positive items and negative items. Furthermore, the utility of each positive item in the transaction is stored. Here, we stored the count of each negative item to construct the NIMC table.

Table 3. TIPN table of database D

Time Interval	TID	Positive Items	Positive Utility	Negaitve Items	Negative Count
1	T_1	f	f: 16	b, d	b: 3, d: 2
	T_2	c, e	c: 21, e: 10	b, d	b: 2, d: 4
2	T_3	a, c	a: 30, c: 9	d	d: 4
	T_4	a, e	a: 20, e: 4	d	d: 2
3	T_5	c, e	c: 24, e: 8	b, d	b: 3, d: 5
	T_6	c	c: 18	b, d	b: 1, d: 3

We constructed the data structure of the NIMC table to decrease the overestimated utility. Moreover, we used two counts, global noise correction (GNC) and local noise correction (LNC), to record the minimum count of each negative item (b and d). The Global minimum count (GMC) of each negative item is also recorded. For each negative item, we calculated the minimum count of each time interval and stored it in the GMC column. When count = 0, we skipped it. For example, for negative item b, we recorded 2, 0, 1, for time intervals 1, 2, 3, respectively, and recorded GMC = 1. IO_Bitmap was used for storing the occurrence of each item in the database and it is constructed during the initial pass of the database. For example, for item c in time interval 1, the related bits are 01.

TIList was used to discover HOUIs efficiently based on the utility list. We defined a list of itemset X as $TIUL(X) = \{(TI, \{(TID, P\_Util, N\_Util, NGC\_Util, R\_Util)\})\}$, where Tl is the time interval, P_Util is the positive utility of itemset X in transaction TrID. N_Util is the negative utility of itemset X in transaction TID and NGC_Util is the negative utility of itemset X that considers GMC in transaction $T_{T\,ID}$. UTGC(I,Tj) = TU (i, Tj), if P_V al(i) > 0. Moreover, UTGC(i, Tj) = GC(i) × P_Val(i), if P_V al(i) < 0. Furthermore, the Remaining Utility (R_Util) of itemset X in transaction $T_{T\,ID}$ is defined as follows.

$$R_{Util(X,T_j)} = \sum_{i \in T_j \wedge i > x \forall x \in X} UTLC(i, T_j) \quad (1)$$

where $UTGC(i, T_j)$ adopts the GMC of each negative item according to the NIMC table. Then, the utility of item i is calculated as UTLC(b, T_1) = GC(b) × P_Val(i) = 2 × (-4) = −8.

The TIList of itemset X stores the positive utility and negative utility of itemset X. Moreover, The list stores the negative utility by considering the GMC of each period. Table 4 shows the TIList of item e. In the mining process, the algorithm constructs a list for each single item in the database. Then, the list is used for extracting HOUIs in the mining procedure. To calculate the utility of the itemset and the overestimated utility of the item, we defined variables as follows. (1) sumP_Util(X) means the sum of Positive Utilities P_Util(X) of TIUL of X; (2) sumN_Util(X) means the sum of Negative Utilities N_Util(X) of TIUL of X.; (3) sumNLC_Util(X) means the sum of Negative Utilities with Local minimum Count NNGC_Util of TIUL of X.; (4) sumR_Util(X**)** means the sum of Remaining Utilities R_Util of TIUL of X.; (5) sumN_Util(X) means the accumulation of Positive Utilities and Negative Utilities of TIUL of X.

Table 4. TIList of item e.

Itemset	Time Interval	TID	P_Util	N_Util	NLC_Util	R_Util
e	1	2	10	0	0	15
	2	4	4	0	0	-2
	3	5	8	0	0	19

3.3. Pruning Strategies

We introduced three pruning strategies. In the first pruning strategy called TWUGC, which is motivated by the RTWU [5], we define the TWUGC as follows.

$$TWUGC(X) = \sum_{X \in T_j \wedge T_j \in D} TotalU\_GC(T_j) \quad (2)$$

$$TotalU\_GC(T_j) = \sum_{i \in T_j \wedge T_j \in D} UTGC(i, T_j). \quad (3)$$

$$UTGC(i, T_j) = \begin{cases} TU(i, T_j), & if\ P\_Val(i) > 0. \\ GC(i) \times P\_Val(i), & if\ P\_Val(i) < 0. \end{cases} \quad (4)$$

The TWUGC of itemset X is the sum of $TotalU\_GC(T_j)$ in transactions, where itemset X appears. $TotalU\_GC(T_j)$ is calculated as the sum of utilities of items in transactions T_j. If the item is negative, the utility of the item is calculated as the product of GMC of the item and the profit of the item. For the second pruning strategy, the RLC pruning strategy prunes hopeless candidates. We utilized the utility list to calculate the overestimated utility value of the relative utility of the itemset. For the third pruning strategy, the TIO pruning strategy prunes the subtrees of the set-enumeration tree which do not appear in the database during the mining process. We introduced the IO Bitmap which recorded the occurrence statuses of all of the items.

3.4. Threshold Increased Strategies

We introduced two strategies with increased thresholds. The RPRU_Size1 strategy was used to calculate the relative utility for all of the positive items in database D and insert those positive items into the RPRU_Size1_List. Then, the RPRU_Size1 strategy sorts the RPRU_Size1 list according to the descending order of relative utilities and it increases the threshold value to the k-highest relative utility in the RPRU_Size1 list. The RRU_Size2 strategy was used to calculate the relative utilities for all of the items in database D and insert them into RRU_Size2 list. The RRU_Size2 strategy sorts the RRU_All_List according to the descending order of relative utilities and it increases the threshold value to the k-highest relative utility in the RRU_All_List.

3.5. Mining Process

The mining process of the developed algorithm is as follows.

3.5.1. Preprocessing Step

In this step, we constructed TIList, TotalTI table, TIPN table, IO table, and NIMC table. In addition, LMC and GCM were added. After scanning the database once, the algorithm performed the RPRU-Size1 strategy to increase the value of ThreVal. To obtain the real relative utility of positive item a, we calculated the utility of positive item a by using the TIPN table. The results showed that UD(a) = 115 TI_Occu_List(a) = {1, 2, 3}, and TI_Occu_Total(a) = 133. The real relative utility of item a is calculated as UD(a) / TI_Occu_Total(a) = 0.86. The result of RPRU_Size1 list which stores real relative utilities of positive items is [Each positive item, RPRU] = [[a, 0.86], [c, 0.54], [e, 0.17], [f, 0.09]]. If

the size of RPRU Size1 list was not less than k, we increased the threshold value to the k-highest value in RPRU_Size1_List. On the other hand, TK_List stored the k-highest HOUIs by the descending order of the relative utility. Therefore, TK_List was updated by RPRU_Size1_List.

Then, we calculated the TWUGC value of all of the items in the database to overestimate the relative utility. We defined a total order (TWUGC_Order) for finding the Top-k HOUIs efficiently. TWUGC_Order has three ordering rules. First, positive items are sorted by the descending order of TWUGC. Second, negative items are sorted by the descending order of TWUGC. Third, negative items are sorted in positive items. TWUGC_Order of database D2 is [f, a, e, c, b, d]. Table 5 shows the result of the sorted TIPN table.

After the TIPN table was sorted by the TWUGC_Order, the developed algorithm created a TIList for all of the single items in the database. The TIList of each item was used for discovering itemsets with a large size by using the itemset expansion method.

Table 5. Sorting TIPN table in TWUGC

Time Interval	TID	Positive Items	Positive Utility	Negaitve Items	Negative Count
1	T_1	f	f: 16	b, d	b: 3, d: 2
	T_2	e, c	c: 21, e: 10	b, d	b: 2, d: 4
2	T_3	a, c	a: 30, c: 9	d	d: 4
	T_4	a, e	a: 20, e: 4	d	d: 2
3	T_5	e, c	c: 24, e: 8	b, d	b: 3, d: 5
	T_6	c	c: 18	b, d	b: 1, d: 3

To increase the minimum threshold value for finding Top-k HOUIs, the algorithm applied another threshold-increased strategy called the RRU-Size2 strategy. In the RRU-Size2 strategy, the real relative utility is calculated for each size 2 itemsets and stored in the RRU_Size2 list. Moreover, the algorithm uses the Time Interval_Utility list and IO_Table to calculate the real relative utility of the size 2 itemset efficiently. For obtaining the real relative utility of itemset x∪y, we need to calculate the utility of itemset $x \cup y$ and the total utility of each time Interval which contains the occurrence of itemset $x \cup y$. The real relative utility of $x \cup y$ can be calculated as $Relative(x \cup y) = UD(x \cup y)/TI\_Occu\_Total(x \cup y)$. The TK list is updated by the RRU_ALL list.

3.5.2. Mining Step

In the mining step, the algorithm applies the pattern growth method to discover Top-k HOUIs. At the beginning of the mining process, the algorithm traverses through all of the TIULs of items from the root node of the set- enumeration tree, i.e. empty itemset. In each iteration, the original itemset P is appended with the current item x to obtain a new itemset $NewP = P \cup \{x\}$. Then, the algorithm checks whether the new itemset NewP is an

HOUI or not. If the relative utility of the itemset is not less than the threshold value ThreVal, the itemset is a HOUI. If the relative utility of the new itemset NewP is greater than the k-highest relative utility in the TK list, the algorithm removes the k-highest itemset and inserts the new itemset NewP into the TK list. Moreover, the threshold value ThreV al is updated as the k-highest relative utility in the TK list. To calculate the relative utility of new itemset NewP, we calculated the utility of new itemset NewP in the database and the total utility of each time interval that contains the occurrence of the new itemset in the database TI Occu_Total. For calculating the utility of new itemset NewP, the algorithm uses TIUL of itemset NewP. The sum of positive utilities and negative utilities of itemset NewP contains sumPN_Util(NewP), which is the accumulation of positive utilities sump_Util(NewP), and the accumulation of negative utilities sumN_Util(NewP).

If sumPN_Util(NewP) is less than 0, the algorithm skips it directly. For calculating TI_Occu_Total, the algorithm performs the AND operation of all of the time intervals of itemset NewP in IO_Bitmap. If the related utility of the new itemset NewP is not less than the threshold value ThreV al, the new itemset NewP is a HOUI. The objective is to discover the Top-k HOUIs within the database. Moreover, the algorithm creates a list called TK list to store the Top-k HOUIs during the mining process. The TK list is a list that dynamically sorts all of the HOUIs in the TK list according to the descending order of relative utilities. If the size of the TK list is less than the user-defined parameter k, the algorithm inserts the new itemset NewP into the TK list directly. If the size of the TK list is equal to the user-defined parameter k, the algorithm checks whether the relative utility of new itemset NewP is greater than the k- highest relative utility in the TK list or not. If the result is true, the new itemset NewP is inserted into the TK list. After the above checking, whether the new itemset NewP is HOUI or not, the algorithm applies the TWUGC pruning strategy to prune unpromising itemsets.

4. Performace Evaluation

The TIPN-table-based algorithm and the KOSHU algorithm were evaluated for their performance [4]. For evaluation, we utilized two databases, real and synthetic databases.

4.1. Performance Model

The real database is the sparse database downloaded from the SPMF library [5]. The retail database has a density of lower than 1% as a sparse database. The value of the time interval is 5, which is equal to the consideration of the KOSHU algorithm [4]. We set k to the range between 50 and 150. The real database has 88162 transactions (containing 16470 items) with density=0.06. For the synthetic database, we utilized four parameters T, I, MI, and NP to experiment with the TIPN-table-based algorithm and KOSHU algorithm, where T represents the total amount of transactions in the database, I represents the total amount of distinct items of the database. MI represents the maximum amount of distinct items of a single transaction and NP represents the percentage of counts of items with negative profits. For example, T_10000_I4000_MI10_NP_80 is a synthetic database with 10000 transactions, 4000 distinct items, up to 10 distinct items in a single transaction, and 80% of items with negative profits. These synthetic databases are obtained from the IBM Almaden Quest research group [1].

4.2. Experiment Results

We compared the performance of the TIPN-table-based algorithm and the KOSHU algorithm [4]. The KOSHU algorithm produced the EMPRS data structure during the preprocessing step, which was time-consuming. Moreover, the number of candidates for the TIPN-table-based algorithm is smaller than that of the KOSHU algorithm. There are two

reasons why the algorithm could generate less number of candidates than the KOSHU algorithm. First, the TWUGC pruning strategy considered the GMC of all of the items with negative profits. Second, the RLC pruning strategy utilizes the LMC of all of the items with negative profits at each time interval. Therefore, the remaining utility of the RLC pruning strategy is tighter than that of the KOSHU algorithm. Moreover, the number of candidates was pruned by the algorithm more than that of the KOSHU algorithm during the mining process.

Figure 1 shows the comparisons of the performance between these two algorithms by using the real database retail. As the value of k is increased, the performance measures including the processing time and the number of candidates of our proposed algorithm are better than those of the KOSHU algorithm. Moreover, the reasons for such results are the same as those reasons described before. For the synthetic databases T_10000_I100_MI10_NP_80 (the dense database), Figures 1(c) and (d) show the comparisons of the two concerned algorithms. The result are similar to the comparison between the two algorithms.

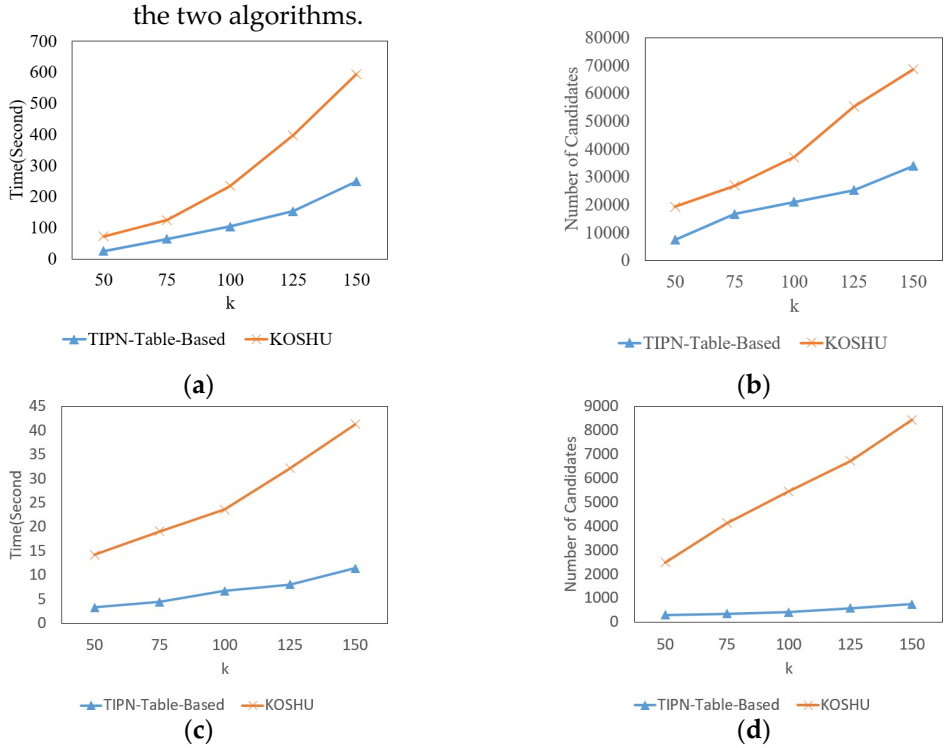

Figure 1. A comparison under the change of k. (a) The processing time by using the real database retail. (b) The total amount of candidate by using the real database retail. (c) The processing time by using the synthetic database T10000_I100_MI10_NP80by. (d) The total amount of candidate using the synthetic database.

5. Conclusion

In this study, we developed the TIPN-table-based algorithm to mine top-k high on-shelf utility itemsets efficiently. The TIPN-table-based algorithm only scans the database once and sorts the database once. Moreover, we proposed the global and local concepts to make the tight upper bound. We utilized a bit map strategy to decrease the processing time. The experiment results showed that the TIPN-Table-Based algorithm better performed than the KOSHU algorithm.

References

1. IBM. IBM Quest Synthetic Data Generation Code. Available online: http://www.almaden.ibm.com/cs/quest/syndata.html (accessed on 5 July 2025).
2. Ashraf, M.; Abdelkader, T.; Rady, S.; Gharib, T.F. TKN: An efficient approach for discovering top-k high utility itemsets with positive or negative profits. *Inf. Sci.* **2022**, *587*, 654–678.
3. Chen, J.; Guo, X.; Gan, W.; Chen, C.-M.; Ding, W.; Chen, G. On-shelf utility mining from transaction database. *Eng. Appl. Artif. Intell.* **2022**, *107*, 1–12.
4. Dam, T.-L.; Li, K.; Fournier-Viger, P.; Duong, Q.-H. An efficient algorithm for mining top-k on-shelf high utility itemsets. *Knowl. Inf. Syst.* **2017**, *52*, 621–655.
5. Fournier-Viger, P.; Lin, C.W.; Gomariz, A.; Gueniche, T.; Soltani, Z.D.A.; Lam, H.T. The SPMF open-source data mining library version 2. In *Proceedings of the 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016), Part III*; Springer: Riva del Garda, Italy, 19–23 September 2016; pp. 36–40.
6. Krishnamoorthy, S. Mining top-k high utility itemsets with effective threshold raising strategies. *Expert Syst. Appl.* **2019**, *117*, 148–165.
7. Lee, J.; Yun, U.; Lee, G.; Yoon, E. Efficient incremental high utility pattern mining based on pre-large concept. *Eng. Appl. Artif. Intell.* **2018**, *72*, 111–123.
8. Singh, K.; Singh, S.S.; Kumar, A.; Biswas, B. TKEH: An efficient algorithm for mining top-k high utility itemsets. *Appl. Intell.* **2018**, *49*, 1078–1097.
9. Singh, K.; Kumar, A.; Singh, S.S.; Shakya, H.K.; Biswas, B. EHNL: An efficient algorithm for mining high utility itemsets with negative utility value and length constraints. *Inf. Sci.* **2019**, *484*, 44–70.
10. Vo, B.; Bui, H.; Vo, T.; Le, T. Mining top-rank-k frequent weighted itemsets using WN-list structures and an early pruning strategy. *Knowl.-Based Syst.* **2020**, *201–202*, 1–12.

11. **Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.