Support-Based High Utility Mining with Negative Utility Values

Pushp and Satish Chand

Abstract High utility itemset mining (HUIM) aims at knowledge discovery from the datasets by finding patterns that have high utility values. Most of the existing algorithms suffer from the drawback of generating huge number of results that overwhelm the decision-making process for industry applications. Also, the real-life datasets often consist of items that have both positive and negative utility values in order to represent the profit and losses, respectively. In this paper, we propose a novel mining algorithm that maps closely to the real-life applications by producing only a reasonable number of outputs based on a support measure, from the datasets that have both positive and negative utility values. Several experiments are undertaken to test the efficacy of the proposed approach. Empirical evaluation suggests that the proposed approach is highly efficient for dense datasets.

Keywords Knowledge discovery · Data mining · High utility itemset mining

1 Introduction

Knowledge discovery in datasets has attracted the attention of the research community in the last decade. Most of the data mining algorithms are designed for extracting imperceptible knowledge from large datasets. The mined information reflects the trends and patterns in the underlying database and can be useful in various paradigms, depending on the use case. One of the earliest applications of data mining is frequent pattern mining (FPM) that aims at discovering the frequently occurring patterns from the customer transaction datasets, which is also referred to as *market-basket analysis*. The mined outputs enhance the decision-making process and aid in business growth-related operations like designing of cross marketing strategies, customer classification and market segmentation.

Pushp (✉) · S. Chand
Jawaharlal Nehru University, New Delhi, India
e-mail: srapushp@gmail.com

S. Chand
e-mail: schand@mail.jnu.ac.in

A. K. Bashir et al. (eds.), *Proceedings of International Conference on Computing and Communication Networks*, Lecture Notes in Networks and Systems 394,
https://doi.org/10.1007/978-981-19-0604-6_18

The commonly used techniques for FPM are based on the *downward closure property*, which states that all the supersets of a non-frequent itemset are also non-frequent. This property is intuitively correct as it is based on the recurrence of an item in a database or the *support* measure. Recent advances in knowledge discovery have given rise to more sophisticated mining tasks. One of the emerging areas in this field is mining the high utility itemsets, where utility is a user-defined parameter, that can hold an aesthetic or a quantitative value. The information mined using the FPM approach is indicative of the most frequently occurring patterns in a dataset; however, it may not completely imply its usefulness in terms of the measures like profit. Example, for a set of sales data in a retail store over a week, while bread and butter can be the most frequently sold items that are produced as an output by FPM techniques, the items sold at highest profit might be rare that are not produced as an output by FPM. The high utility itemset mining addresses this challenge by extracting those patterns from a database that have utilities higher than a user-defined threshold. The mined patterns are referred to as high utility itemsets (HUIs). The utility measure for transaction datasets is defined as the profit made by selling an item which is the product of the profit per unit of the item and the quantity of that item within a transaction. Mining the high utility patterns alone leads to a large number of outputs being produced, which can easily overwhelm the decision-making process in an organisation. So, for the mining results to be suitable for real-life applications, it is essential to design algorithms that take into account both, support and utility measure. These mined outputs are therefore, representatives of those itemsets that occur atleast with a frequency corresponding to a minimum support value within a database.

Moreover, the real-life datasets often contain items that have negative utility values. This is because certain business decisions are strategically designed to sell some items at losses, which thereby hold a negative profit value. For example, in order to enhance the sale of a newly launched product, another product can be tagged along with it, to be sold for free or at a discounted price.

In this paper, we design an algorithm to effectively address the real-life data mining requirements. The main contribution of the proposed study is the design of an efficient algorithm for mining the frequently occurring HUIs from the databases with items holding negative utility values.

2 Problem Statement

Here, we introduce the problem of mining frequently occurring HUIs from the databases with items holding negative utility value, by using a supporting example. Consider a sample dataset consisting of three items and six transactions as given in Table 1. Every transaction consists of a unique identifier, *TID*, and the quantity of individual items p, q, r. The unit profit of each item is also provided. It can be observed from this table that item (q) holds a negative utility value.

Table 1 Transaction database

TID	p	q	r
TID_1	1	0	0
TID_2	4	0	0
TID_3	7	0	1
TID_4	2	2	0
TID_5	1	1	0
TID_6	0	1	10

Item	Unit profit
p	2
q	-3
r	7

The *utility* of an item is the product of the quantity and it's per unit profit. Here, the utility of the item (p) in TID_1 is *quantity* $(p, TID_1) \times profit\,(p) = 1 \times 2 = 2$ units. The *total utility* of an item is defined as the sum of its utilities across all transactions in the database. Therefore, the utility of item (p) in the database is $\sum_{i=1}^{6} util(p)_{TID(i)}$, which computes to 30. For utility threshold 25, the item (p) (with utility value 30) is a high utility item. The *support* of an item is defined as the count of the number of transactions in which the item is present. For the dataset in Table 1, the support of item (p) is 5 as it occurs in transactions $TID_1, TID_2 \ldots TID_5$. If the support threshold is set to 4, the item (p) would be considered as a frequent item. The item (q) holds a utility value of -12 and a support of 3, so, it is a low utility, low frequency item.

It may be noticed that even though the item (q) holds a negative utility value, it is still possible to have combinations of items with item (q), that qualify as having high utility. For example, consider the itemset (q, r) that holds a combined utility value of 73, which is higher than the specified minimum utility threshold of 25, and is therefore a high utility item.

Given this premises, the objective is to discover those combinations of the items that have utility value and support higher than the pre-defined threshold values for the utility and support, respectively.

3 Related Work

Several algorithms exist in the literature for FPM based on a given minimum support threshold, *minsup*. The most widely applicable algorithms for FPM are the apriori algorithm [1] and the FP tree [6] algorithm. The apriori algorithm [1] is based on the *downward closure property*, which utilises the anti-monotone trend of the frequent patterns and enables efficient pruning of the search space. It first scans the database and generates the candidates in a level by level fashion by combining items from the previous level. This algorithm simultaneously prunes the candidates to effectively mine the correct results.

The FP tree [6] algorithm stores the information regarding the frequency of occurrence of an itemset in a tree like structure and then explores the tree in a depth first search fashion to mine the high frequency items. These algorithms are however not suitable to mine the high utility itemsets, primarily because the high utility itemsets do not hold the downward closure property. The non-compliance of utility mining techniques with the downward closure property is justifiable as the utility of an item is dependant on both, the quantity and per unit profit. Therefore, the supersets of a non-high utility item may or may not have high utility.

In order to avoid the combinatorial explosion in the generation of candidate itemsets for HUIM, it is essential to establish an ordering between the patterns, which allows for pruning of non-promising candidates. A close resemblance to the downward closure property is introduced in the two-phase algorithm [11], called the transaction weighted utilisation (TWU)-based property. The property states that for an itemset X, if $TWU(X) < minutility$, then (X) and all its supersets are low utility itemsets, where the transaction weighted utilisation of (X) is defined as the total summation of transaction utilities of all those transactions in which the itemset (X) is present. Other techniques to efficiently organise the search space include using tree structures and utility lists [2, 7, 10]. The utility lists efficiently store the utility information of every item in form of transaction ID, utility value and the remaining utility of the transaction. HUI-miner [10] uses utility lists to find HUIs by recursively exploring the extensions of single itemsets. An improvement to the HUI-miner [10] is introduced in *FHM* [5] which uses an EUCS structure to store the TWU values of pair of itemsets and improves the pruning of non-potential candidates. A few recent studies make use of heuristics [12] and distributed computing [8] to mine HUIs.

Mining of HUIs from the databases with items holding the negative utility values is a complicated task, as the negative utility value can decrease the transaction weighted utilisation, that may result in incorrect pruning of the search space. This can underestimate or overestimate the utilities of items which can lead to erroneous results. There are only two algorithms in the literature that account for negative utility values. The authors in [3] discuss the *HUINIV* which is based on two-phase algorithm [11] and overestimates HUIs by considering only the positive values of the items. The second algorithm is *FHN* [9] that is based on the utility lists [10] and the EUCS structure proposed in *FHM* [5]. The FHN maintains separate records for positive and negative utility items and deploys the TWU-based pruning using only the positive utility values. However, for large input datasets, FHN produces a huge volume of HUIs which can undermine the usability of the generated results. To the best of our knowledge, no algorithm exists in the literature to address the problem of finding frequent HUIs from the databases where the items hold negative utility values. In this paper, we introduce the support-based mining with negative utility (SMNU) algorithm to efficiently mine the high utility items while taking into consideration the item support and negative utility items.

4 Proposed SMNU Algorithm

In this section, we introduce our proposed SMNU algorithm that is inspired by *FHN* [9]. The main procedure of SMNU scans the input database D, to compute the transaction weighted utilisation (TWU) and support of the single itemsets. As suggested in [9], only the itemsets with positive utility values are used for computing TWU (line: 9). SMNU takes into account the frequency of itemsets unlike the FHN. For every item in the database D, if the item is present in a transaction T_j, then its support is incremented (line: 5). An important point regarding the computation of the support is that the support measure of the items is incremented regardless of the fact if the item holds a negative or a positive utility value. This is because the support of an item should reflect the total frequency of its occurrence in D. After computation of support, transaction weighted utilisation of single items is computed by taking only the positive utility values into consideration. The utility lists of the single items are then formed (line: 15), and the search procedure (2) is called (line: 17), to recursively produce the candidate items.

Algorithm 1 SMNU main procedure

INPUT:
D: Database of size N
OUTPUT:
UL: Set of Utility Lists L_i

1: $TWU(i) = 0$
2: $Sup(i) = 0$
3: **for all** Single Items $i \in D$ **do**
4: **for all** $T_j = 1$ to N **do**
5: $Sup(i) \leftarrow Sup(i) + 1$
6: **if** $Utility(i) < 0$ **then**
7: $NI^* : NegativeItemsets \leftarrow (i)$
8: **else**
9: $TWU(i) = TWU(i) + TU(T_j)$
10: **end if**
11: $I^* \leftarrow (i)$
12: **end for**
13: **end for**
14: **for all** $i \in I^*$ **do**
15: Form $UtilityList\ UL_i$
16: **end for**
17: SEARCH($\emptyset$, I^*, $MinUtilThresh$, UL)

The search procedure (2) computes the sum of positive and negative utilities of each item and checks the sum against the minimum utility threshold (line: 4). Additionally, in order to ensure that only the frequent HUIs are produced as outputs, a check against the minimum support value $SThrsh$ is implemented (line: 4). If the input itemset passes these two checks, it is produced as a HUI (line: 5). Further, all the combinations of itemsets are explored sequentially only if the sum of utilities and remaining utilities for an itemset is higher that the minimum utility (line: 7). A noteworthy point is that the check against the minimum support value is implemented

Algorithm 2 Search procedure

INPUT:
I: ItemSet
ExtI: Extension ItemSets of *I*
MThrsh: Minimum Utility Threshold
SThrsh: Minimum Support Threshold
UL: Set of Utility Lists
OUTPUT:
HUIMs: Set of HUIs

```
1: for all I_x ∈ ExtI do
2:     SumUtil = SumUtilp + SumUtiln
3:     SumRUtil = SumRUtilD
4:     if SumUtil > MThrsh & Sup(i) > SThrsh then
5:         HUI ← I_x
6:     end if
7:     if SumUtil + SumRUtil > MThrsh & Sup(i) > SThrsh then
8:         ExtI_x ← ∅
9:         for all I_y ∈ ExtI do
10:            I_xy ← I_x ∪ I_y
11:            UL_xy ←Ext_x, Ext_y
12:            if UL_xy.iutil > MThrsh then
13:                ExtI_x ← ExtI_x ∪ I_xy
14:            end if
15:        end for
16:        SEARCH(I_x, ExtI_x, MThrsh, SThrsh, UL)
17:    end if
18: end for
```

here because, if an itemset is non-frequent, then all its supersets will also be non-frequent as per the *downward closure property*. This allows for pruning of the search space, as the extensions of the non-frequent itemsets need not be explored. The qualified candidates after the check (line: 7) are used to perform the join operation between the utility lists of the extensions (line: 9 to line: 15). The search procedure is called (line: 16) with the formed extensions to recursively explore all the valid itemsets in a depth first fashion.

The main contribution of SMNU is the computation of the support count of each itemset and new pruning conditions based on the support measure, which helps to minimise the search space and improves efficiency. In the next section, we perform several experiments to validate the performance of SMNU against the state-of-the-art algorithms.

5 Experimental Evaluation

SMNU is the first algorithm in the literature to mine HUIs based on a support measure from datasets where items can have negative utility values also. Therefore, to test the efficiency of our method, we carry out extensive experiments on various datasets that

Table 2 Statistics of datasets

Dataset	Transactions	No of items	Average length
Retail	88,162	16,470	10,30
Mushroom	8416	119	23
Chess	3196	75	37

include *retail*, *mushroom* and *chess*. The summary of characteristics of these datasets is provided in Table 2. These datasets are available in the SPMF library [4]. First, we compare the performance of SMNU and FHN, when a very small support threshold supplied to SMNU. Then, we vary the minimum support threshold and compare the performance of SMNU with FHN, for iterations by varying the minimum utility threshold. The comparison of candidate count and HUI count between SMNU and FHN is provided in Figs. 1 and 2, respectively. For the convenience of representation, the candidate count in Fig. 1 is represented in multiples of 100.

For the *retail* dataset, a minimum support of 0.01 is set, which means that only those itemsets should be produced as an output by SMNU, that occur in atleast 1% of the total transactions in the input dataset. For the *mushroom* and *chess* datasets, the minimum support threshold is set to 0.1, which is 10% of the total transactions. As it is evident from Fig. 1, for dense dataset (*retail*), SMNU reduces the candidate count by almost a factor of seven as compared to FHN. Also, as shown in Fig. 2, the total number of HUIs for *retail* dataset is about three times higher for FHN as compared to SMNU. These candidate itemsets and HUIs are reflective of the itemsets when the support threshold is set to 0.01. This implies that only one third of the HUIs in the entire database occur atleast in 10% of the transactions. The remaining HUIs as produced by FHN, occur in less than 10% of the transactions. In real-life applications, analysing such itemsets manually can consume a lot of irrelevant time and manpower for large datasets. Similar observations can be found for the dataset *mushroom* from Figs. 1 and 2, where the candidates and HUI count differ by almost a factor of two for a relatively higher support value of 0.1. For smaller datasets like *chess*, it can be observed that the difference between the two algorithms for candidate and HUI count is negligible. So, it can be established that the performance of SMNU is highly efficient for large and dense datasets.

The comparisons for memory and time requirements between SMNU and FHN is presented in Figs. 3 and 4, respectively. It can be observed from these figures that the difference between FHN and SMNU is significant for large and dense datasets like *retail* and *mushroom*.

The comparisons for memory, time and HUI count are shown in Figs. 5, 6 and 7 by varying the minimum support threshold from 0.1 to 0.7. Both the algorithms have been executed on the three datasets with varying minimum utility thresholds.

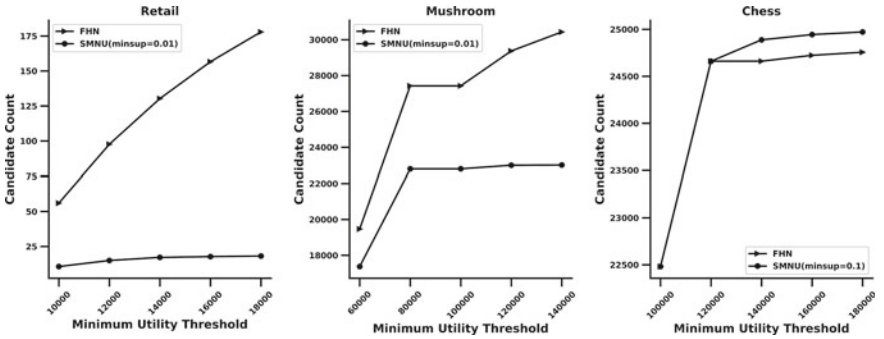

Fig. 1 Candidate count

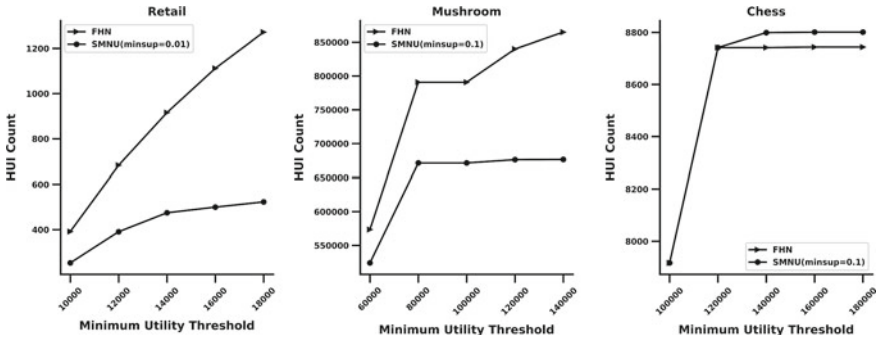

Fig. 2 High utility itemset count

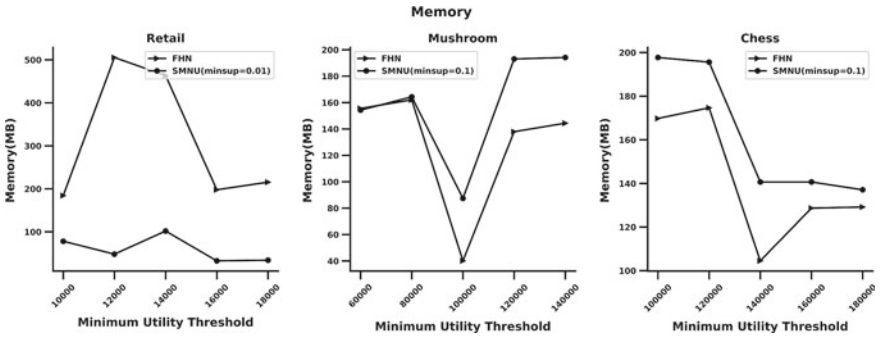

Fig. 3 Memory (MB)

It is evident from Fig. 5 that for the *retail* dataset, the SMNU outperforms the FHN algorithm in terms of memory, time and HUI count for all runs of the varying support values. Similar trends have been observed for the *mushroom* dataset as shown in Fig. 6. The results on *chess* dataset as shown in Fig. 7, are comparable for FHN and SMNU algorithms.

So, the SMNU guarantees an optimal count of HUIs to be produced as output and is also more scalable than the FHN algorithm for large and dense datasets.

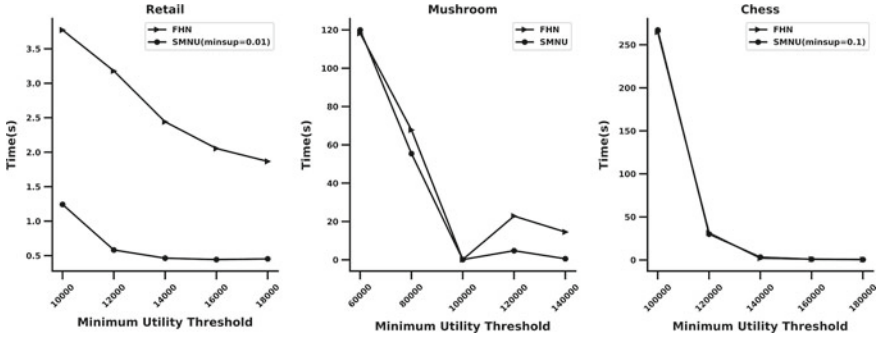

Fig. 4 Time (s)

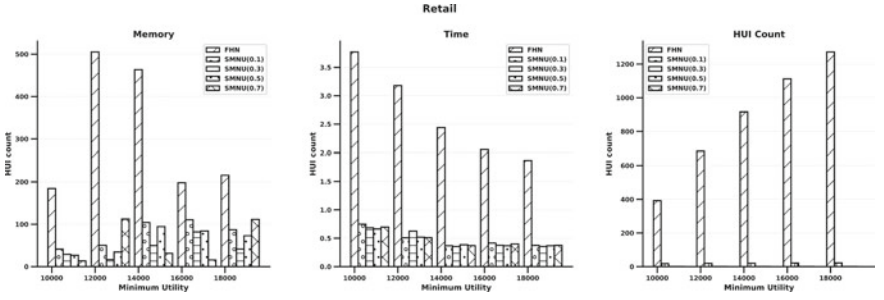

Fig. 5 Retail

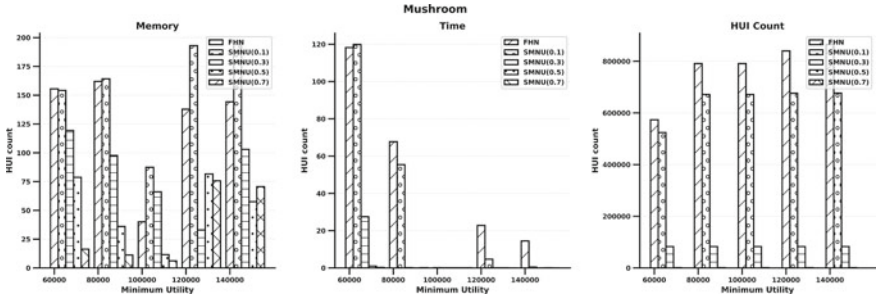

Fig. 6 Mushroom

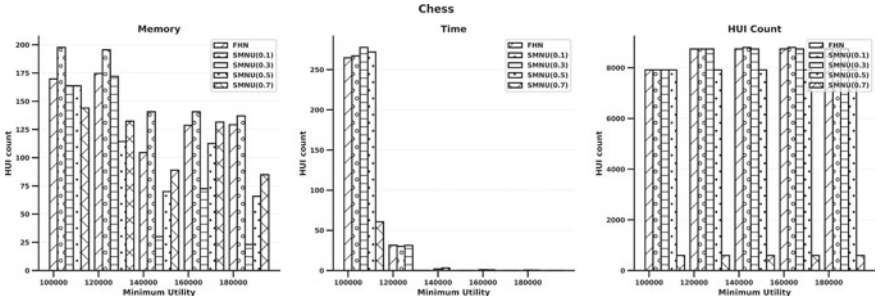

Fig. 7 Chess

6 Conclusion

In this paper, we have presented an algorithm, called SMNU, for support-based mining of HUIs from the databases with items that have negative utility values. The SMNU produces HUIs that can be directly utilised for real-life applications, unlike for most of the other mining algorithms where the support or negative utility values are not taken into consideration. The experimental results show that SMNU is more scalable and requires less execution time as well as memory for large datasets. However, for small and sparse datasets, the proposed approach has similar performance as the approaches that do not use support.

In future, we aim to propose optimisations to further enhance the performance of SMNU for large as well as small datasets. We also aim to design a novel method to generate association rules for the mined HUIs.

References

1. R. Agrawal, R. Srikant et al., Fast algorithms for mining association rules, in *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, vol. 1215 (Citeseer, 1994), pp. 487–499
2. Y. Baek, U. Yun, H. Kim, J. Kim, B. Vo, T. Truong, Z.-H. Deng, Approximate high utility itemset mining in noisy environments. Knowl.-Based Syst. **212**, 106596 (2021)
3. C.-J. Chu, V.S. Tseng, T. Liang, An efficient algorithm for mining high utility itemsets with negative item values in large databases. Appl. Math. Comput. **215**(2), 767–778 (2009)
4. P. Fournier-Viger, *SPMF: A Java Open-Source Data Mining Library*. Philippe-fournier-viger.com. (2021)
5. P. Fournier-Viger, C.-W. Wu, S. Zida, V.S. Tseng, FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning, in *International Symposium on Methodologies for Intelligent Systems* (Springer, 2014), pp. 83–92
6. J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation. ACM SIGMOD Rec. **29**(2), 1–12 (2000)
7. X. Han, X. Liu, J. Li, H. Gao, Efficient top-k high utility itemset mining on massive data. Inf. Sci. **557**, 382–406 (2021)

8. S. Kumar, K.K. Mohbey, High utility pattern mining distributed algorithm based on spark RDD, in *Computer Communication, Networking and IoT* (Springer, 2021), pp. 367–374
9. J.C.-W. Lin, P. Fournier-Viger, W. Gan, FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits. Knowl.-Based Syst. **111**, 283–298 (2016)
10. M. Liu, J. Qu, Mining high utility itemsets without candidate generation, in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (2012), pp. 55–64
11. Y. Liu, W.-K. Liao, A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, in *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (Springer, 2005), pp. 689–695
12. W. Song, C. Zheng, C. Huang, L. Liu, Heuristically mining the top-k high-utility itemsets with cross-entropy optimization. Appl. Intell. 1–16 (2021)