Chapter 2

Literature Review

In this Chapter, we present an overview of HUIs mining which contains an introduction to the research so far, HUIs mining, constraint-based HUIs mining, top-k HUIs mining, closed HUIs mining and HUIs mining with negative utility value.

2.1 High Utility Itemset Mining

The limitations of frequency-based itemset mining solutions motivated researchers to conceive a utility-based mining approach, which allows a user to express his or her perspectives concerning the usefulness of itemsets as utility values and then find itemsets with utility values not less than a specified minimum utility threshold. In HUIs mining, the meaning of utility refers to the interestingness, importance, or profitability of the itemsets. Studies in this area generally focus on mining HUIs from transactional datasets [21, 22, 25, 27, 28, 31].

In 2003, Chan et al. introduced first time the concept of HUIs [8]. Later on, some other algorithms have been proposed [9, 10]. However, these works did not support *downward closure property*. In 2005, Liu et al. proposed an algorithm named Two-Phase [22] which presents TWU based overestimation strategy. TWU follows *downward closure property* to speed up and prune the search space.

Yao et al. proposed two new algorithms named UMining and UMining_H [32]. These algorithms proposed two new pruning strategies named utility and support upper bound to reduce the cost of finding HUIs. With these pruning strategies, a k-itemset with an utility upper bound less than min_util can be pruned immediately without accessing the dataset to calculate its actual utility value. These algorithms also suffer from scalability issues due to its level-wise candidate generation, prune and test methodology. Comparing both of these algorithms, UMining is preferred over UMining_H, as UMining guarantees the discovery of all HUIs. The depth first search based approaches have many advantages over level-wise mining. In 2008, Li et al. also proposed two new algorithms named FUM and DCG+ [33]. These algorithms also find HUIs using level-wise mining. Where the author presented an strategy named IIDS to generate limited the number of candidates. IIDS identifies isolated items and ignores them in the process of candidate generation. To improve performance, IIDS can be applied to any two-phase based algorithm to reduce the candidates.

Lan et al. proposed projection-based (PB) mining algorithm to speeding up the runtime time and reduce the memory requirement [34]. PB based algorithm uses special indexing structure to efficiently mine HUIs. This indexing mechanism can imitate the traditional projection algorithms to achieve the aim of projecting sub-datasets for mining. This algorithm does not copy original dataset for each itemset to find HUIs, instead of that, they can directly extract through the indexing structure. Therefore, it consume less memory than that needed by directly copying the original dataset. The authors showed that PB algorithm is much efficient than Two-phase and CTU-PRO [35].

Ahmed et al. proposed an efficient method for mining HUIs from incremental datasets [11]. The author presented three tree structures to handle incremental dataset. First tree structure is IHUPL-Tree (Incremental HUP Lexicographic Tree), where tree has been arranged in lexicographic order according to its items. It is very easy and simple to construct and handle. It requires very small restructuring operation for incremental update of the datasets and hence this tree requires less memory. Second tree structure is the IHUPTF-Tree (IHUP Transaction Frequency Tree). It obtains a compact size by arranging itemsets in descending order according to their transaction frequency. Third tree structure is IHUPTWU-Tree (IHUP-Transaction-Weighted Utilization Tree). IHUPTWU-Tree arranges TWU values of itemsets in descending order. It reduces the execution time compared to previous methods. All three tree structures need maximum

two dataset scans. These tree structures are highly suitable for interactive mining and follow the property of *build once mine many*. Performance analysis shows that proposed tree structures are very scalable and efficient for incremental HUIs mining. All proposed tree structure algorithms outperform the existing algorithms in execution time as well as memory usage. Furthermore, all the tree structure algorithms are scalable for handling a large number of distinct items and transactions.

Tseng et al. proposed Utility Pattern Tree (UP-Tree) based algorithm named UP Growth [12]. UP-Growth tries to solve two major problems that are multiple dataset scans and the huge number of candidate itemsets. UP-Tree based algorithms generate candidates by only two scans of the dataset. In the first scan, it computes transaction utility (TU) as well as TWU of each item. UP-Growth prunes the low utility itemsets by using TWU pruning strategy. In the second scan, the transactions are inserted into UP-Tree. UP-Growth algorithm mines HUIs in three phases. First phase constructs of UP-Tree and makes it compact. UP-tree maintains the information of items and transactions. The second phase generates the candidate HUIs using UP-Tree. Third phase mines HUIs from the set of candidate HUIs. Though UP-Growth outperforms the state-of-the-art IHUP algorithm. Experimental results show that UP-Growth outperform IHUP algorithms particularly when the size of transactions is long. UP-Growth generates too many candidate itemsets. To reduce this limitation, Tseng et al. proposed UP Growth+ algorithm by pushing two strategies (DLU and DLN) into UP-Growth algorithm [21]. Using these strategies overestimated utilities of itemsets can be decreased and thus the number candidate itemsets can be further reduced. Jin et al. proposed an algorithm which is extension of UP-Growth algorithm. In this extended algorithm author incorporate the minimum support and mines frequent and HUIs [36].

Ahemd et al. presented a novel tree structure and pruning technique named HUC-tree and HUC-Prune respectively [23]. HUC-Prune solves too many candidate itemsets generation problems. HUC-Prune algorithm is completely independent of length of candidate itemsets . It solves the problem of level-wise candidate itemsets generation and test problem. As authors demonstrated, HUC-Prune requires a maximum of three dataset scans to calculate the resultant set of HUIs for the given minimum utility threshold. IHUP, HUC-Prune, UP-Growth and UP-Growth+ are tree based algorithms and mines HUIs without expensive candidate generation and test.

The tree based algorithms process a large number of candidate itemsets during their mining process. In most of the cases, many candidate itemsets are not HUIs and are discarded finally. To overcome this limitation Liu and Qu proposed a novel data structure, utility-list and developed an efficient one-phase, depth-first search based algorithm named HUI-Miner [24]. HUI-Miner mines HUIs without candidate generation, which avoids generation and test strategy. HUI-miner uses set-enumeration tree data-structure to store itemsets. A utility-list stores information such as Transaction T_{id}, utility of an itemset and utility of remaining itemsets. The utility-list structure allows directly computing the utility of generated itemsets in main memory without scanning the original dataset. HUI-Miner computes TWU and performs TWU based pruning. After that transactions are sorted ascending order according to their TWU. In the end, the remaining utility based pruning is used to find HUIs. The author demonstrated that HUI-Miner is superior to IHUP, UP-Growth+ and all other two-phase methods.

In 2012, Liu et al. proposed an efficient algorithm named d2HUP [37]. It is also one-phase that mines HUIs without generating candidate. d2HUP introduces two new pruning strategies, irrelevant item filtering and look-a-head strategy to prune the search space. d2HUP further uses these strategies to enhance the performance by recursive irrelevant item filtering with sparse data and by the look-a-head strategy with dense data. d2HUP and HUI-Miner algorithms were presented in 2012 by different set of authors. However, the key pruning strategies presented by both of these algorithms are conceptually equivalent though the underlying different data structures.

In 2015, Krishnamoorthy proposed an algorithm HUP-Miner [25] that is an extension of HUI-Miner algorithm. HUP-Miner presents a new data structure called partitioned utility list (PUL) that maintains utility information at the partition level. It introduces two new pruning strategies named PU-Prune and LA-Prune. These pruning strategies are very effective in pruning unpromising candidates, especially for sparse datasets. HUP-Miner is faster than the state-of-the-art algorithms, HUI-Miner, IHUP, UP-Growth and UP-Growth+. HUI-Miner and HUP-miner performs costly join operations to evaluate the utility of each itemset. To overcome this costly joins operations; Viger et al. proposed an algorithm FHM [27]. FHM proposes EUCP (Estimated Utility Co occurrence Pruning) that prunes itemsets without performing the join operation. EUCP relies on a new structure. FHM method is very similar to HUI-Miner, the only difference is in pruning strategy.

All the one-phase algorithms are more efficient concerning time and space compare to two-phase algorithms. The utility-lists provide not only utility information about itemsets but also important pruning information. The complexity of building of utility-list is not efficient concerning time and memory. So there are chances to improve the storage structure for efficient storage space and efficient algorithm to mine HUIs quickly.

All the one-phase algorithms discussed above generate the candidates and calculate the utility of an itemset by joining the utility-lists. To overcome this limitation, Zida et al. proposed an algorithm EFIM [28]. EFIM presents new techniques to speed up HUIs mining process. It also presents two new upper-bounds, SU-Prune and LU-Prune for pruning. Recently Srikumar Krishnamoorthy proposed an algorithm named HMiner [31]. HMiner uses compact utility-list and virtual hyperlink data structure to store the informations of the items. The author demonstrated that HMiner is thirty percent to three orders of magnitude faster than the state-of-the-art algorithms. It is the fastest HUIs mining algorithm at the present time among all one-phase algorithms. The definition of HUIs mining is as follows.

TABLE 2.1: A transaction dataset

T_{id}	Transaction
T_1	(A, 2) (B, 2) (D, 1) (E, 3)
T_2	(B, 1) (C, 5) (E, 1)
T_3	(B, 2) (C, 1) (D, 3) (E, 2)
T_4	(C, 2) (D, 1) (E, 3)
T_5	(A, 2)
T_6	(A, 2) (B, 1) (C, 4) (D, 2) (E, 1)
T_7	(B, 3) (C, 2) (E, 2)

TABLE 2.2: External utility value

Item	A	B	C	D	E
External Utility	2	3	1	4	1

Definition 2.1.1. (Transaction dataset). Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of distinct items. A set $X \subseteq I$ is called an itemset. $D = \{T_1, T_2, ..., T_n\}$ be a transaction dataset where each transaction is represented by $T_j \in D$ where n is the total number of transactions in the dataset D.

Consider the sample transactional dataset D which is given in TABLE 2.1. This dataset contains seven transactions $(T_1, T_2, ..., T_7)$. Transaction T_1 indicates that items A, B, D and E appear with quantity respectively $2, 2, 1$ and 3. TABLE 2.2 indicates the external utility of each item.

Definition 2.1.2. (Internal utility). Internal utility or purchase quantity for an item x is a positive number denoted by $IU(x, T_j)$.

For example, $IU(A, T_1) = 2$ and $IU(D, T_1) = 1$ as shown in TABLE 2.1.

Definition 2.1.3. (External utility). External utility (EU) of an item x denoted as $EU(x)$. Each item has their corresponding external utility.

For example, $EU(A) = 2$ as shown in TABLE 2.2.

Definition 2.1.4. (Utility of an item). The utility of an item x in a transaction T_j is denoted as $U(x, T_j)$ and is defined as $U(x, T_j) = IU(x, T_j) \times EU(x)$.

For example, $U(A, T_1) = IU(A, T_1) \times EU(A) = 2 \times 2 = 4$.

Definition 2.1.5. (Utility of an itemset in a transaction). The utility of an itemset X in a transaction T_j is denoted as $U(X, T_j)$ and is defined as $U(X, T_j) = \sum_{x \in X} U(x, T_j)$.

For example, $U(AD, T_1) = 4 + 4 = 8$.

Definition 2.1.6. (Utility of an itemset). The utility of an itemset X in D is denoted as $U(X)$ and is defined as $U(X) = \sum_{X \subseteq T_j \wedge T_j \in D} U(X, T_j)$.

For example, $U(AD) = U(AD, T_1) + U(AD, T_6) = 8 + 12 = 20$.

TABLE 2.3: Transaction Utility (TU)

T_{id}	Transaction	Purchase quantity (IU)	Utility (U)	TU
T_1	A, B, D, E	2, 2, 1, 3	4, 6, 4, 3	17
T_2	B, C, E	1, 5, 1	3, 5, 1	9
T_3	B, C, D, E,	2, 1, 3, 2	6, 1, 12, 2	21
T_4	C, D, E	2, 1, 3	2, 4, 3	9
T_5	A	2	4	4
T_6	A, B, C, D, E	2, 1, 4, 2, 1	4, 3, 4, 8, 1	20
T_7	B, C, E	3, 2, 2	9, 2, 2	13

Definition 2.1.7. (Transaction utility). The transaction utility of a transaction T_j is denoted as $TU(T_j)$ and is defined as $TU(T_j) = \sum_i^m U(x_i, T_j)$ where m is the number of items in transaction T_j.

For example, $TU(T_1) = U(A,T_1) + U(B,T_1) + U(D,T_1) + U(E,T_1) = 4+6+4+3 = 17$. TABLE 2.3 shows TU of all the transactions.

TABLE 2.4: TWU values of items

Item	A	B	C	D	E
TWU	41	80	72	67	89

Definition 2.1.8. (Transaction weighted utilization). The transaction weighted utilization of an itemset X is the sum of the transaction utility values of all the transactions containing X which is denoted as $TWU(X)$ and is defined as $TWU(X) = \sum_{X \subseteq T_j \wedge T_j \in D} TU(T_j)$.

For example, $TWU(A) = TU(A,T_1) + TU(A,T_5) + TU(A,T_6) = 17+4+20 = 41$. TWU of the items are shown in TABLE 2.4.

Definition 2.1.9. (Support of an itemset). The support of an itemset X is defined as the number of transactions containing X in D and is denoted as $support(X)$.

For example, support of an item B is 5. The support of an itemset $\{A,B,D,E\}$ is 2, it occurs in transaction T_1 and T_6.

Definition 2.1.10. (High utility Itemset). An itemset X is called high utility itemset if $U(X) \geq min\_util$. Otherwise, itemset X is a low-utility itemset.

While many HUIs mining algorithms have been proposed to discover HUIs. However, the main problem is that HUIs mining algorithms discover too many HUIs with lots of tiny itemsets. These huge tiny itemsets are less useful for taking the decision. Therefore, constraint HUIs play an important role to discover more relevant and actionable itemsets. Hence, we define the problem of mining HUIs with length constraints. Length constraint based FIM mining is proposed by [38] and utilized by [39, 40].

2.2 Constraint-based High Utility Itemset Mining

Constraints are essential for many data mining applications. In the process of data mining, some constraints may be given by users in order to find relevant rules or patterns and increase the execution efficiency. Constraint-based mining can greatly reduce the number of patterns, reduce search space and make decision making easier. Some studies [41, 42, 43, 44] have categorized constraints into several categories such monotone, anti-monotone, succinct, convertible monotone and convertible anti-monotone constraints. Han et al. published some studies about constraint-based mining [38, 45, 46]. They classified the constraints according to their applications, including item constraint, length constraint, super-pattern constraint (Model-based constraint), aggregate constraint, regular expression constraint, duration constraint and gap constraint.

While many HUIs mining algorithms have been proposed to discover HUIs. However, the main problem is that HUIs mining algorithms discover too many HUIs with lots of tiny itemsets. These huge tiny itemsets are less useful for taking the decision. Therefore, constraint-based HUIs can play an important role to discover more relevant and actionable itemsets. In order to target these issues, in 2016, Fournier-Viger et al. proposed an algorithm named FHM+ which mines HUIs with length constraints [47]. FHM+ is the first algorithm that uses the concept of length constraints in HUIs mining. It presents a novel concept named LUR (Length Upper-bound Reduction) to reduce the upper-bounds on the utility of itemsets using length constraints. LUR reduces the search space. The author demonstrated that the proposed algorithm reduces the number of itemsets compare to the state-of-the-art algorithm FHM.

Length based constraints have been proposed in some FIM and sequential itemset mining studies such as [38, 45, 46]. The use and applications of length constraint have been proposed in [39, 40].

2.3 Top-k High Utility Itemset Mining

One of the limitations of HUIs mining is the assignment of *min_util* threshold which is very tough task for the end users. The user does not know which *min_util* threshold utility value is good for mining the required or actionable HUIs. If *min_util* threshold value is too low, then a very high HUIs will come out, hence analyzing of such a huge itemsets is not an easy and is not usable. If *min_util* threshold value is too high, then a very few HUIs come out to be used. So choosing the appropriate *min_util* threshold value is very crucial task. To address these issues, top-k HUIs mining was proposed because specifying the value of *k* is easy instead of specifying *min_util* threshold. And the top-k HUIs are also very useful in many domains.

Recently, Several algorithms have been proposed to mine top-k HUIs [14, 48, 49, 50, 51, 52, 53, 54]. They follow the same phenomena. Initially, the internal *min_util* threshold is set to one or zero. In the next step, the procedures automatically raise the internal threshold using some strategies. Important things about internal threshold raising strategies are that algorithms do not miss any top-k HUIs. And often some strategies are introduced to prune the search space. At last, the algorithms terminate and find top-k HUIs.

In the next chapter, we provide an overview of the key techniques used by top-k HUIs mining algorithms and explain their advantages and limitations. These techniques are very important as they have inspired numerous researchers in the field of utility mining.

2.3.1 Two-Phase Algorithms

Several algorithms have been proposed in recent years for mining top-k HUIs mining. TABLE 2.5 shows the overview of the state-of-the-art two-phase algorithms. The first and most popular algorithm for mining top-k HUIs is TKU [48] which is an extension of UP-Growth algorithm [12, 21]. TKU follows two-phase model and inherits their useful properties. In the first phase, potential top-k HUIs (PKHUIs) are generated and the algorithm uses four strategies (PE, MD, NU and MC) to raise the internal *min_util* threshold and prune the search space. In the second phase, top-k HUIs are identified from the set of PKHUIs that are discovered in first-phase. TKU uses fifth strategies (SE)

TABLE 2.5: An overview of two-phase top-k High utility itemset mining algorithms

Algorithm	**Author**	**Data structure**	**Dataset**	**Mining**	**Pruning strategy**	**State-of-the-art algorithms**	**Utility value**	**Base algorithm**
TKU, 2016[48]	Vincent S. Tseng et al.	UP-tree	Transactional	Top-k HUIs	PE, NU, MD, MC and SE	UP-Growth[12] and HUI-Miner[24]	Positive only	UP-Growth
T-HUDS, 2014[14]	Morteza Zihayat et al.	HUDS-tree (FP-tree like)	Data Stream	Top-k HUIs	PerfixUtil	TKU and HUMPS[55]	Positive only	UP-Growth
REPT, 2015[49]	Heungmo Ryang et al.	UP-tree	Transactional	Top-k HUIs	PUD, RIU, RSD and SEP	TKU, UP-Growth, UP-Growth+[21] and MU-Growth[56]	Positive only	TKU
TUS, 2013[50]	Junfu Yin et al.	TUSlist	Transactional	Top-k High utility sequential itemsets	Pre-insertion, Sorting concatenation order, Keep refreshing the blacklist	TUSNaive[50] (self)	Positive only	.....

to sort the candidate itemsets and raise the internal threshold. To improve the performance of TKU, Ryang and Yun proposed an algorithm, called REPT [49]. REPT also relies on the UP-Tree structure. REPT improves the performance of TKU by applying additional strategies that raise the border min_util rapidly. REPT develops four strategies named PUD, RIU, RSD and SEP to reduce the search space by raising the internal min_util. In the first phase, PKHUIs are generated. Then the algorithm uses two strategies (NU and MC) which are utilized from TKU algorithm. Three proposed strategies (PUD, RIU and RSD) are also used to raise the internal utility threshold and prune the search space. In the second phase, REPT uses SEP strategy to sort the candidate itemsets and raise the internal threshold. One of the limitations of REPT is that user has to set parameter N to control the effectiveness of RSD strategy. It is very tough for end users to choose an appropriate value for N and the choice of N greatly influences the performance of REPT. But still, TKU and REPT generate a large number of candidates because they follow the two-phase model. They also scan the dataset repeatedly to obtain the exact utility of candidate itemsets and mine the actual top-k HUIs.

Zihayat et al. proposed an algorithm named T-HUDS for mining top-k HUIs over data streams [14]. T-HUDS proposed a novel over-estimate utility model called PrefixUtil model. This model follows compressed FP-tree like data structure, HUDS-tree and it

TABLE 2.6: An overview of one-phase top-k High utility itemset mining algorithms

Algorithm	Author	Data structure	Dataset	Mining	Pruning strategy	State-of-the-art algorithms	Utility value	Base algorithm
TKO, 2016[53]	Vincent S. Tseng et al.	utility list	Transactional	Top-k HUIs	RUC,RUZ and EPB	UP-Growth and HUI-Miner	Positive only	HUI-Miner
kHMC, 2016[51]	Quang-Huy Duong et al.	utility-list	Transactional	Top-k HUIs	RIU, CUD and COV	TKO and REPT	Positive only	FHM[27]
TKUL-Miner, 2016[52]	Serin Lee et al.	utility-list	Transactional	Top-k HUIs	FSD, RUZ and FCU	TKU, UPGrwoth+, REPT	Positive only	FHM
KOSHU, 2017[54]	Thu-Lan Dam et al.	utility list	Transactional	Top-k on-shelf HUIs	EMPRP, PUP and CE2P	FOSHU[57]	Positive and negative	FOSHU, FHM

has two auxiliary lists, maxUtilList and MIUList. These lists are designed to store the information that is needed for computing PrefixUtil and for initializing and dynamically adjusting the internal threshold. Yin et al. have proposed a new algorithm named TUS for mining top-k high utility sequential patterns [50]. It introduced a pre-insertion and a sorting strategy to raise the internal utility threshold. To improve the performance of TUS, three effective strategies are introduced; two for raising min_util threshold and one for reducing the search space.

2.3.2 One-Phase Algorithms

In order to overcome the limitations of two-phase algorithms, recently one-phase algorithm has been introduced. Tseng et al. proposed an algorithm named TKO to mine top-k HUIs [53] which outperforms TKU and REPT. TKO utilizes the utility-list structure of the HUI-Miner. It presents the novel pruning strategies RUC, RUZ and EPB to improve the performance. TABLE 2.6 shows the overview of the state-of-the-art two-phase algorithms.

kHMC is another single-phase algorithm proposed by Duong et al. which relies on the utility-list structure [51]. kHMC is an extension of the FHM algorithm [27]. It employs two pruning strategies called EUCPT and Transitive Extension Pruning strategy (TEP). EUCPT avoids the join operations for calculating the utilities of itemsets as in FHM.

The TEP strategy reduces the search space using a novel upper-bound on the utilities of itemsets. kHMC introduces three strategies named RIU, CUD and COV to raise internal min_util effectively. The COV strategy introduces a novel concept of coverage. The concept of coverage can be employed to prune the search space or to raise the threshold in top-k HUIs mining. Moreover, kHMC proposes a new pruning strategy named TEP for reducing the search space. Another utility-list and the EUCS based top-k HUIM algorithm is TKUL-Miner [52]. It utilizes several strategies called FSD, RUZ and FCU to raise min_util rapidly and prune the search space effectively.

Recently, an on-shelf based top-k HUIs mining algorithm is proposed KOSHU [54]. It introduced three pruning strategies to speed-up the execution named EMPRP, PUP and CE2P. KOSHU is a top-k version on the FOSHU algorithm [57].

In this survey, we present an in-depth analysis of two-phase and one-phase of algorithms which made significant contribution to improving the efficiency of top-k HUIs mining.

2.4 Closed High Utility Itemset Mining

Traditional HUIs mining algorithms mine lots of repeated HUIs which degrade the performance. These algorithms may have good performance when min_util threshold is high and itemset space is sparse. However, when min_util drops low, the number of HUIs goes up dramatically and the performance of these algorithms deteriorates quickly because of the generation of a huge number of itemsets. Moreover, the effectiveness of the mining of the complete set degrades because it generates numerous redundant itemsets. A simple example is that in a dataset having only one transaction of length 100, it will generate 2^{100-1} HUIs if the absolute min_util is set to very low.

To overcome the drawbacks of traditional HUIs algorithms, closed HUIs has been introduced. Closed HUIs mining algorithms mine only those HUIs having no proper superset with the same support. Mining closed HUIs, as shown in [58], can lead to orders of magnitude smaller result set (than mining HUIs) while retaining the completeness, i.e., from this concise result set, it is straightforward to generate all HUIs.

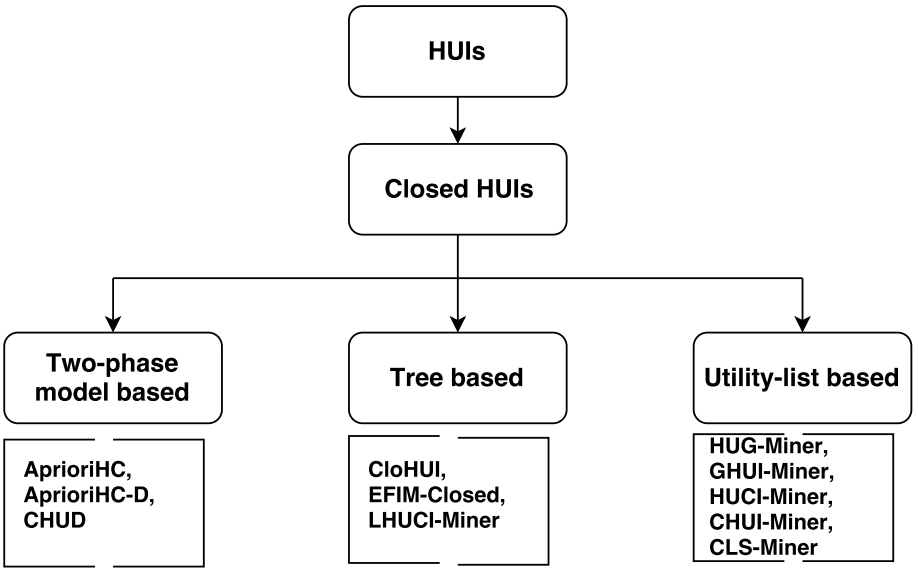

FIGURE 2.1: Taxonomy of closed high utility itemsets mining according to mining approaches

Closed HUIs mining can be classified into three groups as shown in FIGURE 2.1, namely, level-wise (join-based or Apriori-based), tree-based (pattern growth) and utility-list based (vertical).

2.4.1 Level-wise Mining Algorithms

CHUD: In 2011, Wu et al. proposed an algorithm named CHUD (Closed+ High Utility itemset Discovery) [59]. CHUD is the first work that incorporates closed concept into HUIs mining. It presents a compact representation. The author demonstrated that it reduces the number of itemsets by several orders of magnitude (up to 800 times), especially for dense datasets. It is extension of Two-phase algorithm. It presents a lossless representation by using a new structure named utility unit array. CHUD presents three pruning strategies (REG, RML and DCM) which greatly improve the performance. CHUD presents a top-down approach DAHU (Derive All High Utility itemsets) for efficiently recovering all HUIs from the Closed HUIs.

AprioriHC & AprioriHC-D: In 2015, Tseng et al. proposed three efficient algorithms AprioriHC (Apriori-based algorithm for mining High utility Closed+ itemsets), AprioriHC-D (AprioriHC algorithm with Discarding unpromising and isolated items)

and CHUD [58]. This chapter is a continuation and extension of earlier work [59]. AprioriHC and AprioriHC-D are also two-phase based algorithms like CHUD. AprioriHC and AprioriHC-D perform a breadth-first search for mining closed HUIs from horizontal dataset, while CHUD performs a depth-first search for mining closed HUIs from vertical dataset. AprioriHC is a HUIs version of well known FIM algorithm Apriori [1]. AprioriHC-D is an improved version of AprioriHC which includes two strategies (DGU [21] and IIDS[33]) to prune the search space. AprioriHC-D performs better than AprioriHC because of these pruning strategies. AprioriCH and AprioriHC-D can not perform well on dense datasets when min_util is low because of its Apriori like structure. CHUD performs better than AprioriHC and AprioriHC-D. However, AprioriHC and AprioriHC-D are easier to implement and extend for more applications.

2.4.2 Tree-based Mining Algorithms

CloHUI: In 2016, Guo et al. proposed an algorithm named CloHUI [60]. CloHUI works like UP-Growth algorithm. It presents a new data structure HUITWU-Tree (High Utility Itemsets tree which arranges items according to Transaction Weighted Utility of single itemsets) to store the information of itemsets. CloHUI needs two dataset scans to construct proposed HUITWU-Tree. In first dataset scan, it discovers TWU of all the items. In second dataset scan, it discovers closed HUIs. The author demonstrated that CloHUIs is an order of magnitude faster than the state-of-the-art algorithm CHUD.

EFIM-Closed: In 2016, Fournier-Viger et al. proposed an algorithm named EFIM-Closed [61]. It is an extension version of HUIs mining algorithm EFIM [28]. It introduces two new techniques (Transaction merging and dataset projection) to reduce the dataset scanning cost. It presents two tree-based pruning strategies (local-utility and subtree-utility). It also presents a jumping closure and backward closure checking to prune non-closed HUIs. The author demonstrated that EFIM-Closed is up to 71 times faster and consumes up to 18 times less memory than the state-of-the-art CHUD algorithm.

LHUCI-Miner: In 2017, Mai et al. present an algorithm named LHUCI-Miner [62]. LHUCI-Miner presents a lattice-based approach to mine closed HUIs and their generations. It works like HUCI-Miner [63].

2.4.3 Utility-list based Mining Algorithms

HUG-Miner & GHUI-Miner: In 2014, Fournier-Viger et al. proposed two novel algorithms, HUG-Miner (High Utility Generators) and GHUI-Miner (Generator of High Utility Itemsets) to mine closed HUIs and their *generators*[1] [64]. HUG-Miner mines only HUGs and does not consider LUGs (Low Utility Generators) that are not HUIs. Hence, HUG-Miner is up to two orders of magnitude faster than the state-of-the-art FHM and CHUD algorithms for closed HUIs mining and HUIs mining. GHUI-Miner discovers complete set of GHUIs and spends more time to consider LUGs. GHUI-Miner firstly mines closed HUIs by using a closed HUIs mining algorithm CHUD and then mines generators using a modified FHM algorithm. HUG-Miner is over 100 times faster than GHUI-Miner but misses GHUIs that are LUGs.

CLS-Miner: In 2017, DAM et al. proposed an algorithm named CLS-Miner [Thu-Lan DAM]. CLS-Miner utilizes the utility-list structure to directly compute the utilities of itemsets without producing candidates. It presents three pruning strategies Chain-EUCP (Chain-Estimated Utility Co-occurrence Pruning), LBP (Lower Branch Pruning) and Coverage. Chain-EUCP calculates utility of pairs of items to determine if an itemset and its extensions should be pruned alike FHM [27]. Chain-EUCP works on ECUS structure proposed by FHM. LBP reduces the search space by pruning low utility transitive extensions of itemsets. Chain-EUCP and LBP eliminate candidates without fully constructing their utility-lists. Coverage prunes low utility itemsets and calculates the closure of itemsets quickly. These three pruning strategies can greatly reduce the number of join operations for constructing utility-lists. CLS-Miner presents pre-check based method to quickly compute closure. The author demonstrated that Chain-EUCP and the LBP strategies could prune up to 93% of candidates and pre-check method can reduce the total runtime by up to 50% compare to the current state-of-the-art CHUD [58] and CHUI-Miner [65].

CHUI-Miner: In 2015, Wu et al. proposed an algorithm named CHUI-Miner [65]. CHUI-Miner is an extension and closed HUIs version of HUI-Miner algorithm. Hence, it does not generate candidate itemsets to mine CHUIs. It uses the extended utility-list structure called EU-List (Extended Utility-List). The author demonstrated that

[1] generators itemsets are the set of HUIs that have no subsets having the same support.

TABLE 2.7: An overview of closed high utility itemsets mining algorithms

Algorithm	Year	Author	Dataset scanning	Data structure	Dataset	Mining	Pruning strategy	State-of-the-art algorithms	Base algorithm
Level-wise mining algorithms									
CHUD[59]	2011	Wu et al.	Multiple times	...	Transactional	CHUIs, HUIs	TWU	UP-Growth [21]	Two-Phase [22]
AprioriHC, AprioriHC-D & CHUD (extended) [58]	2015	Tseng et al.	Multiple times	...	Transactional	CHUIs, HUIs	TWU, DGU[a] & IIDS[b]	Two-Phase & UP-Growth	Two-Phase
Tree-based mining algorithms									
CloHUI[60]	2016	Guo et al.	Two times	HUITWU-Tree[c]	Transactional	CHUIs	TWU	CHUD	UP-Growth
EFIM-Closed [61]	2016	Fournier-Viger et al.	One time only	UP-Tree	Transactional	CHUIs	Local utility, sub-tree utility & Backward extension pruning	CHUD	EFIM[28]
LHUCI-Miner [62]	2017	Mai et al.	One time only	Lattice	Transactional	CHUIs	TWU	HUCI-Miner[63].	
Utility-list based mining algorithms									
HUG-Miner & GHUI-Miner [64]	2014	Fournier-Viger et al.	Two times	Utility-list	Transactional	CHUIs, GHUIs & HUGs	TWU & EUCP	FHM & CHUD	FHM[27]
HUCI-Miner [63]	2014	Sahoo et al.	...	Utility-list	Transactional	CHUIs	TWU, Remaining utility based & EUCP	FHM	FHM
CHUI-Miner [65]	2015	Wu et al.	Two times	Extended Utility-List	Transactional	CHUIs	TWU & Remaining utility based	CHUD & HUI-Miner	HUI-Miner [24]
CLS-Miner [Thu-Lan DAM]	–	DAM et al.	One time only	Utility-list	Transactional	CHUIs	Chain-EUCP, LBP[d] & Coverage	CHUD & CHUI-Miner	FHM

[a] Discarding Global Unpromising items
[b] Isolated Items Discarding Strategy
[c] High Utility Itemsets Tree which arranges items according to Transaction Weighted Utility of single itemsets
[d] Lower Branch Pruning

CHUI-Miner has good scalability even on large datasets and it uses less memory than the state-of-the-art algorithm CHUD, especially for dense dataset.

HUCI-Miner: In 2014, Sahoo et al. proposed an algorithm named HUCI-Miner [63]. HUCI-Miner mines CHUIs and their generators. It is an extension of FHM. It initially mines HUIs using FHM algorithm and then mines CHUIs and generators among HUIs.

2.4.4 Discussion

The previous section reviewed three types of closed HUIs mining algorithms; level-wise algorithms based on the Two-phase model [22], tree-based algorithms based on the UP-tree [12] and utility-list based algorithm following vertical data structure like Eclat [66]. TABLE 2.7 provides a summary of the characteristics of the algorithms discussed in previous section. Level-wise algorithms work like Apriori algorithm. Hence, these algorithms scan the dataset multiple time and generate too many candidate itemsets. Tree-based algorithms improve the mining process by scanning the dataset only twice. Tree-based algorithms generate many candidate itemsets but less than level-wise algorithms. Utility-list based algorithms scan the dataset only once and store all the information into utility-lists. These algorithms suffer to utility-list joining problem when distinct number of itemsets are high. But utility-list based algorithms still perform better than level-wise and tree-based algorithms. TABLE 2.7 shows the data structure, pruning strategies and some other important characteristics of closed HUIs mining algorithms.

2.5 High Utility Itemset Mining with Negative Utility Value

HUIs mining is an important research area of data mining. HUIs mining gained the immense importance because of huge application areas. HUIs with negative item value recently received much attention from the decision making community. It is quite useful in the marketing and retail communities as well as other more diverse fields. The definitions of HUIs mining with negative utility value are as follows.

A transactional dataset D is a set of transactions where each transaction contains a set of items and is associated with a unique transaction identity T_{id}. Let $I = \{i_1, i_2, \ldots, i_n\}$ be the complete set of distinct items appearing in D. An itemset X is a non-empty subset of I and is called a $k-$itemset if it contains k items. An itemset $x_1, \ldots, x_k$ is also denoted as $x_1 \ldots x_k$. The number of transactions in D containing itemset X is called the support of itemset X, denoted as $support(X)$. For example, let us consider the data presented in TABLE 2.8 and TABLE 2.9. TABLE 2.8 presents a dataset comprising seven transactions $T = \{T_1, \ldots, T_7\}$ and five items $I = \{A, B, C, D, E\}$. The profit (external utility) of these items is presented in TABLE 2.9.

TABLE 2.8: Transactional dataset

T_{id}	Transaction
T_1	(A, 2) (B, 2) (D, 1) (E, 3)
T_2	(B, 1) (C, 5) (E, 1)
T_3	(B, 2) (C, 1) (D, 3) (E, 2)
T_4	(C, 2) (D, 1) (E, 3)
T_5	(A, 2)
T_6	(A, 2) (B, 1) (C, 4) (D, 2) (E, 1)
T_7	(B, 3) (C, 2) (E, 2)

TABLE 2.9: External utility of items

Item	A	B	C	D	E
External Utility	2	-3	1	4	1

Definition 2.5.1. (Internal utility). Internal utility or purchase quantity for an item x is a positive number denoted by $IU(x, T_j)$.

For example, $IU(A, T_1) = 2$ and $IU(D, T_1) = 1$ as shown in TABLE 2.8.

Definition 2.5.2. (External utility). External utility (EU) of an item x denoted as $EU(x)$. Each item has their corresponding external utility.

For example, $EU(A) = 2$ as shown in TABLE 2.9. If the EU of an item x is a positive value, i.e., $EU(x) > 0$, the item has positive item profit or positive utility. Otherwise, the item x has negative utility, if $EU(x) < 0$.

Definition 2.5.3. (Utility of an item). The utility of an item x in a transaction T_j is denoted as $U(x, T_j)$ and is defined as $U(x, T_j) = IU(x, T_j) \times EU(x)$.

For example, utility of an item A is $U(A,T_1) = IU(A,T_1) \times EU(A) = 2 \times 2 = 4$.

Definition 2.5.4. (Utility of an itemset in a transaction). The utility of an itemset X in a transaction T_j is denoted as $U(X,T_j)$ and is defined as $U(X,T_j) = \sum_{x \in X} U(x,T_j)$.

For example, utility of an itemset $\{A,D\}$ is $U(\{A,D\},T_1) = 4+4 = 8$.

Definition 2.5.5. (Utility of an itemset). The utility of an itemset X in D is denoted as $U(X)$ and is defined as $U(X) = \sum_{X \subseteq T_j \wedge T_j \in D} U(X,T_j)$.

For example, $U(AD) = U(AD,T_1) + U(AD,T_6) = 8+12 = 20$.

Definition 2.5.6. (Transaction utility). The transaction utility of a transaction T_j is denoted as $TU(T_j)$ and is defined as $TU(T_j) = \sum_i^m U(x_i,T_j)$ where m is the number of items in transaction T_j.

For example, $TU(T_1) = U(A,T_1) + U(B,T_1) + U(D,T_1) + U(E,T_1) = 4+4+3+(-6) = 5$. The TU values for the running example is shown in third column in TABLE 2.10.

Definition 2.5.7. (Transaction weighted utilization). The transaction weighted utilization of an itemset X is the sum of the transaction utilities of all the transactions containing X which is denoted as $TWU(X)$ and is defined as $TWU(X) = \sum_{X \subseteq T_j \wedge T_j \in D} TU(T_j)$.

For example, $TWU(A) = TU(A,T_1) + TU(A,T_5) + TU(A,T_6) = 5+4+14 = 23$. TWU for B, C, D and E is 26, 30, 37 and 35 respectively.

Definition 2.5.8. (High utility Itemset). An itemset X is called high utility itemset if $U(X) \geq min\_util$. Otherwise, itemset X is a low-utility itemset.

For example, HUIs for running example is shown in TABLE 2.13.

2.5.1 Properties to Handle Negative Utility Items

To handle the negative utility, the proposed algorithm follows several properties:

Property 2.5.1. (Relationship between positive utilities and negative utilities within an itemset). For an itemset X, $putil(X)$ and $nutil(X)$ denote the sum of positive utilities and sum of negative utility values of an itemset X respectively, such that $U(X) = putil(X) +$

$nutil(X)$. Hence, the relationship holds: $putil(X) \geq U(X) \geq nutil(X)$ where $U(X)$ contains the actual utility of an itemset X.

Rationale. Positive utility items can only increase utility of an itemset X, on the contrary, the negative utility items can only decrease the utility of an itemset X. Hence, the property holds $putil(X) \geq U(X) \geq nutil(X)$.

Property 2.5.2. (Upper bound using positive utilities). The upper bound utility of itemset X is $U(X) = putil(X)$ because $putil(X)$ is greater than actual utility.

Rationale. Since $U(X) - nutil(X) = putil(X)$, the negative utility of itemsets only can decrease the utility of an itemsets X. Hence, $U(X)$ or $nutil(X)$ cannot be used to overestimate the utility of an itemsets X. As in *Property 2.5.1* and *Property 2.5.2*, $putil(X)$ can only be used to overestimate the utility of an itemsets X.

The state-of-the-art HUIs algorithms only can handle positive utility values and can utilize TWU based *Property 2.5.3* to prune the search space. The positive utility mining algorithms cannot be directly applied to handle negative utility mining. To handle the problem of mining HUIs with negative utility, HUINIV-Mine [29] utilizes redefined transaction utility (RTU) and redefined TWU as described follows.

TABLE 2.10: Redefined Transaction Utility

T_{id}	Transaction	TU	RTU
T_1	A,B,D,E	5	11
T_2	B,C,E	3	6
T_3	B,C,D,E	9	15
T_4	C,D,E	9	9
T_5	A	4	4
T_6	A,B,C,D,E	14	17
T_7	B,C,E	-5	4

Definition 2.5.9. (Redefined transaction utility). The redefined transaction utility is denoted by $RTU(T_j)$ for transaction T_j and is computed as $RTU(T_j) = \sum_i^m U(x_i, T_j)$ in which m is the number of items in T_j transaction. To calculate RTU, items must be sorted according to descending order to their utility values.

For the running example RTU of T_1 is as $RTU(T_1)$= $U(A,T_1)$ + $U(D,T_1)$ + $U(E,T_1)$ + $U(B,T_1) = 4+4+3+(-6) = 11$ because we calculate RTU by only adding the positive

items. $RTU(X) \geq TU(X)$, because, TU is the summation of all items in a transaction without any deletion and summation of negative items where RTU is the summation of remaining items and summation with only positive items. TABLE 2.10 shows the TU and RTU value of each transaction.

TABLE 2.11: $RTWU$ value of each item

Item	A	B	C	D	E
RTWU	32	53	51	52	62

Definition 2.5.10. (Redefined transaction weighted utility). The redefined transaction weighted utility ($RTWU$) of an itemset X is defined as $RTWU(X) = \sum_{X \subseteq T_j \in D} RTU(T_j)$

For example, $RTWU$ value of each item is shown in TABLE 2.11.

Property 2.5.3. (Pruning using $RTWU$). The $RTWU$ downward closure property states that any superset of a low $RTWU$ itemset is low utility. For an itemset X, if the $RTWU(X) < min\_util$ then X is not a HUIs and all the supersets of itemset X are also not HUIs. The detailed proof of this property can be found in [29].

Property 2.5.4. (Relationship between $RTWU$ and TWU). The $RTWU$ based overestimation has larger utility value compare to TWU. For an itemset X, the relationship between these upper-bounds is $RTWU(X) \geq TWU(X)$.

Proof: For the itemset X, we can observe that $RTU(X) \geq TU(X)$. Hence, $RTWU(X) \geq TWU(X)$ where $RTWU(X) = \sum_{X \subseteq T_j \in D} RTU(T_j)$ and $TWU(X) = \sum_{X \subseteq T_j \in D} TU(T_j)$.

TABLE 2.12: Sorted items according to $\succ$ total order

T_{id}	Transaction	Utility (U)	RTU
T_1	A, D, E, B	4, 4, 3, -6	11
T_2	C, E, B	5, 1, -3	6
T_3	C, D, E, B	1, 12, 2, -6	15
T_4	C, D, E	2, 4, 3	9
T_6	A, C, D, E, B	4, 4, 8, 1, -3	17
T_7	C, E, B	2, 2, -9	4

Definition 2.5.11. (Ordering of items). The items in the transactions are sorted according to the $\succ$ total order as $RTWU$ ascending order. For the running example, the sorted items are as $A \succ C \succ D \succ E \succ B$. The negative items are always follow the positive items. For the sample dataset, the ordered set of items are provided in TABLE 2.12.

TABLE 2.13: HUIs of the running example

Itemset	Utility	*Itemset*	Utility
$\{A,C,D\}$	16	$\{C,D,E\}$	37
$\{A,C,D,E\}$	17	$\{C,D,E,B\}$	19
$\{A,D\}$	20	$\{C,E\}$	23
$\{A,D,E\}$	24	$\{D\}$	28
$\{A,D,E,B\}$	15	$\{D,E\}$	37
$\{C,D\}$	31	$\{D,E,B\}$	15
$\{C,D,B\}$	16	$--$	–

Definition 2.5.12. (Extension of an item). Let the itemset be α and set of items that used to extend the itemset α is denoted as $E(\alpha)$ and is defined as $E(\alpha) = \{x \mid x \in I \wedge x \succ i, \forall i \in \alpha\}$.

Definition 2.5.13. (Extension of an itemset). Let the itemset be α and Y is an extension of α that appears in a sub-tree of α in the set-enumeration tree. If $Y = \alpha \cup \{X\}$ for an itemset $X \in 2^{E(\alpha)}$.

Here, Y is a single-item extension of α that is a child of α in the set-enumeration tree. If $Y = \alpha \cup \{X\}$ for an item $X \in E(\alpha)$.

For example, $\alpha = \{C\}$ and hence, the set $E(\alpha)$ is $\{D,E\}$. And single-item extensions of α are $\{C,D\}$, $\{C,E\}$ and $\{D,E\}$. The itemsets extensions of α is $\{C,D,E\}$.

Property 2.5.5. (Extension of negative item). Let the itemset be α which can be extended to itemset X, where $Y = \alpha \cup \{X\}$ and X is a set of items with negative utility.

Rationale. $\alpha \cup \{X\}$ only occurs in less than or equal to the number of transactions containing itemset α. Extensions of itemset α with positive utility item may be less than or equal to or greater than the utility of itemsets α. But extensions of itemset X with negative utility item always decrease the utility of itemsets as proposed by the *Property 2.5.1*. Hence, from the above properties, if an itemset $U(\alpha) > min\_util$ then the negative utility itemset X is added to itemset α. The utility of extended itemset is still greater than or equal to min_util then the itemset is HUIs.

This section categories HUIs mining algorithms into three parts as shown in FIGURE 2.2: level-wise (Apriori-based), tree-based (UP-tree like) and utility-list (vertical dataset format) based algorithms.

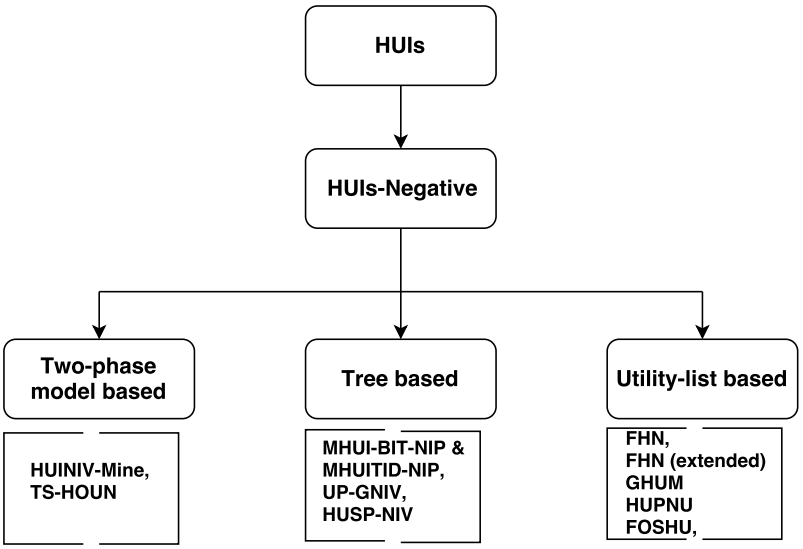

FIGURE 2.2: Taxonomy of high utility itemsets mining with negative utility according to mining approaches

2.5.2 Level-wise Mining Algorithms

Level-wise algorithms follow Apriori-like approach to create candidate such as joining. k-itemsets can generates (k+1)-candidates. Level-wise approach generates itemsets of length k before length (k + 1)-itemsets. Only two (HUINIV-Mine and TS-HOUN) HUIs with negative algorithms mine HUIs using level-wise mining approach.

HUINIV-Mine: In 2009, Chu et al. proposed an algorithm named HUINIV-Mine. It is the first level-wise algorithm that considers negative utility [29]. It is an extension of the Two-Phase algorithm. It mines HUIs with negative item values by following Two-Phase model. HUINIV-Mine presents $RTWU$ based overestimation and pruning strategy. The author demonstrated that HUIs mining with negative utility mast has at-least one positive item. Otherwise, utility would be negative and it would not be a HUIs. It is a level-wise algorithm. Hence, it needs to maintain a large amount of itemsets in memory to find larger itemsets. It scans dataset three times and mines HUIs.

TS-HOUN: In 2014, Lan et al. proposed an algorithm named TS-HOUN [67]. It is the first algorithm that mines HUIs with negative utility and on-shelf time periods. Using on-shelf time periods, the actual utility values of itemsets in a temporal dataset can be accurately evaluated. Most algorithms consider that items have the same shelf time, i.e. that all items are on sale for the same time period. In real-life, some items are only sold

during some short time period (e.g., the summer). The proposed algorithm scans dataset three times and efficiently find high on-shelf utility itemsets with negative profit values from temporal datasets. TS-HOUN uses simply Two-phased pruning strategy to prune the search space. It is purely extension of Two-Phase approach. Hence, it needs much runtime and memory space to finish the mining task and thus generates a lot of candidates.

2.5.3 Tree-based Mining Algorithms

Tree-based algorithms are based on set-enumeration concepts. The candidates can be explored with the use of lexicographic tree or enumeration tree. The main characteristic of tree-based algorithms is that the enumeration tree (or lexicographic tree) provides a certain order of exploration that can be extremely useful in many scenarios. It is assumed that a lexicographic ordering exists among the items in dataset. This lexicographic ordering is essential for efficient set enumeration without repetition.

Tree-based mining algorithms mines HUIs by starting from itemset length-1 (as an initial suffix itemset), constructing its UP-tree and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix itemset with HUIs from UP-tree. Tree-based algorithms transform the problem of finding long itemsets to searching for shorter ones recursively and then concatenating the suffix. Only four (MHUI-BIT-NIP, MHUI-TID-NIP, UP-GNIV and HUSP-NIV) HUIs with negative algorithms mine HUIs using tree-based approach.

MHUI-BIT-NIP & MHUI-TID-NIP: In 2011, Li et al. proposed two algorithms to mine HUIs with negative item profit over continuous stream transaction-sensitive sliding windows [68]. An efficient data structure LexTree-2HTU (Lexicographical Tree with 2-HTU-itemsets) is presented for maintaining a set 2-HTU (high transaction-weighted utilization)-itemsets from current transaction-sensitive sliding window. LexTree-2HTU consists of two components, item-information and a set of trees with prefixes. The item-information is bit-vectors and TID-lists for MHUI-BIT algorithm and MHUI-TID algorithm respectively. The prefix is an entry contained in item-information. Bit-vector and TID-list improve the performance of proposed algorithms. The MHUI-BIT-NIP & MHUI-TID-NIP algorithms mine HUIs in three phases: window initialization phase,

window sliding phase and HUIs generation phase. The proposed algorithms use TWU based search space pruning technique.

UP-GNIV: In 2015, Subramanian et al. proposed an algorithm named UP-GNIV [69] (Utility Pattern-Growth approach for Negative Item Values) to mine HUIs with negative values by using tree-based approach without candidate generation. It is negative utility version of UP-Growth [12] algorithm. It inserts and maintains the itemsets in UP-Tree alike proposed in [11]. UP-GNIV proposed two strategies RNU (Removing Negative item Utilities) and PNI (Pruning Negative Itemsets). UP-GNIV calculates the TU using RNU strategies. PNI is applied for mining HUIs. The author checked the performance of proposed algorithm and the state-of-the-art algorithm HUINIV-Mine using IBM synthetic dataset.

HUSP-NIV: In 2017, Xu et al. proposed high utility sequential itemsets with negative utility value [70]. In high utility sequential itemsets mining, an item occurs more than once in a sequence and its utility has multiple values. None of the state-of-the-art algorithms are suitable for sequential mining. It is the first algorithm that mines high utility sequential itemsets with negative utility. HUSP-NIV is an extension of USPAN algorithm [71]. It uses the same LQS-tree (lexicographic quantitative sequence tree) as USPAN to extract the high utility sequence using I-Concatenation and S-Concatenation mechanisms. I-Concatenation and S-Concatenation mechanisms are utilized from USPAN algorithm to generate new candidate sequences and calculate the utility of sub-nodes based on its super node's utility. The author demonstrated that the proposed algorithm is the first method of its kind.

2.5.4 Utility-list based Mining Algorithms

Both the level-wise and tree-based algorithms mine HUIs from a set of transactions in horizontal data format. Alternatively, mining can also be performed with data presented in vertical data format like proposed in Eclat [66]. Vertical data based mining algorithms, first scan of dataset builds the T_{id} set of each single item. The computation is done by intersection of the T_{id} sets of HUIs k-itemsets to compute the T_{id} sets of the corresponding (k+ 1)-itemsets. This process repeats until itemsets fulfill min_util threshold. In the generation of candidate (k + 1)-itemset from k-itemsets, the merit of

TABLE 2.14: An overview of High utility itemset mining algorithms with negative utility values

Algorithm	Year	Author	Dataset scanning	Data structure	Dataset	Mining	Pruning strategy	State-of-the-art algorithms	Base algorithm
Level-wise mining algorithms									
HUINIV-Mine [29]	2009	Chu et al.	Thrice	...	Transactional	HUIs	TWU-based	None (first of its kind)	Two-Phase[22]
TS-HOUN [67]	2014	Lan et al.	Thrice	...	Temporal transactional	High on-shelf utility itemsets	TWU-based	None (first of its kind)	Two-Phase[22]
Tree-based mining algorithms									
MHUI-BIT-NIP & MHUITID-NIP [68]	2011	Li et al.	Once	Lexicographical tree	Data stream	HUIs	TWU-based	self	THUI-Mine[72] (Two-Phase)
UP-GNIV [69]	2015	Subramanian et al.	Twice	UP-Tree	Transactional	HUIs	DGU, DGN [12]	HUINIV-Mine	UP-Growth[12]
HUSP-NIV [70]	2017	Xu et al.	Once	Lexicographical tree	Sequential transactional	High utility sequential itemsets	Depth-pruning, width-pruning and depth & width pruning.	None (first of its kind)	USpan[71]
Utility-list based mining algorithms									
FHN [73]	2014	Viger et al.	Once	Utility-list	Transactional	HUIs	TWU-based & EUCP	HUINIV-Mine	FHM[27]
FOSHU [57]	2015	Viger et al.	Once	Utility-list	Temporal transactional	high on-shelf utility itemsets	TWU-based & EUCP	TS-HOUN	FHN
FHN[30] (extended)	2016	Lin et al.	Once	Utility-list	Transactional	HUIs	TWU-based, EUCP & LA-Prune	HUINIV-Mine	FHN
GHUM [74]	2017	Krishnamoorthy	Once	Simplified utility-list	Transactional	HUIs	U-Prune, LA-Prune, N-Prune & A-Prune	FHN	FHN
HUPNU [75]	2017	Gan et al.	Once	PU^{+-}-list [a]	Uncertain transactional	HUIs	PU-Prune, RTWU & EUCP	None (first of its kind)	...

[a] Probability-Utility list with Positive-and-Negative profits

this method is that there is no need to scan dataset to find the utility of (k + 1)-itemsets. This is because the T_{id} set of each k-itemset carries the complete information required for counting utility. Only four (FHN, GHUM, HUPNU and FOSHU) HUIs with negative algorithms mine HUIs using vertical dataset mining approach.

FHN: In 2014, Fournier-Viger et al. proposed an algorithm named FHN (Faster High-Utility itemset miner with Negative unit profits) [73]. FHN is an extension of FHM algorithm [27]. It utilizes utility-list structure to explore the search space of itemsets. FHN uses separate utility-list data structure for positive and negative utility values. It also utilizes EUCS (Estimated Utility Co-occurrence Structure) which provides an efficient pruning strategy to limit the search space. The author demonstrated that FHN is up to 500 times faster and uses up to 250 times less memory than the state-of-the-art algorithm HUINIV-Mine. Later, in 2016, an extensive version of basic FHN algorithm was proposed. The extended FHN [30] utilizes LA-Prune strategy proposed in [25] to prune the search space. The extended FHN is shown to be 2-3 orders of magnitude faster than HUINIV-Mine.

GHUM: In 2017, Krishnamoorthy et al. proposed an efficient algorithm named GHUM (Generalized High Utility Mining) [74]. GHUM presents a simplified utility-list based data structure to store information of itemsets. It does not use separate utility-list to store items. It sorts the negative items in ascending order using support (frequency) of items to generate candidates efficiently. GHUM utilizes and modifies existing pruning strategies U-Prune and LA-Prune [25]. Utility-list based algorithms require expensive utility list intersections and candidate evaluations. Hence, it presents a novel pruning strategy (N-Prune) to significantly reduce the total number of evaluations. It also presents an *anti-monotonic property* based pruning strategy (A-Prune) for mining HUIs with negative items. GHUM is shown more than an order of magnitude improvement at a fraction of the memory over the current state-of-the-art FHN.

HUPNU: In 2017, Gan et al. proposed an algorithm named HUPNU (mining High-Utility itemsets with both Positive and Negative unit profits from Uncertain databases) to mining HUIs with negative utility value from uncertain datasets [75]. It considers the probability values of items for mining HUIs. It uses a vertical PU$\pm$-list (Probability-Utility list with Positive-and-Negative profits) structure to store both

negative and positive items. It presents six pruning strategies to reduce the search space, a number of unpromising itemsets can be early pruned when constructing the PU±-list.

FOSHU: In 2015, Fournier-Viger et al. proposed an algorithm FOSHU (Faster On-Shelf High Utility itemset miner) [57] to mine HUIs with negative utility from on-shelf datasets. On-shelf items consider the shelf time of items. It is an extension of FHN algorithm [73]. Hence, it utilizes the utility-list structure to store the information of items. FOSHU mines itemsets in a single phase without generating candidates and also mines all times periods at the same time rather than mine each time period separately unlike USpan [71]. Therefore, it avoids the costly merge operations of itemsets found in each time period.

2.5.5 Discussion

The previous section has reviewed three main types of HUIs with negative utility mining algorithms. The key differences between these algorithms can be described in terms of dataset scanning, data structure used, pruning strategies and base algorithm. TABLE 2.14 summarizes the characteristics of ten negative utility based HUIs algorithms. It is noticed that there is no comprehensive study covering all of these algorithms in the literature.

TABLE 2.14 depicts basic details of algorithms such as name of algorithms, publishing year of algorithms, name of the authors. The "Dataset scanning" column of this table gives the information how many times algorithm scans dataset; the "Data structure" shows the type of data structure used to store the information of items; the "Dataset" type of inputted dataset used for experiments; This table shows that all the mining algorithms; the "Mining" represents output HUIs; the "Pruning strategy" column shows the strategies used to prune search space; the "State-of-the-art algorithms" shows the name the state-of-the-art algorithms and the "Base algorithm" represents the base algorithm for the presented algorithms. This table shows all the mining algorithms used to mine HUIs with negative utility value.

2.6 Summary

In this chapter, we introduce HUIs mining algorithms and their related work. In Section 2.1, we provide the preliminary definitions of HUIs mining and some existing work. In Section 2.2, we briefly discuss the constraint-based HUIs mining. We present constraint-based FIM algorithms. Then, we describe constraint-based HUIs mining algorithm. In Section 2.3, we discuss existing top-k HUIs mining algorithms. We classify top-k HUIs mining algorithms into two parts, Two-Phase and One-Phase. We briefly discuss both (Two-Phase and One-Phase) type of algorithms. In Section 2.4, we describe the recent work on closed high utility itemset mining. Here, we classify closed HUIs mining algorithms into three parts, Level-wise, Tree-based and Utility-list based. Lastly, in Section 2.5, we discuss HUIs mining algorithms with negative utility value.