Fournier-Viger., P., Lin, J. C.-W., Truong, T., Nkambou, R. (2019). **A survey of high utility itemset mining**. In: Fournier-Viger et al. (eds). High-Utility Pattern Mining: Theory, Algorithms and Applications, Springer (to appear), p. 1-46.

This is the draft of the chapter. Final version on SpringerLink.

# A Survey of High Utility Itemset Mining

Philippe Fournier-Viger, Jerry Chun-Wei Lin, Tin Truong Chi, Roger Nkambou

**Abstract** High utility pattern mining is an emerging data science task, which consists of discovering patterns having a high importance in databases. The utility of a pattern can be measured in terms of various objective criterias such as its profit, frequency, and weight. Among the various kinds of high utility patterns that can be discovered in databases, high utility itemsets are the most studied. A high utility itemset is a set of values that appears in a database and has a high importance to the user, as measured by a utility function. High utility itemset mining generalizes the problem of frequent itemset mining by considering item quantities and weights. A popular application of high utility itemset mining is to discover all sets of items purchased together by customers that yield a high profit. This chapter provides an introduction to high utility itemset mining, reviews the state-of-the-art algorithms, their extensions, applications, and discusses research opportunities. This chapter is aimed both at those who are new to the field of high utility itemset mining, as well as researchers working in the field.

## 1 Introduction

The goal of data mining is to extract patterns or train models from databases to understand the past or predict the future. Various types of data mining algorithms

Philippe Fournier-Viger
Harbin Institute of Technology (Shenzhen), Shenzhen, China, e-mail: philfv8@yahoo.com

Jerry Chun-Wei Lin
Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway e-mail: jerrylin@ieee.org

Tin Truong Chi
University of Dalat, Dalat, Vietnam, e-mail: tintc@dlu.edu.vn

Roger Nkambou
University of Quebec in Montreal, e-mail: nkambou.roger@uqam.ca

have been proposed to analyze data [1, 38]. Several algorithms produce models that operates as black boxes. For example, several types of neural networks are designed to perform predictions very accurately but cannot be easily interpreted by humans. To extract knowledge from data that can be understood by humans, *pattern mining algorithms* are designed [27, 28]. The goal is to discover patterns in data that are interesting, useful, and/or unexpected. An advantage of pattern mining over several other data mining approaches is that discovering patterns is a type of *unsupervised learning* as it does not require labeled data. Patterns can be directly extracted from raw data, and then be used to understand data and support decision-making. Pattern mining algorithms have been designed to extract various types of patterns, each providing different information to the user, and for extracting patterns from different types of data. Popular types of patterns are sequential patterns [27], itemsets [28], clusters, trends, outliers, and graph structures [38].

Research on pattern mining algorithms has started in the 1990s with algorithms to discover *frequent patterns* in databases [2]. The first algorithm for frequent pattern mining is Apriori [2]. It is designed to discover *frequent itemsets* in customer transaction databases. A *transaction database* is a set of records (transactions) indicating the items purchased by customers at different times. A *frequent itemset* is a group of values (items) that is frequently purchased by customers (appears in many transactions) of a transaction database. For example, a frequent itemset in a database may be that many customers buy the item *noodles* with the item *spicy sauce*. Such patterns are easily understandable by humans and can be used to support decision-making. For instance, the pattern *{noodles, spicy sauce}* can be used to take marketing decisions such as co-promoting noodles with spicy sauce. The discovery of frequent itemsets is a well-studied data mining task, and has applications in numerous domains. It can be viewed as the general task of analyzing a database to find co-occurring values (items) in a set of database records (transactions) [10, 16, 20, 37, 61, 64, 65, 66].

Although, frequent pattern mining is useful, it relies on the assumption that frequent patterns are interesting. But this assumption does not hold for numerous applications. For example, in a transaction database, the pattern $\{milk, bread\}$ may be highly frequent but may be uninteresting as it represents a purchase behavior that is common, and may yield a low profit. On the other hand, several patterns such as $\{caviar, champagne\}$ may not be frequent but may yield a higher profit. Hence, to find interesting patterns in data, other aspects can be considered such as the profit or utility.

To address this limitation of frequent itemset mining, an emerging research area is the discovery of *high utility patterns* in databases [31, 52, 56, 58, 59, 83, 87, 94]. The goal of utility mining is to discover patterns that have a high utility (a high importance to the user), where the utility of a pattern is expressed in terms of a *utility function*. A utility function can be defined in terms of criteria such as the profit generated by the sale of an item or the time spent on webpages. Various types of high utility patterns have been studied. This chapter surveys research on the most popular type, which is *high utility itemsets* [83]. Mining high utility itemsets can be seen as a generalization of the problem of frequent itemset mining where the input is a transaction database where each item has a weight representing its importance, and

where items can have non binary quantities in transactions. This general problem formulation allows to model various tasks such as discovering all itemsets (sets of items) that yield a high profit in a transaction database, finding sets of webpages where users spend a large amount of time, or finding all frequent patterns as in traditional frequent pattern mining. High utility itemset mining is a very active research area. This chapter provides a comprehensive survey of the field that is both an introduction and a guide to recent advances and research opportunities.

The rest of this chapter is organized as follows. Section 2 introduces the problem of high utility itemset mining, its key properties, and how it generalizes frequent itemset mining. Section 3 surveys popular techniques for efficiently discovering high utility itemsets in databases. Section 4 presents the main extensions of high utility itemset mining. Section 5 discusses research opportunities. Section 6 present open-source implementations. Finally, section 7 draws a conclusion.

## 2 Problem Definition

This section first introduces the problem of frequent itemset mining [2], and then explains how it is generalized as high utility itemset mining [31, 52, 56, 58, 59, 83, 87, 94]. Then, key properties of the problem of high utility itemset mining are presented and contrasted with those of frequent itemset mining.

### *2.1 Frequent Itemset Mining*

The problem of frequent itemset mining consists of extracting patterns from a transaction database. In a transaction database, each record (called transaction) is a set of items (symbols). Formally, a transaction database $D$ is defined as follows. Let there be the set $I$ of all items (symbols) $I = \{i_1, i_2, \ldots, i_m\}$ that occur in the database. A transaction database $D$ is a set of records, called transactions, denoted as $D = \{T_0, T_1, \ldots, T_n\}$, where each transaction $T_q$ is a set of items (i.e. $T_q \subseteq I$), and has a unique identifier $q$ called its TID (Transaction IDentifier). For example, consider the customer transaction database shown in Table 1. The database in Table 3 contains five transactions denoted as $T_0, T_1, T_3$ and $T_4$. The transaction $T_2$ indicates that the items $a$, $c$ and $d$ were purchased together by a customer in that transaction.

The goal of frequent itemset mining is to discover itemsets (sets of items) that have a high support (that appear frequently). Formally, an *itemset* $X$ is a finite set of items such that $X \subseteq I$. Let the notation $|X|$ denote the set cardinality or, in other words, the number of items in an itemset $X$. An itemset $X$ is said to be of length $k$ or a $k$-itemset if it contains $k$ items ($|X| = k$). For instance, $\{a, b, c\}$ is a 3-itemset, and $\{a, b\}$ is a 2-itemset. The support measure is defined as follows.

Table 1: A transaction database

| TID | Transaction |
|---|---|
| $T_0$ | $a, b, c, d, e$ |
| $T_1$ | $b, c, d, e$ |
| $T_2$ | $a, c, d$ |
| $T_3$ | $a, c, e$ |
| $T_4$ | $b, c, e$ |

**Definition 1 (Support measure).** The support (frequency) of an itemset $X$ in a transaction database $D$ is denoted as $sup(X)$ and defined as $sup(X) = |\{T|X \subseteq T \wedge T \in D\}|$, that is the number of transactions containing $X$.

For example, the support of the itemset $\{a, c\}$ in the database of Table 3 is 3, since this itemset appears in three transactions ($T_0, T_2$ and $T_3$). This definition of the support measure is called *relative support*. Another equivalent definition is to express the support as a percentage of the total number of transactions (called *absolute support*). For example, the absolute support of {a,c} is 60% since it appears in 3 out of 5 transactions. The problem of frequent itemset mining is defined as follows:

**Definition 2 (Frequent itemset).** Let there be a threshold $minsup > 0$, defined by the user. An itemset $X$ is a *frequent itemset* if its support $sup(X)$ is no less than that $minsup$ threshold (i.e. $sup(X) \geq minsup$). Otherwise, $X$ is an *infrequent itemset*.

**Definition 3 (Problem definition).** The *problem of frequent itemset mining* is to discover all frequent itemsets in a transaction database $D$, given the $minsup$ threshold set by the user.

For example, consider the database of Table 3 and $minsup = 3$. There are 11 frequent itemsets, listed in Table 2.

Table 2: The frequent itemsets for $minsup = 3$

| Itemset | Support | Itemset | Support | Itemset | Support |
|---|---|---|---|---|---|
| $\{a\}$ | 3 | $\{e\}$ | 4 | $\{b, e\}$ | 3 |
| $\{b\}$ | 3 | $\{a, c\}$ | 3 | $\{c, e\}$ | 4 |
| $\{c\}$ | 5 | $\{b, c\}$ | 3 | $\{b, c, e\}$ | 3 |
| $\{d\}$ | 3 | $\{c, d\}$ | 3 | | |

The problem of frequent itemset mining has been studied for more than two decades. Numerous algorithms have been proposed to discover frequent patterns efficiently, including Apriori [2], FP-Growth [39], Eclat [91], LCM [81] and H-Mine [69]. Although frequent itemset mining has many applciations, a strong assumption of frequent itemset mining is that frequent patterns are useful or interesting to the user, which is not always true. To address this important limitation of traditional frequent pattern mining, it has been generalized as high utility itemset mining, where items are annotated with numerical values and patterns are selected based on a user-defined utility function.

## 2.2 High Utility Itemset Mining

The task of high utility itemset mining [31, 52, 56, 58, 59, 87, 94] consists of discovering patterns in a generalized type of transaction database called *quantitative transaction database*, where additional information is provided, that is the quantities of items in transactions, and weights indicating the relative importance of each item to the user.

Formally, a quantitative transaction database $D$ is defined as follows. Let there be the set $I$ of all items $I = \{i_1, i_2, \ldots i_m\}$. A quantitative transaction database $D$ is a set of transactions, denoted as $D = \{T_0, T_1, \ldots, T_n\}$, where each transaction $T_q$ is a set of items (i.e. $T_q \subseteq I$), and has a unique identifier $q$ called its TID (Transaction IDentifier). Each item $i \in I$ is associated with a positive number $p(i)$, called its *external utility*. The external utility of an item is a positive number representing its relative importance to the user. Furthermore, every item $i$ appearing in a transaction $T_c$ has a positive number $q(i, T_c)$, called its *internal utility*, which represents the quantity of $i$ in the transaction $T_c$.

To illustrate these definitions, consider an example customer transaction database depicted in Table 3, which will be used as running example. In this example, the set of items is $I = \{a, b, c, d, e\}$. It can be considered as representing different products sold in a retail store such as $apple$, $bread\ cereal$, $duck$ and $egg$. The database in Table 3 contains five transactions $(T_0, T_1, \ldots T_4)$. The transaction $T_3$ indicates that items $a$, $c$, and $e$ were bought with purchase quantities (internal utilities) of respectively 2, 6, and 2. Table 4 provides the external utilities of the items, which represents their unit profits. Assume that the dollar ($) is used as currency. The sale of a unit of items $a$, $b$, $c$, $d$, and $e$ yields a profit of 5$, 2$, 1$, 2$ and 3$, respectively.

Table 3: A quantitative transaction database

| TID | Transaction |
|---|---|
| $T_0$ | $(a, 1), (b, 5), (c, 1), (d, 3), (e, 1)$ |
| $T_1$ | $(b, 4), (c, 3), (d, 3), (e, 1)$ |
| $T_2$ | $(a, 1), (c, 1), (d, 1)$ |
| $T_3$ | $(a, 2), (c, 6), (e, 2)$ |
| $T_4$ | $(b, 2), (c, 2), (e, 1)$ |

Table 4: External utility values

| Item | External utility |
|---|---|
| a | 5 |
| b | 2 |
| c | 1 |
| d | 2 |
| e | 3 |

The goal of high utility itemset mining is to discover itemsets (sets of items) that appear in a quantitative database and have a high utility (e.g. yield a high profit). The utility of an itemset is a measure of its importance in the database, which is computed using a utility function. The utility measure is generally defined by the following definition, although alternative measures have been proposed [83] (which will be reviewed in Section 4). In the running example, the utility measure is interpreted as the amount of profit generated by each set of items.

**Definition 4 (Utility measure).** The utility of an item $i$ in a transaction $T_c$ is denoted as $u(i, T_c)$ and defined as $p(i) \times q(i, T_c)$. In the context of analyzing customer transactions, it represents the profit generated by the sale of the item $i$ in the transaction $T_c$. The *utility of an itemset $X$ in a transaction $T_c$* is denoted as $u(X, T_c)$ and defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$ if $X \subseteq T_c$. Otherwise $u(X, T_c) = 0$. The *utility of an itemset $X$* in a database $D$ is denoted as $u(X)$ and defined as $u(X) = \sum_{T_c \in g(X)} u(X, T_c)$, where $g(X)$ is the set of transactions containing $X$. It represents the profit generated by the sale of the itemset $X$ in the database.

For example, the utility of item $a$ in transaction $T_2$ is $u(a, T_2) = 5 \times 2 = 10$. The utility of the itemset $\{a, c\}$ in $T_2$ is $u(\{a, c\}, T_2) = u(a, T_2) + u(c, T_2) = 5{\times}2 + 1{\times}6 = 16$. The utility of the itemset $\{a, c\}$ in the database is $u(\{a, c\}) = u(a) + u(c) = u(a, T_0) + u(a, T_2) + u(a, T_3) + u(c, T_0) + u(c, T_2) + u(c, T_3) = 5 + 5 + 10 + 1 + 1 + 6 = 28$. Thus, the utility of $\{a, c\}$ in the database can be interpreted as the total amount of profit generated by items $a$ and $c$ when they are purchased together. The problem of high utility itemset mining is defined as follows:

**Definition 5 (High-utility itemset).** An itemset $X$ is a *high-utility itemset* if its utility $u(X)$ is no less than a user-specified minimum utility threshold $minutil$ set by the user (i.e. $u(X) \geq minutil$). Otherwise, $X$ is a *low-utility itemset*.

**Definition 6 (Problem definition).** The *problem of high-utility itemset mining* is to discover all high-utility itemsets, given a $minutil$ threshold set by the user [83].

Note that in some studies, the utility of an itemset is expressed as a percentage of the total utility in the database. Discovering patterns using this definition called *absolute utility* [79], is equivalent to using the above definition, and results in finding the same set of patterns.

High utility itemset mining has numerous applications. For the application of market basket analysis, the problem of high-utility itemset mining can be interpreted as finding all sets of items that have generated a profit greater than or equal to $minutil$. For example, for the running example, if $minutil = 25$, the set of HUIs is shown in Table 5. Several algorithms have been proposed to discover high utility itemsets (reviewed in the next section).

It is interesting to note that because the problem of high utility itemset mining is more general than the problem of frequent itemset mining, any algorithm for discovering high utility itemsets can also be used to discover frequent itemsets in a transaction database. To do that, the following steps can be applied:

Table 5: The high utility itemsets for $minutil = 25$

| Itemset | Utility | Itemset | Utility | Itemset | Utility |
|---|---|---|---|---|---|
| $\{a, c\}$ | 28 | $\{b, c, d\}$ | 34 | $\{b, d, e\}$ | 36 |
| $\{a, c, e\}$ | 31 | $\{b, c, d, e\}$ | 40 | $\{b, e\}$ | 31 |
| $\{a, b, c, d, e\}$ | 25 | $\{b, c, e\}$ | 37 | $\{c, e\}$ | 27 |
| $\{b, c\}$ | 28 | $\{b, d\}$ | 30 | | |

1. The transaction database is converted to a quantitative transaction database. For each item $i \in I$, the external utility value of $i$ is set to 1, that is $p(i) = 1$ (to indicate that all items are equally important). Moreover, for each item $i$ and transaction $T_c$, if $i \in T_c$, set $q(i, T_c) = 1$. Otherwise, set $q(i, T_c) = 0$.
2. Then a high utility mining algorithm is applied on the resulting quantitative transaction database with *minutil* set to *minsup*, to obtain the frequent itemsets.

For example, the database of Table 1 can be transformed in a quantitative database. The result is the transaction database of Tables 6 and 7. Then, frequent itemsets can be mined from this database using a high utility itemset mining algorithm. However, although a high utility itemset mining algorithm can be used to mine frequent itemsets, it may be preferable to use frequent itemset mining algorithms when performance is important as these latter are optimized for this task.

Table 7: External utility values for the database of Table 6

| TID | Transaction |
|---|---|
| $T_0$ | $(a, 1), (b, 1), (c, 1), (d, 1), (e, 1)$ |
| $T_1$ | $(b, 1), (c, 1), (d, 1), (e, 1)$ |
| $T_2$ | $(a, 1), (c, 1), (d, 1)$ |
| $T_3$ | $(a, 1), (c, 1), (e, 1)$ |
| $T_4$ | $(b, 1), (c, 1), (e, 1)$ |

| Item | External utility |
|---|---|
| a | 1 |
| b | 1 |
| c | 1 |
| d | 1 |
| e | 1 |

## *2.3 Key Properties of the Problem of High Utility Itemset Mining*

For a given quantative database and minimum utility threshold, the problem of high utility itemset mining always has a single solution. It is to enumerate all patterns that have a utility greater than or equal to the user-specified minimum utility threshold.

The problem of high utility itemset mining is difficult for two main reasons. The first reason is that the number of itemsets to be considered can be very large to find those that have a high utility. Generally, if a database contains $m$ distinct items there are $2^m - 1$ possible itemsets (excluding the empty set). For example, if $I = \{a, b, c\}$, the possible itemsets are $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, and $\{a, b, c\}$. Thus, there

are $2^3 - 1 = 7$ itemsets, which can be formed with $I = \{a, b, c\}$. A naive approach to solve the problem of high utility itemset mining is to count the utilities of all possible itemsets by scanning the database, to then keep the high utility itemsets. Although this approach produces the correct result, it is inefficient. The reason is that the number of possible itemsets can be very large. For example, if a retail store has 10,000 items on its shelves ($m = 10,000$), the utilities of $2^{10,000} - 1$ possible itemsets should be calculated, which is unmanageable using the naive approach. It is to be noted that the problem of high utility itemset mining can be very difficult even for small databases. For example, a database containing a single transaction of 100 items can produce $2^{100} - 1$ possible itemsets. Thus, the size of the search space (the number of possible itemsets) can be very large even if there are few transactions in a database. In fact, the size of the search space does not only depend on the size of the database, but also on how similar the transactions are in the database, how large the utility values are, and also on how low the *minutil* threshold is set by the user.

A second reason why the problem of high utility itemset mining is difficult is that high utility itemsets are often scattered in the search space. Thus, many itemsets must be considered by an algorithm before it can find the actual high utility itemsets. To illustrate this, Fig. 1 provides a visual representation of the search space for the running example, as a *Hasse diagram*. A Hasse diagram is a graph where each possible itemset is represented as a node, and an arrow is drawn from an itemset $X$ to another itemset $Y$ if and only if $X \subseteq Y$ and $|X| + 1 = |Y|$. In Figure 1, high utility itemsets are depicted using light gray nodes, while low utility itemsets are represented using white nodes. The utility value of each itemset is also indicated. An important observation that can be made from that figure is that the utility of an itemset can be greater, higher or equal, to the utility of any of its supersets/subsets. For example, the utility of the itemset $\{b, c\}$ is 28, while the utility of its supersets $\{b, c, d\}$ and $\{a, b, c, d, e\}$ are 34 and 25, respectively. Formally, it is thus said that the utility measure is neither monotone nor anti-monotone.

*Property 1 (The utility measure is neither monotone nor anti-monotone).* Let there be two itemsets $X$ and $Y$ such that $X \subset Y$. The relationship between the utilities of $X$ and $Y$ is either $u(X) < u(Y)$, $u(X) > u(Y)$, or $u(X) = u(Y)$ [83].

Because of this property, the high utility itemsets appear scattered in the search space, as it can be observed in Fig. 1. This is the main reason why the problem of high utility itemset mining is more difficult than the problem of frequent itemset mining [2]. In frequent itemset mining, the support measure has the nice property of being monotone [2], that is, the support of an itemset is always greater than or equal to the frequency of any of its supsersets.

*Property 2 (The support measure is monotone).* Let there be two itemsets $X$ and $Y$ such that $X \subset Y$. It follows that $sup(X) \geq sup(Y)$ [2].

For example, in the database of Table 1, the support of $\{b, c\}$ is 3, while the support of its supersets $\{b, c, d\}$ and $\{a, b, c, d, e\}$ are 2 and 1, respectively. The monotonicity of the support measure makes it easy to find frequent patterns as it

guarantees that all supersets of an infrequent itemset are also infrequent [2]. Thus, a frequent itemset mining algorithm can discard all supersets of an infrequent itemset from the search space. For example, if an algorithm finds that the itemset $\{a, d\}$ is infrequent, it can directly eliminate all supersets of $\{a, d\}$ from further exploration, thus greatly reducing the search space. The search space for the example database of Table 1 is illustrated in Fig. 2. The anti-monotonicity of the support can be clearly observed in this picture as a line is drawn that clearly separates frequent itemsets from infrequent itemsets. Property 2 is also called the *downward-closure property*, *anti-monotonicity-property* or *Apriori-property* [2]. Although it holds for the support measure, it does not hold for the utility measure used in high utility itemset mining. As a result, in Fig. 1, it is not posssible to draw a clear line to separate low utility itemsets from high utility itemsets.

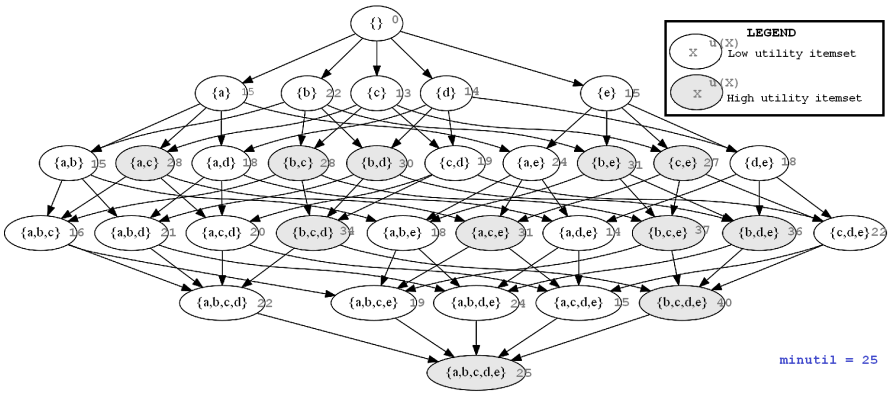


Fig. 1: The search space of high utility itemset mining for the running example and $minutil = 25$

Due to the large search space in high utility itemset mining, it is thus important to design fast algorithms that can avoid considering all possible itemsets in the search space and that process each itemset in the search space as efficiently as possible, while still finding all high utility itemsets. Moreover, because the utility measure is not monotone nor anti-monotone, efficient strategies for reducing the search space used in frequent itemset mining cannot be directly used to solve the problem of high utility itemset mining. The next section explains the key ideas used by the state-of-the-art high utility itemset mining algorithms to solve the problem efficiently.

## 3 Algorithms

Several high utility itemset mining algorithms have been proposed such as UMining [82], Two-Phase [59], IHUP [5], UP-Growth [79], HUP-Growth [52], MU-

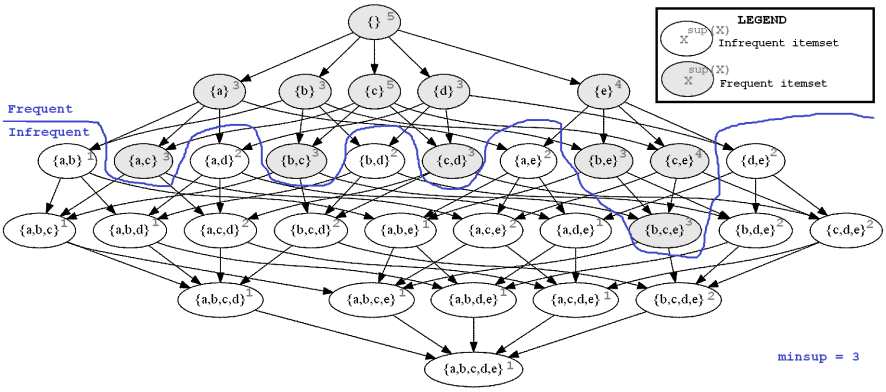


Fig. 2: The search space of frequent itemset mining for the database of Table 1 and $minsup = 3$

Growth [87], HUI-Miner [58], FHM [31], ULB-Miner [17], HUI-Miner* [71] and EFIM [94]. All of these algorithms have the same input and the same output. The differences between these algorithms lies in the data structures and strategies that are employed for searching high utility itemsets. More specifically, algorithms differ in (1) whether they use a depth-first or breadth-first search, (2) the type of database representation that they use internally or externally, (3) how they generate or determine the next itemsets to be explored in the search space, and (4) how they compute the utility of itemsets to determine if they satisfy the minimum utility constraint. These design choices influence the performance of these algorithms in terms of execution time, memory usage and scalability, and also how easily these algorithms can be implemented and extended for other data mining tasks. Generally, all high utility itemset mining algorithms are inspired by classical frequent itemset mining algorithms, although they also introduce novel ideas to cope with the fact that the utility measure is neither monotone nor anti-monotone.

Early algorithms for the problem of high utility itemset mining were incomplete algorithms that could not find he complete set of high utility itemsets due to the use of heuristic strategies to reduce the search space. For example, this is the case of the UMining and UMining_H algorithms [82]. In the rest of this section, complete algorithms are reviewed, which guarantees to find all high utility itemsets. It is also interesting to note that the term *high utility itemset mining* has been first used in 2003 [11], although the problem definition used by most researchers nowadays, and used in this chapter, has been proposed in 2005 [83].

### *3.1 Two Phase Algorithms*

The first complete algorithms to find high utility itemsets perform two phases, and are thus said to be *two phase algorithms*. This includes algorithms such as Two-Phase [59], IHUP [5], UP-Growth [79], HUP-Growth [52], and MU-Growth [87]. The breakthrough idea that has inspired all these algorithms was introduced in Two-Phase [59]. It is that it is possible to define a monotone measure that is an upper-bound on the utility measure, and to use that measure to safely reduce the search space without missing any high utility itemsets. The measure proposed in the Two-Phase algorithm is the TWU (Transaction Weighted Utilization) measure, which is defined as follows:

**Definition 7 (The TWU measure).** The *transaction utility* (*TU*) of a transaction $T_c$ is the sum of the utilities of all the items in $T_c$. i.e. $TU(T_c) = \sum_{x \in T_c} u(x, T_c)$. The *transaction-weighted utilization* (*TWU*) of an itemset $X$ is defined as the sum of the transaction utilities of transactions containing $X$, i.e. $TWU(X) = \sum_{T_c \in g(X)} TU(T_c)$.

For instance, the transaction utilities of $T_0, T_1, T_2, T_3$ and $T_4$ are respectively 25, 20, 8, 22 and 9. The TWU of single items $a$, $b$, $c$, $d$, $e$ are respectively 55, 54, 84, 53 and 76. The TWU of the itemset $\{c, d\}$ is $TWU(\{c, d\}) = TU(T_0) + TU(T_1) + TU(T_2) = 25 + 20 + 8 = 53$. The TWU measure is said to be an upper-bound on the utility measure that is monotone. This idea is formalized as the next property.

*Property 3 (The TWU is a monotone upper-bound on the utility measure).* Let there be an itemset $X$. The TWU of $X$ is no less than its utility ($TWU(X) \geq u(X)$). Moreover, the TWU of $X$ is no less than the utility of its supersets ($TWU(X) \geq u(Y) \forall Y \supset X$). The proof is provided in [59]. Intuitively, since the TWU of $X$ is the sum of the utility of transactions where $X$ appears, its TWU must be greater or equal to the utility of $X$ and any of its supersets.

The TWU measure is interesting because it can be used to reduce the search space. For this purpose, the following property was proposed.

*Property 4 (Pruning the search space using the TWU).* For any itemset $X$, if $TWU(X) < minutil$, then $X$ is a low-utility itemset as well as all its supersets. This directly follows from Property 3.

For example, the utility of the itemset $\{a, b, c, d\}$ is 20, and $TWU(\{a, b, c, d\}) = 25$. Thus, by the Property 4, it is known that any supersets of $\{a, b, c, d\}$ cannot have a TWU and a utility greater than 25. As a result, if the user sets the *minutil* threshold to a value greater than 25, all supsersets of $\{a, b, c, d\}$ can be eliminated from the search space as it is known by Property 4 that their utilities cannot be greater than 25.

Algorithms such as IHUP [5], PB [47], Two-Phase [59], UP-Growth [79], HUP-Growth [52] and MU-Growth [87] utilize Property 4 as main property to prune the search space. They operate in two phases:

1. In the first phase, these algorithms calculate the TWU of itemsets in the search space. For an itemset $X$, if $TWU(X) < X$, then $X$ and its supersets cannot be high utility itemsets. Thus, they can be eliminated from the search space and their TWU do not need to be calculated. Otherwise, $X$ and its supersets may be high utility itemsets. Thus, $X$ is kept in memory as a candidate high utility itemset and its supersets may be explored.
2. In the second phase, the exact utility of each candidate high utility itemset $X$ found in phase 1 is calculated by scanning the database. If $u(X) \geq minutil$, then $X$ is output since it is a high utility itemset.

This two phase process ensures that only low utility itemsets are pruned from the search space. Thus, two phase algorithms can find all high utility itemsets while reducing the search space to improve their performance. A representative two phase algorithm is Two-Phase [59]. It is described next, and then its limitations are discussed.

### 3.1.1 The Two-Phase Algorithm

The Two-Phase algorithm generalizes the Apriori algorithm, which was proposed for frequent itemset mining [2]. Two-Phase explores the search space of itemsets using a *breadth-first search*. A breadth-first search algorithm first considers single items (1-itemsets). In the running example, those are $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$ and $\{e\}$. Then, Two-Phase generates 2-itemsets such as $\{a, b\}, \{a, c\}, \{a, d\}$, and then 3-itemsets, and so on, until it generates the largest itemset $\{a, b, c, d, e\}$ containing all items. Two-Phase [59] takes a quantitative transaction database and the *minutil* threshold as input. Two-Phase uses a standard database representation, as shown in Tables 3, also called a *horizontal database*. The pseudocode of Two-Phase is given in Algorithm 1. In phase 1, Two-Phase scans the database to calculate the TWU of each 1-itemset (line 1). Then, Two-Phase uses this information to identify the set of all candidate high-utility items, denoted as $P_1$ (line 2). An itemset $X$ is said to be a *candidate high utility itemset* if $TWU(X) \geq minutil$. Then, Two-Phase performs a breadth-first search to find larger candidate high utility itemsets (line 4 to 10). During the search, Two-Phase uses the candidate high utility itemsets of a given length $k - 1$ (denoted as $P_{k-1}$) to generate itemsets of length $k$ (denoted as $P_k$). This is done by combining pairs of candidate high utility itemsets of length $k$ that share all but one item (line 5). For example, if the candidate high utility 1-itemsets are $\{a\}, \{b\}, \{c\}$ and $\{e\}$, Two-Phase combine pairs of these itemsets to obtain the following 2-itemsets: $\{a, b\}$, $\{a, c\}$, $\{a, e\}$, $\{b, c\}$, $\{b, e\}$, and $\{c, e\}$. After generating itemsets of length $k$, Two-Phase checks if the $(k - 1)$-subsets of each itemset are candidate high utility itemsets. If an itemset $X$ has a $(k - 1)$-subset that is not a candidate high utility itemset, $X$ cannot be a high utility itemset (it would violate Property 4) and it is thus removed from the set of $k$-itemsets. Then, Two-Phase scans the database to calculate the TWU of all remaining itemsets in $P_k$ (line 7). Each itemset having a TWU not less than *minutil* is added to the set $P_k$ of candidate high utility $k$-itemsets (line 8). This process is repeated until no candidate high utility

itemsets can be generated. Then, the second phase is performed (line 12 to 13). Two-Phase scans the database to calculate the exact utility of each candidate high utility itemsets. The set of all candidate high utility itemsets that have a utility not less than *minutil* are the high utility itemsets. They are returned to the user (line 13).

**Algorithm 1:** The Two-Phase algorithm

**input :** $D$: a horizontal transaction database, $minutil$: a user-specified threshold
**output :** the set of high utility itemsets

1 Scan the database to calculate the TWU of all items in $I$; `// PHASE 1`
2 $P_1 = \{i | i \in I \wedge sup(\{i\}) \geq minsup\}$ ; `//` $P_1$ `: candidate high utility 1-itemsets`
3 $k = 2$;
4 **while** $P_k \neq \emptyset$ **do**
5 $P_k$ = `itemsetGeneration` $(P_{k-1})$ ; `//` $P_k$ `: k-itemsets`
6 Remove each candidate $X \in P_k$ that contains a $(k-1)$-itemset that is not in $P_{k-1}$;
7 Scan the database to calculate the TWU of each candidate $X \in P_k$;
8 $P_k = \{X | X \in P_k \wedge TWU(X) \geq minutil\}$ ; `//` $P_k$ `: candidate high utility k-itemsets`
9 $k = k + 1$;
10 **end**
11 P = $\bigcup_{k=1 \ldots k} P_k$ ; `//` $P$ `: all candidate high utility itemsets`
12 Scan the database to calculate the utility of each itemset in $P$ ; `// PHASE 2`
13 return each itemset $X \in P$ such that $u(X) \geq minutil$;

Two-Phase is an important algorithm, since it is one of the first complete high utility itemset mining algorithm, and it has introduced the TWU upper-bound, used by most high utility itemset mining algorithms thereafter. However, Two-Phase suffers from important limitations with respect to performance. The first one is that because Two-Phase generates itemsets by combining itemsets without looking at the database, it can generate some patterns that do not even appear in the database. Thus, Two-Phase can spend a large amount of time processing itemsets that do not exist in the database. The second limitation is that Two-Phase repeatedly scans the database to calculate the TWU and utilities of itemsets, which is very costly. The third limitation is that using a breadth-first search can be quite costly in terms of memory as it requires at any moment to keep in the worst case all $k$-itemsets and $(k-1)$-itemsets in memory (for $k > 1$). Moreover, Two-Phase must keep all candidate high utility itemsets in memory before performing the second phase. But a huge amount of candidates can be generated by Two-Phase to find just a few high utility itemsets [79]. The reason is that the TWU is a loose upper-bound on the utility of itemsets. In subsequent studies, tighter upper-bounds have been designed, and techniques to decrease these upper-bounds. In terms of complexity, Two-Phase is based on Apriori. A very detailed complexity analysis of the Apriori algorithm has been done by Hegland [40]. Briefly, the time complexity of Apriori is $O(m^2 n)$, where $m$ is the number of distinct items and $n$ is the number of transactions. In terms of complexity, the main difference between Apriori and Two-Phase is that the

latter performs a second phase where the exact utility of each pattern is calculated by scanning the database. Various optimizations can be used to reduce the cost of the second phase such as storing itemsets in a hash-tree to avoid comparing each itemset with each transaction [2]. However, the second phase remains very costly [79, 90].

### 3.1.2 Pattern-growth Two Phase Algorithms

To address some of the drawbacks of the Two-Phase algorithm, several *pattern-growth* algorithms have been proposed such as IHUP [5], UP-Growth [79], HUP-Growth [52], PB [47] and MU-Growth [87]. The concept of *pattern-growth algorithm* was first used in frequent itemset mining algorithms such as FP-Growth [39], H-Mine [69] and LCM [81]. The main idea of pattern-growth algorithms is to scan a database to find itemsets, and thus avoid generating itemsets that do not appear in the database. Furthermore, to reduce the cost of scanning the database, pattern-growth algorithms have introduced compact database representations and the concept of *projected database* to reduce the size of databases as an algorithm explore larger itemsets.

All pattern-growth algorithms discussed in this chapter utilize a depth-first search rather than a breadth-first search to explore the search space of itemsets. The advantage of using the former instead of the latter is that less itemsets need to be kept in memory during the search. A *depth-first search algorithm* starts from each 1-itemset and then recursively try to append items to the current itemset to generate larger itemsets. For example, in the running example, a typical depth-first search algorithm would explore itemsets in that order: $\{a\}$, $\{a, b\}$, $\{a, b, c\}$, $\{a, b, c, d\}$, $\{a, b, c, d, e\}$, $\{a, b, c, e\}$, $\{a, b, d\}$, $\{a, b, d, e\}$, $\{a, b, e\}$, $\{a, c\}$, $\{a, c, d\}$, $\{a, c, d, e\}$, $\{a, c, e\}$, $\{a, d\}$, $\{a, d, e\}$, $\{a, e\}$, $\{b\}$, $\{b, c\}$, $\{b, c, d\}$, $\{b, c, d, e\}$, $\{b, c, e\}$, $\{b, d\}$, $\{b, d, e\}$, $\{b, e\}$, $\{c\}$, $\{c, d\}$, $\{c, d, e\}$, $\{c, e\}$, $\{d\}$, $\{d, e\}$, $\{e\}$.

The pseudocode of a typical two phase pattern-growth algorithm for high utility itemset mining is shown in Algorithm 2. It takes as input a quantitative transaction database $D$ and the *minutil* threshold. Without loss of generality, assume that there exists a total order on items $\prec$ such as the lexicographical order ($a \prec b \prec c \prec d \prec e$). The pattern-growth algorithm first creates a set $P$ to store candidate high utility itemsets (line 1). Then, the algorithm scans the database $D$ to calculate transaction utilities, denoted as $TUs$ (line 2). The algorithm then explores the search space using a depth-first search by recursively appending items according to the $\prec$ order to candidate high utility itemsets, to obtain larger candidate high utility itemsets. This process is done by a call to the RecursiveGrowth procedure, described in Algorithm 3. At the beginning, the RecursiveGrowth procedure considers that the current itemset $X$ is the empty set. The procedure scans the database $D$ to find the set $Z$ of all items in $D$ that are candidate high utility itemsets (lines 1 and 2). Then, for each such item $z$, the itemset $X \cup \{z\}$ is stored in the set of candidate high utility itemsets $P$ (line 4). Then, the pattern-growth procedure is called to perform a depth-first search to find larger frequent itemsets that are extensions of $X \cup \{z\}$ in the same way (line 6). However, it can be observed that not all items in $D$ can be appended to $X \cup \{z\}$ to generate larger itemsets. In fact, the itemset $X \cup \{z\}$ may not

even appear in all transactions of the database $D$. For this reason, a pattern-growth algorithm will create the projected database of the itemset $X \cup \{z\}$ (line 5) and will use this database to perform the depth-first search (line 6). This will allow reducing the cost of scanning the database. After recursively performing the depth-first search for all items, the set of all candidate high utility itemsets $P$ has been generated.

Then, Algorithm 2 performs a second phase in the same way as the previously described Two-Phase algorithm. The database is scanned to calculate the exact utility of each candidate high utility itemsets (line 4). Those having a utility not less than the *minutil* threshold are returned to the user (line 5).

Now, let's illustrate these steps in more details with an example. Consider the database of Tables 3 and 4 and assume that $minutil = 25$. In phase 1, the algorithm scans the database and finds that 1-itemsets $\{a\}$, $\{b\}$, $\{c\}$ and $\{e\}$, have TWU values of 55, 54, 84, 53, and 76, respectively. These itemsets are thus candidate high utility itemsets. The algorithm first considers the item $a$ to try to find larger candidate itemsets starting with the prefix $\{a\}$. The algorithm then builds the projected database of $\{a\}$ as shown in Table 8. The projected database of an item $i$ is defined as the set of transactions where $i$ appears, but where the item $i$ and items preceding $i$ according to the $\prec$ order have been removed. Then, to find candidate itemsets starting with $\{a\}$ containing one more item, the algorithm scans the projected database of $\{a\}$ and count the TWU of all items appearing in that database. For example, the TWU of items in the projected database of $\{a\}$ are: $\{b\} : 25$, $\{c\} : 55$ and $\{e\} : 47$. This means that the TWU of $\{a, b\}$ is 25, that the TWU of $\{a, c\}$ is 55, and that the TWU of $\{a, e\}$ is 47. Since these three itemsets have a TWU no less than *minutil*, these itemsets are candidate high utility itemsets and are next used to try to generate larger itemsets by performing the depth-first search starting from each itemset. The itemset $\{a, c\}$ is first considered. The algorithm builds the projected database of $\{a, c\}$ from the projected database of $\{a\}$. The projected database of $\{a, c\}$ is shown in Table 9. Then, the algorithm scans the projected database of $\{a, c\}$ to find items having a TWU no less than *minutil* in that database. This process will continue until all candidate high utility itemsets have been found by the depth-first search. Then, in phase 2, the database is scanned to calculate the exact utilities of all candidates found in phase 1. Then, itemsets having a utility less than *minutil* are eliminated. The remaining itemsets are output as the high utility itemsets. The result is shown in Table 5.

A major advantage of pattern-growth algorithms is that they only explore itemsets that actually appear at least once in the input database, contrarily to Apriori-based algorithms, which may generate patterns that do not appear in the database. Besides, the concept of projected database is also useful to reduce the cost of database scans, since projected databases are smaller than the original database. A common question about the concept of projected database is: is it costly to create all these copies of the original database? The answer is no if an optimization called *pseudo-projection* is used, which consists of implementing a projected database as a set of pointers on the original database rather than as a copy [69, 81]. For example, Fig. 3 shows the pseudo-projected database of $\{a, c\}$, which is equivalent to the projected database of Table 4, excepts that it is implemented using three pointers on the original database,

to avoid creating a copy of the original database. Note that many other optimizations can also be integrated in pattern-growth algorithms. For example, the IHUP [5], UP-Growth [79], HUP-Growth [52], and MU-Growth [87] algorithms utilize prefix-tree structures for representing projected-databases and reduce memory usage. These structures extends the FP-tree structure used in frequent itemset mining by the FPGrowth algorithm [39]. The main differences between these algorithms lies in the use of various strategies to reduce the TWU upper-bounds on the utility. Among two phase algorithms, UP-Growth is one of the fastest. It was shown to be up to 1,000 times faster than Two-phase and IHUP. More recent two phase algorithms such as PB and MU-Growth have introduced various optimizations and different design but only provide a small speed improvement over Two-Phase or UP-Growth (MU-Growth is reported to be only up to 15 times faster than UP-Growth).

Although two phase algorithms have been well-studied and introduced many key ideas, they remain inefficient. As explained, two phase algorithm mines high utility itemsets in two phases. In phase 1, a set of candidates is found. Then, in phase 2, the utility of these candidates is calculated by scanning the database. Then, low-utility itemsets are filtered and the high utility itemsets are returned to the user. This approach is inefficient because the set of candidate itemsets found in phase 1 can be very large and performing the second phase to evaluate these candidates is very costly [79, 90]. In the worst case, all candidate itemsets are compared to all transactions of the database during the second phase. Thus, the performance of two phase algorithms is highly influenced by the number of candidates generated to find the actual high utility itemsets. To reduce the number of candidates, various strategies have been design to decrease the TWU upper-bound, and thus prune more candidates [5, 52, 79, 87]. But to address the fundamental problem of two phase algorithms, which is to generate candidates, one-phase algorithms have been designed, which are described next.

**Algorithm 2:** A two phase pattern-growth algorithm

**input :** $D$: a quantitative transaction database, $minutil$: the minimum utility threshold
**output :** the set of high utility itemsets

1 $P = \emptyset$ ; `// P: all candidate high utility itemsets`
2 Scan the database to calculate $TUs$, the transaction utilities of transactions in $D$;
3 `RecursiveGrowth` ($TUs$, $D$, $\emptyset$, $minutil$, $P$); `// PHASE 1`
4 Scan the database to calculate the utility of each itemset in $P$; `// PHASE 2`
5 return each itemset $X \in P$ such that $u(X) \geq minutil$;

## *3.2 One Phase Algorithms*

The second major breakthrough in high utility itemset mining has been the design of algorithms that do not generate candidates. These *one phase algorithms* immediately

**Algorithm 3:** The RecursiveGrowth procedure

**input :** $TUs$: the transaction utilities in the original database, $D$: a quantitative transaction database, $X$: the current itemset, $minutil$: the minimum utility threshold, $P$: a set to store candidate high utility itemsets
**output :** the set of high utility itemsets

```
1 Scan the database D to calculate the TWU of each item in I using TUs;
2 W = {i|i ∈ I ∧ TWU({i}) ≥ minutil};     // W: candidate high utility
    1-itemsets in D
3 foreach item z ∈ W do
4     Add X ∪ {z} to P;
5     D' = Projection(D, z) ;     // Create projected database of X ∪ {z}
6     RecursiveGrowth (TUs, D', {z}, minutil, P);   // recursive call to
        extend X ∪ {z}
7 end
```

Table 9: The projected database of $\{a, c\}$

| TID | Transaction |
|---|---|
| $T_0$ | $(b, 5), (c, 1), (d, 3), (e, 1)$ |
| $T_2$ | $(c, 1), (d, 1)$ |
| $T_3$ | $(c, 6), (e, 2)$ |

| TID | Transaction |
|---|---|
| $T_0$ | $(d, 3), (e, 1)$ |
| $T_2$ | $(d, 1)$ |
| $T_3$ | $(e, 2)$ |

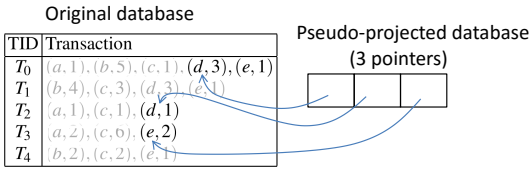


Fig. 3: The pseudo-projected database of $\{a, c\}$

calculate the utility of each pattern considered in the search space. Thus, an itemset can be immediately identified as a low utility or high utility itemset, and candidates do not need to be stored in memory. The concept of one phase algorithm was first published in HUI-Miner [58, 71], and then in the d$^2$HUP [60] algorithm. Then, improved and more efficient one phase algorithms have been designed such as FHM [31], mHUIMiner [70], ULB-Miner [17], HUI-Miner* [71] and EFIM [94]. Besides the novelty of discovering high utility itemsets in one phase, one phase algorithms have also introduced novel upper-bounds on the utility of itemsets that are based on the exact utility of each itemset, and can thus prune a larger part of the search space compared to the TWU measure. These upper-bounds include the *remaining utility* [58, 60], and newer measures such as the *local-utility* and *sub-tree utility* [94]. The next subsections gives an overview of one phase algorithms.

### 3.2.1 The FHM Algorithm

One of the most popular type of high utility itemset mining algorithms are those based on the *utility-list* structure. This structure was introduced in the HUI-Miner algorithm [58] by generalizing the *tid-list* structure [91] used in frequent itemset mining. Then, faster utility-list based algorithms have been proposed such as FHM [31], mHUIMiner [70] and ULB-Miner [17], and extensions have been proposed for several variations of the high utility itemset mining problem. The reason for the popularity of utility-list based algorithms is that they are fast and easy to implement. This subsection describes the FHM algorithm [31] as a representative utility-list based algorithm, which was shown to be up to seven times faster than HUI-Miner, and as been used and extended by many researchers.

FHM is a one-phase algorithm that performs a depth-first search to explore the search space of itemsets. During the search, the FHM algorithm creates a *utility-list* for each visited itemset in the search space. The utility-list of an itemset stores information about the utility of the itemset in transactions where it appears, and information about the utilities of remaining items in these transactions. Utility-lists allows to quickly calculate the utility of an itemset and upper-bounds on the utility of its super-sets, without scanning the database. Moreover, utility-lists of $k$-itemsets $k > 1$) can be quickly created by joining utility-lists of shorter patterns. The utility-list structure is defined as follows.

**Definition 8 (Utility-list).** *Let there be an itemset $X$ and a quantitative database $D$. Without loss of generality, assume that a total order $\succ$ is defined on the set of items $I$ appearing in that database. The* utility-list *$ul(X)$ of $X$ in a quantitative database $D$ is a set of tuples such that there is a tuple $(tid, iutil, rutil)$ for each transaction $T_{tid}$ containing $X$. The $iutil$ element of a tuple is the utility of $X$ in $T_{tid}$. i.e., $u(X, T_{tid})$. The $rutil$ element of a tuple is defined as $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$.*

For example, assume that $\succ$ is the alphabetical order. The utility-lists of $\{a\}$, $\{d\}$ and $\{a, d\}$ are shown in Fig. 4. Consider the utility-list of $\{a\}$. It contains three rows (tuples) corresponding to transactions $T_0$, $T_2$ and $T_3$ since $\{a\}$ appears in these three transactions. The second column of the utility list ($iutil$ values) of $\{a\}$ indicates that the utility of $\{a\}$ in $T_0$, $T_2$ and $T_3$ are 5, 5, and 10, respectively. The third column of the utility list of $\{a\}$ indicates that the $rutil$ values of $\{a\}$ for transactions $T_0$, $T_2$ and $T_3$ are 20, 3, and 10, respectively.

The utility-list of $\{a\}$

| tid | iutil | rutil |
|---|---|---|
| $T_0$ | 5 | 20 |
| $T_2$ | 5 | 3 |
| $T_3$ | 10 | 12 |

The utility-list of $\{d\}$

| tid | iutil | rutil |
|---|---|---|
| $T_0$ | 6 | 3 |
| $T_1$ | 6 | 3 |
| $T_2$ | 2 | 0 |

The utility-list of $\{a, d\}$

| tid | iutil | rutil |
|---|---|---|
| $T_0$ | 11 | 3 |
| $T_2$ | 7 | 0 |

Fig. 4: The utility-lists of $\{a\}$, $\{d\}$ and $\{a, d\}$

The FHM algorithm scans the database once to create the utility-lists of 1-itemsets (single items). Then, the utility-lists of larger itemsets are constructed by joining the utility-lists of smaller itemsets. The join operation for single items is performed as follows. Consider two items $x, y$ such that $x \succ y$, and their utility-lists $ul(\{x\})$ and $ul(\{y\})$. The utility-list of $\{x, y\}$ is obtained by creating a tuple $(ex.tid, ex.iutil + ey.iutil, ey.rutil)$ for each pairs of tuples $ex \in ul(\{x\})$ and $ey \in ul(\{y\})$ such that $ex.tid = ey.tid$. The join operation for two itemsets $P \cup \{x\}$ and $P \cup \{y\}$ such that $x \succ y$ is performed as follows. Let $ul(P)$, $ul(\{x\})$ and $ul(\{y\})$ be the utility-lists of $P$, $\{x\}$ and $\{y\}$. The utility-list of $P \cup \{x, y\}$ is obtained by creating a tuple $(ex.tid, ex.iutil + ey.iutil - ep.iutil, ey.rutil)$ for each set of tuples $ex \in ul(\{x\})$, $ey \in ul(\{y\})$, $ep \in ul(P)$ such that $ex.tid = ey.tid = ep.tid$. For example, the utility-list of $\{a, d\}$ can be obtained by joining the utility-lists of $\{a\}$ and $\{d\}$ (depicted in Fig. 4), without scanning the database.

The utility-list structure of an itemset is very useful, as it allows to directly obtain the utility of an itemset without scanning the database.

*Property 5 (Calculating the utility of an itemset using its utility-list).* Let there be an itemset $X$. The sum of the *iutil* values in its utility-list $ul(X)$ is equal to the utility of $X$ [58]. In other words, $u(X) = \sum_{e \in ul(X)} e.iutil$.

For example, the utility of the itemset $\{a, d\}$ is equal to the sum of the values in the *iutil* column of its utility-list (depicted in Fig. 4). Hence, by looking at the utility-list of $\{a, d\}$, it is found that its utility is $u(\{a, d\}) = 11 + 7 = 18$.

The utility-list of an itemset is also used to prune the search space based on the following definition and property.

**Definition 9 (Remaining utility upper-bound).** Let $X$ be an itemset. Let the *extensions* of $X$ be the itemsets that can be obtained by appending an item $y$ to $X$ such that $y \succ i$, $\forall i \in X$. The *remaining utility upper-bound of* $X$ is the sum of the *iutil* and *rutil* values in its utility-list $ul(X)$. Formally, this upper-bound is defined as $reu(X) = \sum_{e \in ul(X)} (e.iutil + e.rutil)$. The value $reu(X)$ is an upper-bound on the utility of $X$ and all its extensions [58]. In other words, the relationship $u(Y) \leq reu(X)$ holds for any itemset $Y$ that is an extension of $X$.

For example, consider calculating the remaining utility upper-bound of the itemset $\{a, d\}$ using its utility-list (depicted in Fig. 4). The upper-bound is the sum of the values in the *iutil* and *rutil* columns of its utility-list, that is $reu(\{a, d\}) = 11+7 = 18$. It is thus known that the itemset $\{a, d\}$ and all its extensions such as $\{a, d, e\}$ cannot have a utility greater than 18. If we assume that $minutil = 25$, as in the running example, these itemsets can thus be pruned from the search space, as they will be low-utility itemsets. This is formalized by the following property.

*Property 6 (Pruning search space using a utility-list).* Let $X$ be an itemset. If the sum of *iutil* and *rutil* values in $ul(X)$ is less than *minutil* (i.e. $reu(X) < minutil$), $X$ and its extensions are low utility itemsets [58].

The main procedure of FHM (Algorithm 4) takes a quantitative transaction database and a *minutil* threshold as input. FHM first scans the database to calculate the TWU of each item. Then, the algorithm identifies the set $I^*$ of all items having a TWU no less than *minutil* (other items are ignored since they cannot be part of a high-utility itemset by Property 4). The TWU values of items are then used to establish a total order $\succ$ on items, which is the order of ascending TWU values (as suggested in [58]). A database scan is then performed. During this database scan, items in transactions are reordered according to the total order $\succ$, the utility-list of each item $i \in I^*$ is built and a structure named EUCS (Estimated Utility Co-Occurrence Structure) is built [31]. This latter structure is defined as a set of triples of the form $(a, b, c) \in I^* \times I^* \times \mathbb{R}$. A triple (a,b,c) indicates that TWU($\{a, b\}$) = $c$. The EUCS can be implemented as a triangular matrix that stores these triples for all pairs of items. For example, the EUCS for the running example is shown in Fig. 5. The EUCS is very useful as it stores the TWU of all pairs of items, an information that will be later used for pruning the search space. For instance, the top-left cell indicates that $TWU(\{a, b\}) = 25$. Building the EUCS is very fast (it is performed with a single database scan) and occupies a small amount of memory, bounded by $|I^*| \times |I^*|$. The reader is referred to the paper about FHM [31] for more details about the construction of this structure and implementation optimizations. After the construction of the EUCS, the depth-first search exploration of itemsets starts by calling the recursive procedure $FHMSearch$ with the empty itemset $\emptyset$, the set of single items $I^*$, *minutil* and the EUCS structure.

| **Item** | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| b | 30 | | | | | |
| c | 65 | 61 | | | | |
| d | 38 | 50 | 58 | | | |
| e | 57 | 61 | 88 | 50 | | |
| f | 30 | 30 | 30 | 30 | 30 | |
| g | 27 | 11 | 38 | 0 | 38 | 0 |

Fig. 5: The Estimated-Utility Cooccurrence Structure

**Algorithm 4:** The FHM algorithm

**input :** $D$: a transaction database, $minutil$: a user-specified threshold
**output :** the set of high-utility itemsets

1 Scan $D$ to calculate the TWU of single items;
2 $I^* \leftarrow$ each item $i$ such that TWU($i$) $\geq minutil$;
3 Let $\succ$ be the total order of TWU ascending values on $I^*$;
4 Scan $D$ to built the utility-list of each item $i \in I^*$ and build the $EUCS$;
5 Output each item $i \in I^*$ such that SUM($\{i\}.utilitylist.iutils$) $\geq minutil$;
6 `FHMSearch` ($\emptyset$, $I^*$, $minutil$, $EUCS$);

**Algorithm 5:** The $FHMSearch$ procedure

**input :** $P$: an itemset, *ExtensionsOfP*: a set of extensions of $P$, $minutil$: a user-specified threshold, $EUCS$: the $EUCS$ structure
**output :** the set of high-utility itemsets

```
1  foreach itemset Px ∈ ExtensionsOfP do
2      if SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.rutils) ≥ minutil then
3          ExtensionsOfPx ← ∅;
4          foreach itemset Py ∈ ExtensionsOfP such that y ≻ x do
5              if ∃(x, y, c) ∈ EUCS such that c ≥ minutil then
6                  Pxy ← Px ∪ Py;
7                  Pxy.utilitylist ← Construct (P, Px, Py);
8                  ExtensionsOfPx ← ExtensionsOfPx ∪ Pxy;
9                  if SUM(Pxy.utilitylist.iutils) ≥ minutil then output Px;
10             end
11         end
12         FHMSearch (Px, ExtensionsOfPx, minutil);
13     end
14 end
```

**Algorithm 6:** The Construct procedure

**input :** $P$: an itemset, $Px$: the extension of $P$ with an item $x$, $Py$: the extension of $P$ with an item $y$
**output :** the utility-list of $Pxy$

```
1  UtilityListOfPxy ← ∅;
2  foreach tuple ex ∈ Px.utilitylist do
3      if ∃ey ∈ Py.utilitylist and ex.tid = exy.tid then
4          if P.utilitylist ≠ ∅ then
5              Search element e ∈ P.utilitylist such that e.tid = ex.tid.;
6              exy ← (ex.tid, ex.iutil + ey.iutil − e.iutil, ey.rutil);
7          end
8          else exy ← (ex.tid, ex.iutil + ey.iutil, ey.rutil);
9          UtilityListOfPxy ← UtilityListOfPxy ∪ {exy};
10     end
11 end
12 return UtilityListPxy;
```

The $FHMSearch$ procedure (Algorithm 5) takes as input (1) an itemset $P$, (2) extensions of $P$ having the form $Pz$ meaning that $Pz$ was previously obtained by appending an item $z$ to $P$, (3) $minutil$ and (4) the EUCS. The search procedure operates as follows. For each extension $Px$ of $P$, if the sum of the $iutil$ values of the utility-list of $Px$ is no less than $minutil$, then $Px$ is a high-utility itemset and it is output (cf. Property 4). Then, if the sum of $iutil$ and $rutil$ values in the utility-list of $Px$ are no less than $minutil$, it means that extensions of $Px$ should be explored. This is performed by merging $Px$ with all extensions $Py$ of $P$ such that $y \succ x$ to form extensions of the form $Pxy$ containing $|Px| + 1$ items. The utility-list of $Pxy$ is then constructed by calling the $Construct$ procedure to join the utility-lists of $P$, $Px$ and

$Py$. Then, a recursive call to the $Search$ procedure with $Pxy$ is done to calculate its utility and explore its extension(s). Since the $FHMSearch$ procedure starts from single items, it recursively explores the search space of itemsets by appending single items and it only prunes the search space based on Property 6. It can be proven that this procedure is correct and complete to discover all high-utility itemsets.

The utility-list structure used by the FHM algorithm is said to be a *vertical database representation*. A vertical database representation indicates the list of transactions where each itemset appears. This is different from a traditional horizontal database, where each entry is a transaction indicating the items that it contains.

A benefit of utility-list based algorithms is that they are easy to implement, and efficient. It was shown that utility-list based algorithms can be more than two order of magnitude faster than two-phase algorithms [31, 58, 94]. However, utility-list based algorithms have important drawbacks. First, these algorithms may explore some itemsets that never appear in the database since itemsets are generated by combining itemsets, without reading the database. Hence, these algorithms may waste a lot of time constructing the utility-lists of itemsets that do not exist. Second, these algorithms sometimes consume a lot of memory, since a utility-list must be built for each visited itemset in the search space. The utility-list of an itemset can be quite large. In the worst case, it contains a tuple for each transaction of the database. The join operation can also be especially costly as two or three utility-lists must be compared to construct the utility-list of each $k$-itemset ($k > 1$).

To reduce the memory requirement of utility-list based algorithm, the ULB-Miner [17] algorithm was recently proposed by extending HUI-Miner [58] and FHM [31]. ULB-Miner utilizes a buffer to reuse the memory for storing utility-lists. This strategy was shown to improve both the runtime and memory usage. Another improvement of HUI-Miner is HUI-Miner* [71], which relies on an improved utlity-list* structure to speed-up HUI-Miner.

### 3.2.2 Pattern-growth One-phase Algorithms

Pattern-growth one-phase algorithms address several limitations of utility-list based algorithms. They explore the search space by reading the database, and thus only consider itemsets that exist in the database. The $d^2$HUP algorithm [60] is the first such algorithm. It performs a depth-first search, and represents the database and projected databases using an hyper-structure, that is similar to the H-Mine algorithm [69] in frequent pattern mining. Although this algorithm was shown to be faster than several other algorithms, the process of creating and updating the hyperstructure can be quite costly.

Recently, the EFIM algorithm was proposed [94], inspired by the LCM algorithm in frequent itemset mining. It is designed to process each itemset in the search space in linear time and space. EFIM performs a depth-first search using an horizontal database representation to reduce memory usage. Moreover, it introduced two novel upper-bound called the *local-utility* and *subtree-utility* to effectively reduce the search space. EFIM also introduced a novel array-based utility counting tech-

nique named *Fast Utility Counting* to calculate these upper-bounds in linear time and space using a reusable array structure. Moreover, to reduce the cost of database scans, EFIM integrates efficient database projection and transaction merging techniques named *High-utility Database Projection* (HDP) and *High-utility Transaction Merging* (HTM), also performed in linear time. It was shown that EFIM is in general two to three order of magnitudes faster than the $d^2$HUP, HUI-Miner, FHM and UPGrowth algorithms, while having an often much lower memory consumption.

### *3.3 A Comparison of High Utility Itemset Mining Algorithms*

This section has provided an overview of some popular high utility itemset mining algorithms. Table 10 provides a comparison of their characteristics in terms of type of search (breadth-first search or depth-first search), the number of phases (one or two), database representation (horizontal or vertical), and the most similar frequent itemset mining algorithm.

Table 10: Algorithms for high utility itemset mining

| Algorithm | Search type | Nb of phases | DB representation | Extends |
|---|---|---|---|---|
| Two-Phase [59] | breadth-first | Two | Horizontal | Apriori [2] |
| PB [47] | breadth-first | Two | Horizontal | Apriori [2] |
| IHUP [5] | depth-first | Two | Horizontal (prefix-tree) | FP-Growth [39] |
| UPGrowth(+) [79] | depth-first | Two | Horizontal (prefix-tree) | FP-Growth [39] |
| HUP-Growth [52] | depth-first | Two | Horizontal (prefix-tree) | FP-Growth [39] |
| MU-Growth [87] | depth-first | Two | Horizontal (prefix-tree) | FP-Growth [39] |
| D2HUP [60] | depth-first | One | Vertical (hyperstructure) | H-Mine [69] |
| HUI-Miner [58] | depth-first | One | Vertical (utility-lists) | Eclat [91] |
| FHM [31] | depth-first | One | Vertical (utility-lists) | Eclat [91] |
| mHUIMiner [70] | depth-first | One | Vertical (utility-lists) | Eclat [91] |
| HUI-Miner* [71] | depth-first | One | Vertical (utility-lists*) | Eclat [91] |
| ULB-Miner [17] | depth-first | One | Vertical (buffered utility-lists) | Eclat [91] |
| EFIM [94] | depth-first | One | Horizontal (with merging) | LCM [81] |

## 4 Extensions of the Problem

Even though, high utility itemset mining has numerous applications, it also has some limitations for some applications. This section presents an overview of extensions of the high utility itemset mining problem that are designed to address some of these limitations. Most of the algorithms for these extensions are based on the algorithms described in the previous section.

### *4.1 Concise Representations of High Utility Itemsets*

A first limitation of high utility itemset mining algorithms is that they can show a large number of patterns to the user if the minimum utility threshold is set too low. In that case, it can be very difficult and time-consuming for humans to analyze the patterns found. Generally, when a huge number of patterns is found, many of them can be viewed as redundant. To present a small set of meaningful patterns to the user, researchers have designed algorithms to extract *concise representations* of high utility itemsets. A concise representation is a set of itemsets that summarizes all high utility itemsets. A benefit of concise representations is that they can be several orders of magnitude smaller than the set of all high utility itemsets [30, 90]. Besides, mining concise representations of high utility itemsets can be much faster than discovering all high utility itemsets (HUIs) [90]. There are four main representations of high utility itemsets.

- *Closed High Utility Itemsets* [14, 35, 88, 90]. An itemset is a closed high utility itemset (CHUI), if it has no supersets appearing in the same transactions (having the same support), i.e. $CHUIs = \{X | X \in HUIs \wedge \nexists Y \in HUIs$ such that $X \subset Y \wedge sup(X) = sup(Y)\}$. In the example of Table 5, out of eleven frequent itemsets, only seven of them are closed: $\{a, c\}$, $\{a, c, e\}$, $\{b, c, d, e\}$, $\{b, c, e\}$, $\{b, d\}$ $\{c, e\}$ and $\{a, b, c, d, e\}$. Thus, the number of CHUIs can be much less than the number of HUIs. Moreover, if additional information is stored about CHUIs, a representation called closed+ high utility itemset is obtained, which is a *lossless* representation of all high utility itemset itemsets [88, 90]. Hence, using closed+ itemsets the information about all high utility itemsets, including their utility, can be recovered without scanning the database. In the context of analyzing customer transactions, CHUIs are interesting as they represent the largest sets of items common to groups of customers, that yield a high profit.
- *Maximal itemsets* [75, 89]. Maximal high utility itemsets (MHUIs) are the high utility itemsets that do not have supersets that are high utility, i.e. $MHUIs = \{X | X \in HUIs \wedge \nexists X \in HUIs$ such that $X \subset Y\}$. In other words, maximal itemsets are the largest high utility itemsets. The set of maximal itemsets is a subset of the set of closed itemsets ($MHUIs \subseteq CHUIs \subseteq HUIs$), and thus can further reduce the number of itemsets presented to the user. However, maximal itemsets are not a lossless representation of all HUIs. In other words, MHUIs cannot be used to recover all high utility itemsets and their utility without scanning the database. In the example of Table 1, there are only one maximal itemset: $\{a, b, c, d, e\}$.
- *Generators of high-utility itemsets* [30]. An itemset $X$ is a *generator of high-utility itemset* (GHUI) if an only if (1) there exists no itemset $Y \subset X$, such that $sup(X) = sup(Y)$, and (2) there exists an itemset $Z$ such that $X \subseteq Z$ and $u(Z) \geq minutil$ [30]. The set of generator of high utility itemsets is always of equal size as or larger than the set of closed and maximal high utility itemsets. But the set of generators is interesting according to the Minimum Description Length principle [8] since it represents the smallest sets of items that are common to transactions who contain a high utility itemset. For example, in market basket

analysis, a GHUI is the smallest set of items common to a group of customers who bought a set of items that generates a high profit.

- *Minimal high utility itemsets* [29]. An itemset $X$ is a *minimal high-utility itemset* (MinHUI) iff $u(X) \geq minutil$ and there does not exist an itemset $Y \subset X$ such that $u(Y) \geq minutil$. This proposed representation is the opposite of maximal HUIs, i.e. it consists of the smallest sets of items that generate a high profit rather than the largest. The assumption is that the smallest itemsets are often the most interesting. For example, for marketing purpose, a retailer may be more interested in finding the smallest sets of items that generate a high profit, since it is easier to co-promote a small set of items targeted at many customers rather than a large set of items targeted at few customers. In the running example, there are five MinHUIs: $\{b, c\}$, $\{b, d\}$, $\{b, e\}$, $\{a, c\}$ and $\{c, e\}$.

To better illustrate the relationship between the HUIs, CHUIs, MinHUIs and GHUIs, Fig. 6 presents an illustration of these various types of patterns, for the running example. In this figure, all equivalence classes containing at least a HUI are represented. An *equivalence class* is a set of itemsets supported by the same set of transactions, ordered by the subset relation. For example, $\{\{a, e\}, \{a, c, e\}\}$ is the equivalence class of itemsets appearing in transactions $T_0$ and $T_2$. Formally, the relationship between these various sets of HUIs are the following: $MinHUIs \subseteq HUIs \subseteq 2^I$, $MaxHUIs \subseteq CHUIs \subseteq HUIs \subseteq 2^I$, and $GHUIs \subseteq 2^I$ [29][1].

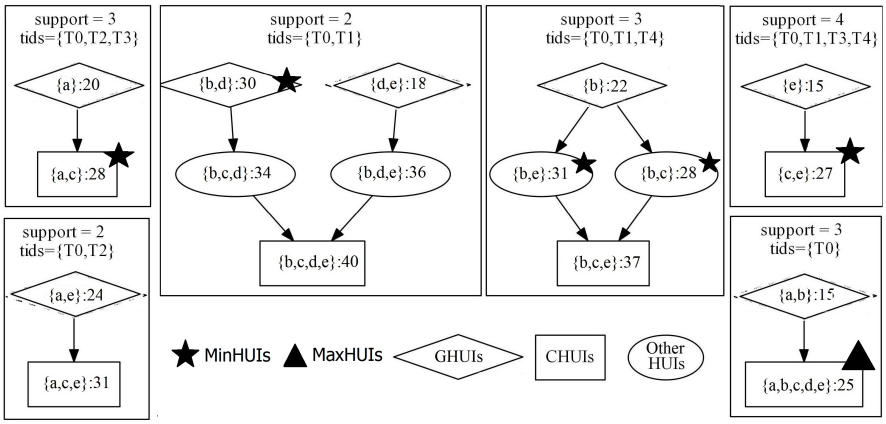


Fig. 6: HUIs and their equivalence classes (represented using Hasse diagrams)

Several algorithms have been proposed to efficiently discover the above concise representations of high utility itemsets. Table 11 provides an overview of these algorithms, and their characteristics. In many cases, mining concise representations

[1] The notation $2^I$ denotes all itemsets that can be created using items from a set of items $I$. It is also called the *powerset* of $I$.

can be much faster than discovering all high utility itemsets since fewer itemsets are found.

Table 11: Algorithms for mining concise representations of high utility itemsets

| Algorithm | Patterns | Nb of phases | DB representation | Extends |
|---|---|---|---|---|
| MinFHM [29] | MinHUIs | One | Vertical (utility-lists) | FHM [31] |
| GHUI-Miner [30] | GHUIs | One | Vertical (utility-lists) | FHM [31] |
| CHUD [14] | CHUIs | Two | Vertical (utility-lists) | DCI_Closed [63] |
| CHUI-Miner [14] | CHUIs | One | Vertical (utility-lists) | DCI_Closed [63] |
| CLS-Miner [14] | CHUIs | One | Vertical (utility-lists) | FHM [31] |
| EFIM-Closed [94] | CHUIs | One | Horizontal (with merging) | EFIM [94] |
| GUIDE [75] | MHUIs | One | Stream | UPGrowth [79] |
| CHUI-Mine [89] | MHUIs | One | Vertical (utility-lists) | HUI-Miner[58, 71] |

## *4.2 Top-k High Utility Itemset Mining*

Another limitation of traditional high utility itemset mining algorithms is that how the *minutil* threshold is set greatly influences the execution time, memory usage, and number of patterns shown to the user. On one hand, if a user sets the *minutil* threshold too low, a huge number of patterns may be found and algorithms may become slow and consume a huge amount of memory. On the other hand, if a user sets the *minutil* threshold too high, few or no patterns may be found. To address this issue, the problem of *top-k high utility itemset mining* was proposed [80], where the user wants to discover the $k$ itemsets having the highest utility in a quantitative database. For this problem, the *minutil* parameter is replaced by a parameter $k$. For example, if $k = 3$ for the running example, the top-$k$ high utility itemsets are $\{b, c, d\}$, $\{b, c, e\}$, and $\{b, c, d, e\}$ with utilities of 34, 37 and 40, respectively.

A top-$k$ high-utility itemset mining algorithm typically works as follows. It initially sets an internal *minutil* threshold to 0, and starts to explore the search space. Then, as soon as $k$ high utility itemsets are found, the internal *minutil* threshold is raised to the utility of the pattern having the lowest utility among the current top-$k$ patterns. Then, the search continues and for each high utility itemset found, the set of the current top-$k$ pattern is updated as well as the internal *minutil* threshold. When the algorithm terminates, the set of the top-$k$ high utility itemsets is returned to the user. The problem of top-$k$ high utility itemset mining is more difficult than the problem of high utility itemset mining since the former must start by considering that $minutil = 0$. Several high utility itemset mining algorithms have been proposed. A comparison of the characteristics of the main algorithms is provided in Table 12.

Table 12: Algorithms for mining the top-k high utility itemsets

| Algorithm | Search type | Nb of phases | DB representation | Extends |
|---|---|---|---|---|
| TKU [80] | Depth-first | Two | Horizontal (prefix-tree) | UP-Growth [79] |
| TKO [80] | Depth-first | One | Vertical (utility-lists) | HUI-Miner [58, 71] |
| REPT [72] | Depth-first | One | Horizontal (prefix-tree) | MU-Growth [87] |
| kHMC [18] | Depth-first | One | Vertical (utility-lists) | FHM [31] |

## *4.3 High Utility Itemset Mining with the Average Utility Measure*

Besides the standard utility measure presented in Definition 4, alternative utility measures have been studied [83]. One of the most popular alternative measure is the *average utility measure* [41]. It is based on the observation that larger itemsets tend to have a higher utility than smaller itemsets on some datasets (although this is not always true). To address this issue, the average utility measure divides the utility of an itemset by its length. Formally:

**Definition 10 (Average utility measure).** The *average utility of an itemset* $X$ in a quantitative database $D$ is denoted as $au(X)$ and defined as $au(X) = u(X)/|X|$.

**Definition 11 (High average utility itemset mining).** The problem of *high average utility itemset mining* in a quantitative database $D$ is to discover all itemsets having an average utility no less than a user-specified minimum average utility threshold $minAvgUtil$ [41].

For example, if $minAvgUtil = 13$ for the database of the running example, eight high average utility itemsets are found, depicted in Table 13.

Table 13: The high average utility itemsets for $minAvgUtil = 13$

| Itemset | Average utility | Itemset | Average utility |
|---|---|---|---|
| $\{a\}$ | 15 | $\{b, c\}$ | 14 |
| $\{b\}$ | 22 | $\{b, d\}$ | 15 |
| $\{e\}$ | 15 | $\{b, e\}$ | 15.5 |
| $\{a, c\}$ | 14 | $\{c, e\}$ | 13.5 |

Several algorithms have been proposed for high average utility itemset mining. A comparison of the main algorithms is provided in Table 14. The most efficient algorithm is to our knowledge dHAUIM [78]. For high utility average itemset mining, many upper-bounds on the average utility have been designed.

A main difference between dHAUIM and previous algorithms is the representation of average utility and its upper bounds designed using a novel vertical form instead of the traditional horizontal form.

To explain in more details these two different representations, it is first observed that to reduce calculation time, the utility of each item $j$ in each transaction can be

Table 14: Algorithms for mining the high average utility itemsets

| Algorithm | Search type | Nb of phases | DB representation | Extends |
|---|---|---|---|---|
| TPAU [41] | Breadth-first | Two | Horizontal (prefix-tree) | Two-Phase [59] |
| PBAU [45] | Depth-first | Two | Vertical (index table) | Two-Phase [59] |
| HAUI-tree [62] | Depth-first | Two | Horizontal (prefix-tree) | FP-Growth [39] |
| HAUI-Miner [53] | Depth-first | One | Horizontal (prefix-tree) | FHM [31] |
| EHAUPM [55] | Depth-first | One | Vertical (utility-lists) | FHM [31] |
| MHAI [85] | Depth-first | One | Vertical (utility-lists) | |
| dHAUIM [78] | Depth-first | One | Vertical (utility-lists) | |

precalculated by multiplying the internal utility by the external utility. The result is an *integrated utility matrix* $Q$. For instance, the matrix $Q$ obtained by transforming the database of the running example is shown in Table 15. In this matrix, each row represents a transaction and each column represents an item. The matrix entry at the $i^{th}$ row and $j^{th}$ column is denoted as $q_{ij}$ and defined as $q_{ij} = u(i, T_j)$. For an integrated matrix of size $n$ by $m$, having $n$ transactions and $m$ items, let $V = \{1, \ldots, n\}$ and $W = \{1, \ldots, m\}$ be the set of indices for the columns and rows, respectively.

Table 15: The integrated utility matrix for the running example

| Transactions | Items | | | | |
|---|---|---|---|---|---|
| | $a$ | $b$ | $c$ | $d$ | $e$ |
| $T_1$ | 5 | 10 | 1 | 6 | 3 |
| $T_2$ | 0 | 8 | 3 | 6 | 3 |
| $T_3$ | 5 | 0 | 1 | 2 | 0 |
| $T_4$ | 10 | 0 | 6 | 0 | 6 |
| $T_5$ | 0 | 4 | 2 | 0 | 3 |

The utility or average-utility of an itemset is traditionally calculated using utility values by considering each matrix line (transaction). Thus, the utility is said to be represented in *horizontal* form. Most upper-bounds on the average-utility are also computed in horizontal form. In fact, most upper-bounds are calculated using the *remaining maximum utility* $remu$ in each line (transaction). For example, some popular upper-bounds on the average-utility, named *auub* [46], *aub* and *lub* [55] are defined as follows:

$$auub(X) = \sum_{T_i \in g(X)} max\{q_{ij}, 1 \leq j \leq m\}$$
$$aub(X) = \sum_{T_i \in g(X)} max\{q_{ij}, j \geq minInd(X)\},$$
$$lub(X) = au(X) + remu(X).$$

In these definitions, $remu(X) = \sum_{T_i \in g(X)} remu(X, T_i)$ and $remu(X, T_i) = max\{q_{ij}, j > maxInd(X)\}$ is called the *remaining maximum utility* of $X$ in each $T_i \in g(X)$, and $maxInd(X) = max\{k | k^{th}$ item in $X\}$ and $minInd(X) = min\{k | k^{th}$ item in $X\}$ are the maximum and minimum indices of items of $X$ in the integrated matrix (or the indices of its last and first items), respectively.

By considering a *vertical* form, we can represent the utility of an itemset $X$ as $u(X) = \sum_{j \in X} v_j(X)$, where $v_j(X) = \sum_{T_i \in g(X)} q_{i,j}$ is the utility of item $j$ in $X$ and is computed based on the $j^{th}$ column of $Q$. An interesting observation based on this vertical perspective is that, for any matrix $Q$ and two non-empty index subsets $V' \subseteq V$ and $W' \subseteq W$, we always have the following inequality:

$$max\{\sum_{i \in V'} q_{ij},\ j \in W'\} \ \leq \ \sum_{i \in V'} max\{q_{ij},\ j \in W'\}$$

In other words, intuitively, the maximum of the sums by column is no greater than the sum of the maximums by line. This observation [78] is very useful, as it allows to easily design many new upper-bounds using the vertical form that are tighter than previous ones. In this context, for any two upper-bounds on the average utility, $ub_1$ and $ub_2$, $ub_1$ is said to be tighter than $ub_2$, which is denoted as $ub_1 \ll ub_2$, if $ub_1(X) \leq ub_2(X)$, for any itemset $X$; and $ub_1$ is said to be strictly tighter than $ub_2$ if $ub_1 \ll ub_2$ and there exists an itemset $Y$ such that $ub_1(Y) < ub_2(Y)$.

Based on the above obervation, three tighter new upper-bounds [78] were proposed inspired by the *auub*, *aub* and *lub* upper-bounds. They are defined and denoted as follows:

$$\overline{aub}_1(X) \ = \ max\{v_j(X) \mid j \ \geq \ 1\},$$
$$\overline{aub}(X) = \ max\{v_j(X) \mid j \ \geq \ minInd(X)\} \text{ and}$$
$$\overline{laub}(X) = au(X) + max\{v_j(X) \mid j > maxInd(X)\}$$

That is, $au \ll \overline{aub}_1 \ll auub(X)$, $au \ll \overline{aub} \ll aub$ and $au \ll \overline{laub} \ll lub$. Moreover, $aub \ll auub$ and $\overline{aub} \ll \overline{aub}_1$, i.e. $aub$ and $\overline{aub}$ are improved UBs of $auub$ and $\overline{aub}_1$, respectively.

For example, for $X = \{a, c\}$, $g(X) = \{T_1, T_3, T_4\}$, $maxInd(X)$= 3. Let there be a function $u_{max}(T_i) = max\{q_{ij}, j \in J\}$. Then, $u_{max}(T_1) = max\{5, 10, 1, 6, 3\}$= 10, $remu(X, T_1) = max\{6, 3\} = 6$ and $v_1(ac) = q_{11} + q_{31} + q_{41} = 5 + 5 + 10 = 20$. Then, we obtain $au(X) = (20 + 8)/2 = 14$, $auub(X) = \sum_{T_i \in g(X)} u_{max}(T_i) = 10 + 5 + 10 = 25$ and $\overline{aub}_1 = max\{20,\ 10,\ 8,\ 8,\ 9\} = \ 20$. Moreover, $lub(X) = 14 + (6 + 2 + 6) = 28$, $\overline{laub}(X) = 14 + max\{8, 9\} = 23$. Besides, since $minInd(ac)$= 1, $aub(X)$= $auub(X)$= 25, $\overline{aub}(X) = \overline{aub}_1(X) = 20$, it follows that $au(X) = 14 < \overline{aub}(X) = \overline{aub}_1(X) = 20 < \overline{laub}(X) = 23 < aub(X) = auub(X) = 25 < lub(X) = 28$, so the new $\overline{aub}$, $\overline{laub}$ and $\overline{aub}_1$ upper-bounds are strictly tighter than the previous *aub*, *lub* and *auub* upper-bounds, respectively.

For the itemset $\{ac\}$, the *aub* and $\overline{aub}$ upper-bounds are not less than *auub* and $\overline{aub}_1$; $\overline{laub}$ and *lub* are larger than $\overline{aub}_1$ and *auub*, respectively. However, for another itemset $Y = \{c, d\}$, $g(Y) = \{T_1, T_2, T_3\}$, we have $au(Y) = (5 + 14)/2 = 9.5$, $auub(Y) = 10 + 8 + 5 = 23$ and $\overline{aub}_1(Y) = max\{10, 18, 5, 14, 6\} = 18$, $lub(Y) = 9.5 + (3 + 3 + 0) = 15.5$, $\overline{laub}(Y) = 9.5 + 6 = 15.5$, and $aub(Y) = 6 + 6 + 2 = 14$, $\overline{aub}(Y) = max\{v_j(cd) | j \geq 3\} = max\{5, 14, 6\} = 14$. In this case, we have $au(Y) < aub(Y) = \overline{aub}(Y) < lub(Y) = \overline{laub}(Y) < \overline{aub}_1(Y) < auub(Y)$. In other words, $\overline{aub}$

and $aub$ are upper-bounds that are strictly tighter than $\overline{aub_1}$ and $auub$. Moreover, $\overline{laub}$ and $\overline{aub_1}$ as well as $lub$ and $auub$ are incomparable. The proposal of the new $\overline{aub}$, $\overline{laub}$, $\overline{aub_1}$ upper-bounds and an improved upper-bound of $\overline{aub}$, named and defined as $\overline{iaub}(X) = max\{v_j(X) \mid j \in X \ or \ j > maxInd(X)\}$, is one of the reasons for the excellent performance of $dHAUIM$.

An in-depth comparison of upper-bounds for the average utility measure and their pruning effects as well as the vertical utility-list structure using *diffset* technique are presented in more details in [78].

## 4.4 High Utility Itemset Mining with Negative Utilities

A limitation of traditional high utility itemset mining algorithms is that they assume that all utility values are positive. However, in real-life applications, databases often contain negative utility values. For example, consider a quantitative transaction database containing customer transactions at a retail store. In such database, it is common to find items having negative unit profits (negative external utilities). The reason is that selected items are often sold at a loss in retail stores to attract customers. It was shown that if negative unit profit values appears in a database, traditional high utility itemset mining algorithms can find an incomplete set of high utility itemsets [13]. The reason is that upper-bounds such as the TWU are no longer upper-bounds on the utility of items when negative utility values are considered. Thus, high-utility itemsets may be incorrectly pruned. To address this problem, algorithms have been proposed with novel upper-bounds. The first algorithm for *mining high utility itemsets with negative utility values* is HUINIV-Mine [13], which is a two-phase algorithm, extending Two-Phase [59]. Then, the FHN [48] algorithms was proposed. It is a one-phase utility-list based algorithm that extends the FHM algorithm, and was shown to be more than two orders of magnitude faster than HUINIV-Mine [13].

## 4.5 High Utility Itemset Mining with Discount Strategies

Another extension of high utility itemset mining that aims at being more realistic for analyzing customer transactions is *high utility itemset mining with discount strategies* [7]. This extension considers that items may be sold with three types of discount strategies: (1) an item can be sold with a discount set between 0% and 100%, (2) if a customer buys $n$ units of an item, he receives $m$ free units of this item, and (3) if a customer buys $n$ units of an item, he receives a $p$% discount on each additional unit purchased. An extended quantitative transaction database is considered where a *discount strategy table* let the user indicate which discount strategy is applied for each item, if any. Moreover, the *unit profit table* used in traditional high utility itemset mining is replaced by a table indicating the $cost$ and $pricetag$ of each item. This

allows calculating the utility of each item(set) by taking into account the discount strategies. A three-phase algorithm was first proposed to mine high utility itemsets while considering discount strategies [7]. Then, three faster algorithms named HUI-DTP, HUI-DMiner and HUI-DEMiner were proposed [51], which extend the Two-Phase [59], HUI-Miner [58] and FHM [31] algorithms, respectively.

### *4.6 Mining High Utility Itemset with a Maximum Length Constraint*

Another extension of high utility itemset mining is to discover high utility itemsets that do not contain more than a user-specified maximum number of items $maxLength$ [23]. The motivation for this extension is that traditional high utility itemset mining algorithms may find itemsets containing many items. But those itemsets are often rare, and thus may be less interesting than smaller itemsets for users. Thus, it is often desirable to set a constraint on the maximum number of items that high utility itemsets can contain. A naive approach to do this is to first discover all high utility itemsets using a standard high utility itemset mining algorithm, and then to apply the constraint as a post-processing step. Although this approach provides the correct result, it is inefficient, as it does not take advantage of the length constraint to reduce the search space. Hence, it is desirable to push the constraints as deep as possible in the mining process to improve the performance of the mining task. In frequent pattern mining, length constraints have been previously used such as the maximum length constraint [68]. The key idea of algorithms using a maximum length constraint is that since itemsets are generated by recursively appending items to itemsets, no item should be appended to an itemset containing the maximum number of items. Although this approach can prune the search space using length constraints, there is a need to find novel ways of reducing the search space using length constraints, to further improve the performance of algorithms. To address this issue, the FHM+ [23] algorithm was proposed by extending the FHM algorithm. It proposed a novel concept named *Length Upper-bound Reduction* (LUR), to reduce the upper-bounds on the utility of itemsets using length constraints, and thus further reduce the search space. It was shown that the proposed algorithm can be many times faster than the FHM algorithm and greatly reduce the number of patterns presented to the user.

### *4.7 Mining High Utility Itemsets that are Correlated*

Another limitation of traditional high utility itemset mining algorithms is that they often find itemsets that have a high profit but contain items that are weakly correlated. Those itemsets are misleading or useless for taking marketing decisions. For example, consider the transaction database of a retail store. Current algorithms may find that

buying a 50 inch plasma television and a pen is a high-utility itemset, because these two items have globally generated a high profit when sold together. But it would be a mistake to use this pattern to promote plasma television to people who buy pens because if one looks closely these two items are rarely sold together. The reason why this pattern may be a high utility itemset despite that there is a very low correlation between pens and plasma televisions, is that plasma televisions are very expensive, and thus almost any items combined with a plasma television may be a HUI. This limitation of traditional high utility itemset mining algorithms is important. In an experimental study, it will be shown that often less than 1% of the patterns found by traditional high utility itemset mining algorithms contains items that are strongly correlated [22].

To address this issue, various measures have been used to measure how correlated the items are in an itemset such as the *bond* [22], *all-confidence* [67] and *affinity* [4, 50] measures. The problem of mining correlated high utility itemsets with the bond measure is defined as follows.

**Definition 12 (bond measure).** Let there be an itemset $X$. The *disjunctive support* of the itemset $X$ in a database $D$ is denoted as $dissup(X)$ and defined as $|\{T_c \in D | X \cap T_c \neq \emptyset\}|$. The *bond* of itemset $X$ is defined as $bond(X) = sup(X)/dissup(X)$. The bond measure takes a value in the [0,1] interval and is monotonic [9].

**Definition 13 (Correlated high utility itemset mining with the bond measure).** For a quantitative transaction database and user-specified *minbond* and *minutil* thresholds, the problem of mining correlated high utility itemsets with the bond measure is to output each itemset $X$ such that $bond(X) \geq minbond$ and $u(X) \geq minutil$.

For instance, if $minutil = 25$ and $minbond = 0.6$ the set of correlated high utility itemsets is shown in Table 16. Consider the high utility itemset $\{a, b, c, d, e\}$. It is considered to not be correlated since its bond is 0.2.

Table 16: The correlated high utility itemsets for $minutil = 25$ and $minbond = 0.6$

| Itemset | Average utility | Bond |
|---|---|---|
| $\{a, c\}$ | 28 | 0.6 |
| $\{b, c\}$ | 28 | 0.6 |
| $\{c, e\}$ | 27 | 0.8 |
| $\{b, c, e\}$ | 37 | 0.6 |

To efficiently discover correlated high-utility itemsets with the bond measure, the FCHM$_{bond}$ algorithm [22] was proposed by extending the FHM algorithm. Experimental results have shown that FCHM can be much more efficient than the FHM algorithm by pruning huge amount of weakly correlated high utility itemset. An alternative measure of correlation named *all-confidence* [67] was integrated in the FCHM$_{allconfidence}$ algorithm [32]. The all-confidence is defined as follows.

**Definition 14 (All-confidence).** The all-confidence of an itemset $X$ is

$$all\text{-}confidence(X) = \frac{support(X)}{argmax\{support(Y)|\forall Y \subset X \wedge Y \neq \emptyset)\}}$$

The all-confidence of an itemset is a value in the [0, 1] interval, where a high value indicates that items are highly correlated. For example, the all-confidence of itemset $\{a, d\}$ is calculated as $all\text{-}confidence(\{a, d\}) = \frac{support(\{a,d\})}{argmax\{support(\{a\}),support(\{d\})\}} = \frac{2}{3} = 0.67$.

## *4.8 Periodic High Utility Itemset Mining*

An inherent limitation of traditional high utility itemset mining algorithms is that they are inappropriate to discover recurring customer purchase behavior, although such behavior is common in real-life situations [24]. For example, in a retail store, some customers may buy some set of products on approximately a daily or weekly basis. Detecting these purchase patterns is useful to better understand the behavior of customers and thus adapt marketing strategies, for example by offering specific promotions to cross-promote products such as reward or points to customers who are buying a set of products periodically. To address this limitation of previous work, the task of *periodic high-utility itemset mining* was proposed [24]. The goal is to efficiently discover all groups of items that are bought together periodically and generate a high profit, in a customer transaction database. The problem of periodic high utility itemset mining is defined as follows.

**Definition 15 (Periods of an itemset).** Let there be a database $D = \{T_1, T_2, ..., T_n\}$ containing $n$ transactions, and an itemset $X$. The set of transactions containing $X$ is denoted as $g(X) = \{T_{g_1}, T_{g_2}..., T_{g_k}\}$, where $1 \leq g_1 < g_2 < ... < g_k \leq n$. Two transactions $T_x \supset X$ and $T_y \supset X$ are said to be *consecutive with respect to X* if there does not exist a transaction $T_w \in g(X)$ such that $x < w < y$. The *period* of two consecutive transactions $T_x$ and $T_y$ in $g(X)$ is defined as $pe(T_x, T_y) = (y - x)$, that is the number of transactions between $T_x$ and $T_y$. The *periods of an itemset X* is a list of periods defined as $ps(X) = \{g_1 - g_0, g_2 - g_1, g_3 - g_2, ...g_k - g_{k-1}, g_{k+1} - g_k\}$, where $g_0$ and $g_k + 1$ are constants defined as $g_0 = 0$ and $g_k + 1 = n$. Thus, $ps(X) = \bigcup_{1 \leq z \leq k+1} (g_z - g_{z-1})$.

For example, consider the quantitative transaction database of Tables 17 and 18. Consider the itemset $\{a, c\}$, The list of transactions containing $\{a, c\}$ is $g(\{a, c\}) = \{T_1, T_3, T_5, T_6\}$. Thus, the periods of this itemset are $ps(\{a, c\}) = \{1, 2, 2, 1, 1\}$.

**Definition 16 (Maximum periodicity measure).** The maximum periodicity of an itemset $X$ is defined as $maxper(X) = max(ps(X))$ [77].

**Definition 17 (Minimum periodicity measure).** The minimum periodicity of an itemset $X$ is defined as $maxper(X) = max(ps(X))$ [24].

**Definition 18 (Average periodicity measure).** The average periodicity of an itemset $X$ is defined as $avgper(X) = \sum_{g \in ps(X)} / |ps(X)|$ [24].

For instance, the periods of itemsets $\{a, c\}$ and $\{e\}$ are respectively $ps(\{a, c\}) = \{1, 2, 2, 1, 1\}$ and $ps(\{e\}) = \{2, 1, 1, 2, 1, 0\}$. The average periodicities of these itemsets are respectively $avgper(\{a, c\}) = 1.4$ and $avgper(\{e\}) = 1.16$.

**Definition 19 (Periodic high utility itemset mining).** Let there be a quantitative transaction database and some user-specified thresholds $minutil$, $minAvg$, $maxAvg$, $minPer$ and $maxPer$, provided by the user. An itemset $X$ is a *periodic high-utility itemset* if and only if $minAvg \leq avgper(X) \leq maxAvg$, $minper(X) \geq minPer$, $maxper(X) \leq maxPer$, and $u(X) \geq minutil$. The goal of periodic high utility itemset mining is to discover all periodic high utility itemsets.

For example, consider the quantitative databases of Tables 17 and 18. If $minutil = 20$, $minPer = 1$, $maxPer = 3$, $minAvg = 1$, and $maxAvg = 2$, the complete set of periodic high utility itemsets is shown in Table 19.

Table 18: External utility values

| TID | Transaction |
|---|---|
| $T_1$ | $(a, 1), (c, 1),$ |
| $T_2$ | $(e, 1)$ |
| $T_3$ | $(a, 1), (b, 5), (c, 1), (d, 3), (e, 1)$ |
| $T_4$ | $(b, 4), (c, 3), (d, 3), (e, 1)$ |
| $T_5$ | $(a, 1), (c, 1), (d, 1)$ |
| $T_6$ | $(a, 2), (c, 6), (e, 2)$ |
| $T_7$ | $(b, 2), (c, 2), (e, 1)$ |

| Item | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| Unit profit | 5 | 2 | 1 | 2 | 3 |

Table 19: The set of periodic high utility itemsets

| Itemset | u(X) | $\|g(X)\|$ | minper(X) | maxper(X) | avgper(X) |
|---|---|---|---|---|---|
| $\{b\}$ | 22 | 3 | 1 | 3 | 1.75 |
| $\{b, e\}$ | 31 | 3 | 1 | 3 | 1.75 |
| $\{b, c, e\}$ | 37 | 3 | 1 | 3 | 1.75 |
| $\{b, c\}$ | 28 | 3 | 1 | 3 | 1.75 |
| $\{a\}$ | 25 | 4 | 1 | 2 | 1.4 |
| $\{a, c\}$ | 34 | 4 | 1 | 2 | 1.4 |
| $\{c, e\}$ | 27 | 4 | 1 | 3 | 1.4 |

To efficiently discover the periodic high utility itemsets, an algorithm named PHM [24] was proposed by extending the FHM algorithm. An experimental evaluation has show that the PHM algorithm is efficient, and can filter a huge number of non periodic patterns to reveal only the desired itemsets.

### *4.9 On-shelf High Utility Itemset Mining*

Another limitation of traditional high utility itemset mining algorithms for market basket analysis is that they consider that all items have the same shelf time (are on sale for the same time period). However, in real-life some items are only sold during some short time period (e.g. the summer). Algorithms ignoring the shelf time of items will have a bias toward items having more shelf time since they have more chance to generate a high profit [43, 44]. To address this limitation the problem of high utility itemset mining has been redefined as the problem *mining high on-shelf utility itemsets* [44]. High on-shelf utility itemset mining generalizes the problem of high utility itemset mining by considering the time time periods during which each item was on sale. Moreover, each transactions is associated to a time period. Formally, let $PE$ be a set of positive integers representing time periods. Each transaction $T_C \in D$ is associated to a time period $pt(T_c) \in PE$, representing the time period during which the transaction occurred.

For example, consider the transaction database shown in Tables 20 and 21. This database contains five transactions ($T_1, T_2 ... T_5$) and three time periods (1, 2, 3). Transaction $T_2$ occurred in time period 1, and contains items $a$, $c$, $e$ and $g$, which respectively appear in $T_2$ with an internal utility of 2, 6, 2 and 5. Table 21 indicates that the external utilities of these items are -5, 1, 3 and 1, respectively. The concept of time period is defined as follows.

Table 20: A Transaction Database with Time Period Information

| TID | Transactions | Period |
|---|---|---|
| $T_1$ | $(a, 1)(c, 1)(d, 1)$ | 1 |
| $T_2$ | $(a, 2)(c, 6)(e, 2)(g, 5)$ | 1 |
| $T_3$ | $(a, 1)(b, 2)(c, 1)(d, 6), (e, 1), (f, 5)$ | 2 |
| $T_4$ | $(b, 4)(c, 3)(d, 3)(e, 1)$ | 2 |
| $T_5$ | $(b, 2)(c, 2)(e, 1)(g, 2)$ | 3 |

Table 21: External Utility Values (Unit Profit)

| Item | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| Profit | -5 | 2 | 1 | 2 | 3 | 1 | 1 |

**Definition 20 (time period).** The *time periods (shelf time) of an itemset* $X \subseteq I$, is the set of time periods where $X$ was sold, defined as $pi(X) = \{pt(T_c) | T_c \in D \wedge X \subseteq T_c\}$.

**Definition 21 (Utility in a time period).** The *utility of an itemset* $X \subseteq I$ *in a time period* $h \in pi(X)$ is denoted as $u(X, h)$ and defined as $u(X, h) = \sum_{T_c \in D \wedge h = pt(T_c)} u(X, T_c)$. The *utility of an itemset* $X \subseteq I$ *in a database* $D$ is defined as $u(X) = \sum_{h \in pi(X)} u(X, h)$.

For instance, the utility of item $e$ in $T_3$ is $u(e, T_2) = 3 \times 2 = 6$. The utility of the itemset $\{c, e\}$ in $T_2$ is $u(\{c, e\}, T_2) = u(c, T_2) + u(e, T_2) = 1 \times 6 + 3 \times 2 = 12$. The time periods of itemset $\{c, e\}$ are $pi(\{c, e\}) = \{1, 2, 3\}$. The utility of $\{c, e\}$ in periods 1, 2 and 3 are respectively $u(\{c, e\}, 1) = 12$, $u(\{c, e\}, 2) = 4$ and $u(\{c, e\}, 3) = 11$. The utility of $\{c, e\}$ in the database is $u(\{c, e\}) = 12 + 4 + 11 = 27$.

**Definition 22 (Relative utility measure).** The *transaction utility* (TU) of a transaction $T_c$ is the sum of the utility of the items from $T_c$ in $T_c$. i.e. $TU(T_c) = \sum_{i \in T_c} u(i, T_c)$. Given an itemset $X$, the *total utility of the time periods of* $X$ is defined as $to(X) = \sum_{h \in pi(X) \wedge T_c \in D \wedge h = pt(T_c)} TU(T_c)$. The *relative utility of an itemset* $X \subseteq I$ *in a database* $D$ is defined as $ru(X) = u(X)/to(X)$, if $to(X) \neq 0$, and is defined as 0 otherwise. The relative utility of an itemset $X$ represents how large the profit/loss generated by $X$ is compared to the total profit/loss generated during the time periods where $X$ was sold. The relative utility measure is useful for retailers as it is an indicator of the relative selling performance (profit/loss) of an itemset during time periods where it was on the shelves. It can thus be used to compare the selling performance of various itemsets in terms of their relative contribution to the overall profit of a retail store, to determine which itemsets are the most profitable.

For example, the transaction utility of transactions $T_1, T_2, ... T_5$ are $TU(T_1) = -2$, $TU(T_2) = 7$, $TU(T_3) = 20$, $TU(T_4) = 20$ and $TU(T_5) = 11$. The total utility of the time periods of $\{c, e\}$ is $to(\{c, e\}) = 58$. The relative utility of $\{c, e\}$ is $ru(\{c, e\} = u(\{c, e\})/to(\{c, e\}) = 27/58 = 0.46$.

**Definition 23 (Problem of high on-shelf utility itemset mining).** An itemset $X$ is a *high on-shelf utility itemset* if its relative utility $ru(X)$ is no less than a user-specified minimum utility threshold $minutil$ given by the user ($0 \geq minutil \geq 1$). Otherwise, $X$ is a *low on-shelf utility itemset*. The *problem of high on-shelf utility itemset mining* is to discover all high on-shelf high utility itemsets in a database [43, 44].

For example, consider the databases of Tables 20 and 21. If $minutil = 0.43$, 21 high on-shelf utility itemsets are found. They are $\{a, b, c, d, e, f\}$:0.44, $\{b, d, f\}$:0.47, $\{b, d, e, f\}$:0.53, $\{b, c, d, e, f\}$:0.55, $\{b, c, d, e\}$:0.49, $\{d, e, f\}$:0.44, $\{c, d, e, f\}$:0.47, $\{b, g\}$:0.54, $\{b, e, f\}$:0.81, $\{b,c, e, g\}$:1.0, $\{b, c, g\}$:0.72, $\{e, g\}$:0.51, $\{c, e, g\}$ :0.77, $\{c, g\}$:0.48, $\{b, d\}$:0.67, $\{b, d, e\}$:0.8, $\{b, c, d, e\}$:0.89, $\{b, c, d\}$:0.75, $\{c, d, e\}$:0.43, $\{b, e\}$:0.45 and $\{b, c, e\}$:0.55, where the relative utility of each itemset is indicated after the colon.

The three-phase TS-HOUN algorithm [43] was proposed to mine high utility itemsets while considering negative and positive utility values. Then, a faster one-phase algorithm named FOSHU [34] was proposed by extending FHM. Thereafter, a top-k on-shelf high utility itemset mining algorithm named KOSHU [15] was proposed by extending FOSHU.

Recently, other works have also considered the time dimension in high utility itemset mining but without using the concept of time periods. Lin et al. proposed an algorithm that find all itemsets that have a high utility in recent times, called *recent high-utility itemsets*, by considering a decay function, which gives more weight to

recent transactions [49]. In another work, the concept of *local high utility itemsets* was proposed to find itemsets that have a high utility in non predefined time periods such as a high sale of notebooks and pen in the first week of the back-to-school season [33].

### *4.10 High Utility Itemset Mining in Dynamic Databases*

Another limitations of traditional high utility itemset mining algorithms is to assume that databases are static. Thus, traditional algorithms cannot update their results when a database is updated. Hence, to obtain updated results, one needs to apply these algorithms again from scratch, which is inefficient. To address this limitation, algorithms have been designed for discovering high-utility itemsets in databases that are incrementally updated when new transactions are inserted [5, 26, 73, 86]. These algorithms reuse results from their previous executions to speed up the discovery of high utility itemsets when a database is updated. Algorithms have also been designed for mining high utility itemsets in an infinite streams of transactions [19, 75].

### *4.11 Other Extensions*

Several other extensions of the problem of high utility itemset mining have been studied. For example, the concept of privacy preservation has been studied in the context of high utility itemset mining to prevent the disclosure of sensitive high utility itemsets [54]. Such algorithms modify a database to ensure that sensitive itemsets cannot be found for a given minimum utility threshold. A challenge it to make as few modifications to the original database as possible to hide itemsets. It can thus be viewed as an optimization problem that can be solved using evoluationary or swarm intelligence algorithms [54]

Evoluationary and swarm intelligence algorithms have also been used to find approximate solutions to the problem of high utility itemset mining and variations [57, 76, 92]. An advantage of such algorithms is to quickly find a solution. However, a drawback is that the algorithms cannot guarantee finding all desired patterns.

Another popular variations of high utility itemset mining is to *mine high utility itemsets using multiple minimum support* [74] *or utility thresholds* [36, 42]. The motivation is that in traditional high utility itemset mining a single threshold is used to evaluate all items. But in real-life not all items are equally popular or can generate as much profit. Thus, the *rare item problem* occurs, which is that few patterns are found containing less frequent or profitable items with other more frequent or profitable items. By letting the user set a different threshold for each item, this problem can be alleviated. To avoid setting all thresholds by hands, some studies utilize a function to automatically set the threshold of each item [42].

The problem of high utility itemset mining has also been generalized to mine high utility patterns in sequences of quantitative transactions. A *quantitative sequence databases* is a database where each entry is a sequence of quantitative transactions. For example, a quantitative sequence database is depicted in Table 22 with a corrresponding external utility table in Table 23. In this example, each sequence represents the transactions made by a customer. For instance, sequence $s1$ means that a customer bought 1 unit of item $a$ and 2 units of items $b$, followed by buying 2 units of item $c$, followed by buying 3 units of item $f$. Two main problems are defined for such database which are to discover *high utility sequential patterns* [3, 6, 12, 84] and *high utility sequential rules* [93]. A high utility sequential pattern is a subsequence of itemsets that appear in quantitative sequence of transactions and has a high utility. A high utility sequential rule is a rule of the form $X \rightarrow Y$ where $X$ and $Y$ are disjoint itemsets, and meaning that if $X$ appears, it is often followed by $Y$ and has a high utility.

Table 23: External Utility Values

| SID | Sequences |
|---|---|
| $s_1$ | $\langle\{(a,1)(b,2)\}(c,2)(f,3)(g,2)(e,1)\rangle$ |
| $s_2$ | $\langle\{(a,1)(d,3)\}(c,4),(b,2),\{(e,1)(g,2)\}\rangle$ |
| $s_3$ | $\langle(a,1)(b,2)(f,3)(e,1)\rangle$ |
| $s_4$ | $\langle\{(a,3)(b,2)(c,1)\}\{(f,1)(g,1)\}\rangle$ |

| Item | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| Profit | 1 | 2 | 5 | 4 | 1 | 3 | 1 |

## 5 Research Opportunities

Even though the problem of high utility itemset mining has been studied for more than a decade, and numerous papers have been published on this topic, there are numerous research opportunities. We have identified four types of opportunities:

- *Novel applications*. The first research opportunities are to apply existing pattern mining algorithms in new ways in terms of application domains. Since pattern mining algorithms are quite general, they can be applied in a multitude of domains. In particular, the use of pattern mining methods in emerging research areas such as social network analysis, the Internet of Things, sensor networks provides several novel possibilities in terms of applications.
- *Enhancing the performance of pattern mining algorithms*. Since pattern mining can be quite time-consuming, especially on dense databases, large databases, or databases containing many long transactions, much research is carried on developing more efficient algorithms. This is an important problem especially for new extensions of the high utility itemset mining problem such as on-shelf high utility itemset mining or periodic high-utility itemset mining, which have been less

explored. Many opportunities also lies in distributed, GPU, multi-core or parallel algorithm development to increase speed and scalability of the algorithms.
- *Extending pattern mining to consider more complex data*. Another research opportunity is to develop high utility pattern mining algorithms that can be applied on complex types of data. As mentioned in this paper, various extensions have been proposed. But it still remains a problem to handle more complex types of data such as spatial data.
- *Extending pattern mining to discover more complex and meaningful types of patterns*. Related to the above opportunity, another important issue to discover more complex types of patterns. Also, another research opportunity is to work on the evaluation of patterns using for example novel measures, because it is also key to ensure that the most interesting or useful patterns are found.

## 6 Open-Source Implementations

Implementations of high utility pattern mining algorithms are offered in the *SPMF data mining library* ( http://www.philippe-fournier-viger.com/spmf/) [21, 25]. It offers more than 130 algorithms for mining patterns such as high utility patterns, itemsets, sequential patterns, sequential rules, periodic patterns, and association rules. It is a multi-platform library developed in Java and released under the GPL3 license. It is designed to be easily integrated in other Java software programs, and can be run as a standalone software using its command-line or graphical user interface. Standard datasets for benchmarking high utility itemset and pattern mining algorithms can be found on the SPMF website at http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

## 7 Conclusion

High-utility itemset mining is an active field of research having numerous applications. This chapter has presented the problem of high-utility itemset mining, discussed the main techniques for exploring the search space of itemsets, employed by high-utility itemset mining algorithms. Then, the paper has discussed extensions of the basic high-utility itemset mining algorithm to overcome some of its limitations, for example, to handle dynamic databases, and the use of various constraints. Lastly, the paper has discussed research opportunities and open-source software.

**Acknowledgements** This work is supported by the Youth 1000 Talent funding from the National Science Fundation of China and Harbin Institute of Technology.

## References


1. Aggarwal, C.C.: Data mining: the textbook. Springer, Heidelberg (2015)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, pp. 487–499. Morgan Kaufmann (1994)
3. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S.: A novel approach for mining high-utility sequential patterns in sequence databases. ETRI journal **32**(5), 676–686 (2010)
4. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Choi, H.J.: A framework for mining interesting high utility patterns with a strong frequency affinity. Information Sciences. **181**(21), 4878–4894 (2011)
5. Ahmed, C.F., Tanbeer, S.K., Jeong, B.-S., Lee, Y.-K.: Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases. IEEE Trans. Knowl. Data Eng. **21**(12), 1708–1721 (2009)
6. Alkan, O.K., Karagoz, P.: Crom and huspext: Improving efficiency of high utility sequential pattern extraction. IEEE Trans. Knowl. Data Eng. **27**(10), 2645–2657 (2015)
7. Bansal, R., Dawar, S. and Goyal, V.: An efficient algorithm for mining high-utility itemsets with discount notion. In: Proc. Intern. Conf. on Big Data Analytics, 84–98. Springer (2015)
8. Barron, A., Rissanen, J., Yu, B.: The minimum description length principle in coding and modeling. IEEE Transactions on Information Theory. **44**(6), 2743–2760 (1998)
9. Bouasker, S., Ben Yahia, S.: Key correlation mining by simultaneous monotone and anti-monotone constraints checking. In: Proc. 30th Symp. on Applied Computing, pp. 851-856. ACM (2015)
10. Brauckhoff, D., Dimitropoulos, X., Wagner, A., Salamatian, K.: Anomaly extraction in backbone networks using association rules. IEEE/ACM Transactions on Networking, **20**(6), 1788–1799 (2012)
11. Chan, R., Yang, Q., Shen, Y.: Mining High Utility Itemsets. In: Proc. of 3rd IEEE Int'l Conf. on Data Mining, pp. 19–26. IEEE (2003)
12. Chi, T.T., Fournier-Viger, P.: A Survey of High Utility Sequential Patten Mining. In: Fournier-Viger et al. (eds). High-Utility Pattern Mining: Theory, Algorithms and Applications, to appear. Springer (2018)
13. Chu, C., Tseng, V.S. Liang, T.: An Efficient Algorithm for Mining High Utility Itemsets with Negative Item Values in large databases. Applied Mathematics and Computation **215**(2), 767–778 (2009)
14. Dam, T.-L., Li, K., Fournier-Viger, P., Duong, H.: CLS-Miner: Efficient and Effective Closed High utility Itemset Mining. Frontiers of Computer Science, doi: https://doi.org/10.1007/s11704-016-6245-4. Springer (2018)
15. Dam, T.-L., Li, K., Fournier-Viger, P., Duong, H.: An efficient algorithm for mining top-k on-shelf high utility itemsets. Knowledge and Information Systems **52**(2), 621–655 (2017)
16. Duan, Y., Fu, X., Luo, B., Wang, Z., Shi, J., Du, X.: Detective: Automatically identify and analyze malware processes in forensic scenarios via DLLs. In: Proc. 2015 IEEE International Conf. on Communications, pp. 5691–5696. IEEE (2015)
17. Duong, Q.H., Fournier-Viger, P., Ramampiaro, H., Norvag, K. Dam, T.-L.:Efficient High Utility Itemset Mining using Buffered Utility-Lists. Applied Intelligence **48**(7), 1859–1877 (2017)
18. Duong, Q.-H., Liao, B., Fournier-Viger, P., Dam, T.-L.: An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. Knowledge-Based Systems **104**, 106–122 (2016)
19. Duong, H., Ramampiaro, H., Norvag, K., Fournier-Viger, P., Dam, T.-L.: High Utility Drift Detection in Quantitative Data Streams. Knowledge-Based Systems **157**(1), 34–51 (2018)
20. Fernando, B., Elisa F., Tinne T.: Effective use of frequent itemset mining for image classification. In: Proc. 12th European Conf. on Computer Vision, pp. 214–227. Springer (2012)
21. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: a Java Open-Source Pattern Mining Library, Journal of Machine Learning Research, **15**,3389–3393 (2014)

22. Fournier-Viger, P., Lin, J.C.-W., Dinh, T., Le, H.B.: Mining Correlated High-Utility Itemsets using the Bond Measure. In: Proc. Intern. Conf. Hybrid Artificial Intelligence Systems. pp. 53–65. Springer (2016)
23. Fournier-Viger, P., Lin, J.C.-W., Duong, Q.-H., Dam, T.-L.: FHM+: Faster High-Utility Itemset Mining using Length Upper-Bound Reduction. In: Proc. 29th Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 115–127. Springer (2016)
24. Fournier-Viger, P., Lin, J.C.-W., Duong, Q.H., Dam, T.L.: PHM: Mining Periodic High-Utility Itemsets. In: Proc. 16th Industrial Conf. on Data Mining, pp. 64–79. Springer (2016)
25. Fournier-Viger, P., Lin, J.C.-W., Gomariz, A., Soltani, A., Deng, Z., Lam, H.T.: The SPMF Open-Source Data Mining Library Version 2. In: Proc. 19th European Conf. on Principles of Data Mining and Knowledge Discovery, pp. 36–40. Springer (2016)
26. Fournier-Viger, P., Lin, J.C.-W., Gueniche, T., Barhate, P.: Efficient Incremental High Utility Itemset Mining. In: Proc. 5th ASE International Conf. on Big Data. ASE (2015)
27. Fournier-Viger, P., Lin, J.C.-W., Kiran, R. U., Koh, Y. S., Thomas, R.: A Survey of Sequential Pattern Mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
28. Fournier-Viger, P., Lin, J.C.-W., Vo, B, Chi, T.T., Zhang, J., Le, H. B.: A Survey of Itemset Mining. WIREs Data Mining and Knowledge Discovery, e1207 doi: 10.1002/widm.1207 (2017)
29. Fournier-Viger, P., Lin, C.W., Wu, C.-W., Tseng, V.S., Faghihi, U.: Mining Minimal High-Utility Itemsets. Proc. 27th International Conf. on Database and Expert Systems Applications, pp. 88-101. Springer (2016)
30. Fournier-Viger, P., Wu, C.W., Tseng, V.S.: Novel Concise Representations of High Utility Itemsets using Generator Patterns. In: Proc. 10th Intern. Conf. on Advanced Data Mining and Applications, pp. 30–43. Springer (2014)
31. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proc. 21st Inter. Symp. Methodologies for Intelligent Systems, pp. 83–92. Springer (2014)
32. Fournier-Viger, P., Zhang, Y., Lin, J. C.-W., Dinh, T., Le, B.: Mining Correlated High-Utility Itemsets Using Various Correlation Measures. Logic Journal of the IGPL, Oxford Academic, *to appear* (2018)
33. Fournier-Viger, P., Zhang, Y., Lin, J.C.-W., Fujita, H., Koh, Y.-S.: Mining Local High Utility Itemsets . In: Proc. 29th International Conf. on Database and Expert Systems Applications (DEXA 2018), to appear. Springer (2018)
34. Fournier-Viger, P., Zida, S.: FOSHU: Faster On-Shelf High Utility Itemset Mining– with or without negative unit profit. In: Proc. 30th Symposium on Applied Computing, pp. 857–864. ACM (2015)
35. Fournier-Viger, P., Zida, S. Lin, C.W., Wu, C.-W., Tseng, V. S.: EFIM-Closed: Fast and Memory Efficient Discovery of Closed High-Utility Itemsets. In: Proc. 12th Intern. Conf. on Machine Learning and Data Mining, pp. 199–213. Springer (2016)
36. Gan, W., Lin, J.C.-W., Fournier-Viger, P., Chao, H.C.: More efficient algorithms for mining high-utility itemsets with multiple minimum utility thresholds. In: Proc. 26th International Conf. on Database and Expert Systems Applications, pp. 71–87. Springer (2016)
37. Glatz, E., Mavromatidis, S., Ager, B., Dimitropoulos, X.: Visualizing big network traffic data using frequent pattern mining and hypergraphs. Computing **96**(1),27–38 (2014)
38. Han, J., Pei, J., Kamber, M.: Data mining: concepts and techniques. Elsevier, Amsterdam (2011)
39. Han, J., Pei, J., Ying, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Discov. **8**(1),53–87 (2004)
40. Hegland, M.: The apriori algorithm–a tutorial. Mathematics and computation in imaging science and information processing. **11**, 209–62 (2005)
41. Hong, T.P., Lee, C.H., Wang, S.L.: Mining High Average-Utility Itemsets. In: Proc. of IEEE Int'l Conf. on Systems, Man, and Cybernetics, pp. 2526-2530. IEEE (2009)
42. Krishnamoorthy, S.: Efficient mining of high utility itemsets with multiple minimum utility thresholds. Engineering Applications of Artificial Intelligence **69**, 112–126 (2018)

43. Lan, G.-C., Hong, T.-P., Huang, J.-P., Tseng, V.S.: On-shelf utility mining with negative item values. Expert Systems with Applications **41**, 3450–3459 (2014)
44. Lan, G.-C., Hong, T.-P. Tseng, V.S.: Discovery of high utility itemsets from on-shelf time periods of products. Expert Systems with Applications **38**, 5851–5857 (2011)
45. Lan, G.-C., Hong, T.P., Tseng, V.S.: A Projection-based Approach for Discovering High Average-utility Itemsets. Journal of Information Science and Engineering textbf28(1), 193–209 (2012)
46. Lan, G.-C., Hong, T.-P., Tseng, V.S.: Efficiently mining high average-utility itemsets with an improved upper-bound strategy. International Journal of Information Technology and Decision Making **11**(5), 1009–1030 (2012)
47. Lan, G.-C., Hong, T.P., Tseng, V.S.: An efficient projection-based indexing approach for mining high utility itemsets. Knowl. and Inform. Syst. **38**(1), 85–107 (2014)
48. Lin, J.C.-W., Fournier-Viger, P., Gan, W.: FHN: An Efficient Algorithm for Mining High-Utility Itemsets with Negative Unit Profits. Knowledge-Based Systems **111**(1), 283–298 (2016)
49. Lin, J.C.-W., Gan, W., Fournier-Viger, P., Chen, H.-C.: Mining Recent High-Utility Patterns from Temporal Databases with Time-Sensitive Constraint. mÈ Proc. 18th Intern. Conf. on Data Warehousing and Knowledge Discovery, pp. 3-16. Springer (2016)
50. Lin, J.C.-W., Gan, W., Fournier-Viger, P., Hong, T.-P. Chao, H.-C.: FDHUP: Fast Algorithm for Mining Discriminative High Utility Patterns. Knowledge and Information Systems **51**(3), 873–909 (2016)
51. Lin, J.C.-W., Gan, W., Fournier-Viger, P., Hong, T.P., Tseng, V.S.: Fast Algorithms for Mining High-Utility Itemsets with Various Discount Strategies. Advanced Engineering Informatics **30**(2), 109–126 (2016)
52. Lin, J.C.-W., Hong, T.P., Lu, W.H.: An effective tree structure for mining high utility itemsets. Expert Systems with Applications **38**(6), 7419–24 (2011)
53. Lin, J.C.-W., Li, T., Fournier-Viger, P., Hong, T.-P., Voznak, M., Zhan, J.: An Efficient Algorithm to Mine High Average-Utility Itemsets. Advanced Engineering Informatics, **30**(2), 233–243 (2016)
54. Lin, J.C.-W., Liu, Q., Fournier-Viger, P., Hong, T.-P., Voznak, M., Zhan, J.: A Sanitization Approach For Hiding Sensitive Itemsets Based On Particle Swarm Optimization. Engineering Applications of Artificial Intelligence **53**:1–18 (2016)
55. Lin, J.C.-W., Ren, S., Fournier-Viger, P., Hong, T.-P.: EHAUPM: Efficient High Average-Utility Pattern Mining with Tighter Upper-Bounds. IEEE Access **5**, 12927–12940. IEEE (2017)
56. Lin, Y.C., Wu, C.W., Tseng, V.S.: Mining high utility itemsets in big data. In: Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining, pp. 649–661. Springer (2015)
57. Lin., J.C.-W., Yang, L., Fournier-Viger, Frnda, J., Sevcik, L., Voznak, M.: An Evolutionary Algorithm to Mine High-Utility Itemsets. Advances in Electrical and Electronic Engineering **13**(5), 392-398 (2015)
58. Liu, M., Qu,. J.: Mining high utility itemsets without candidate generation. In: Proc. 21st ACM Intern. Conf. Information and knowledge management, pp. 55–64. ACM (2012)
59. Liu, Y., Liao, W.K. and Choudhary, A.N.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, pp. 689–695. Springer (2005)
60. Liu, J., Wang, K., Fung, B.: Direct discovery of high utility itemsets without candidate generation, In: Proc. 12th IEEE Intern. Conf. Data Mining, pp. 984–989. IEEE (2012)
61. Liu, Y., Zhao, Y., Chen, L., Pei, J., Han, J.: Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. IEEE Transactions on Parallel and Distributed Systems **23**(11),2138–2149 (2012)
62. Lu, T., Vo, B., Nguyen, H.T., Hong, T.P.: A New Method for Mining High Average Utility Itemsets, In: Proc. 13th Intern. Conf. on Computer Information Systems and Industrial Management Applications, pp. 33–42. Springer (2014)
63. Lucchese, C., Orlando, S., Perego, R.: Fast and Memory Efficient Mining of Frequent Closed Itemsets. IEEE Trans. Knowl. Data Eng. **18**(1),21–36 (2006)
64. Mukherjee, A., Liu, B., Glance, N.: Spotting fake reviewer groups in consumer reviews. In: Proc. 21st international conference on World Wide Web, pp. 191–200. ACM (2012)

65. Mwamikazi, E., Fournier-Viger, P., Moghrabi, C., Baudouin, R.: A Dynamic Questionnaire to Further Reduce Questions in Learning Style Assessment. In: Proc. 10th Int. Conf. Artificial Intelligence Applications and Innovations, pp. 224–235. Springer (2014)
66. Naulaerts, S., Meysman, P., Bittremieux, W., Vu, T.N., Berghe, W.V., Goethals, B, Laukens, K.: A primer to frequent itemset mining for bioinformatics. Briefings in bioinformatics **16**(2), 216–231 (2015)
67. Omiecinski, E.R.: Alternative interest measures for mining associations in databases. IEEE Trans. Knowl. Data Eng. **15**(1), 57–69 (2003)
68. Pei, J., Han, J.: Constrained frequent pattern mining: a pattern-growth view. ACM SIGKDD Explorations Newsletter **4**(1), 31–39 (2012)
69. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D.: H-mine: Hyper-structure mining of frequent patterns in large databases. In: Proc. 2001 IEEE Intern. Conf. Data Mining, pp. 441–448. IEEE (2001)
70. Peng, A.X., Koh, Y.S., Riddle, P.: mHUIMiner: A Fast High Utility Itemset Mining Algorithm for Sparse Datasets. In: Pacific-Asia Conf. on Knowledge Discovery and Data Mining, pp. 196–207 (2017)
71. Qu, J.-F., Liu, M., Fournier-Viger, P.: Efficient algorithms for high utility itemset mining without candidate generation . In: Fournier-Viger et al. (eds). High-Utility Pattern Mining: Theory, Algorithms and Applications, to appear. Springer (2018)
72. Ryang, H., Yun, U.: Top-k high utility pattern mining with effective threshold raising Strategies. Knowl.-Based Syst. **76**, 109–126 (2015)
73. Ryang, H., Yun, U.: High utility pattern mining over data streams with sliding window technique. Expert Systems with Applications **57**, 214–231 (2016)
74. Ryang, H., Yun, U., Ryu, K.: Discovering high utility itemsets with multiple minimum supports. Intell. Data Anal., **18**(6), 1027–1047 (2014)
75. Shie, B.-E., Yu, P.S., Tseng, V.S.: Efficient algorithms for mining maximal high utility itemsets from data streams with different models. Expert Syst. Appl. **39**(17), 12947–12960 (2012)
76. Song, W., Huang, C.: Discovering High Utility Itemsets Based on the Artificial Bee Colony Algorithm. In: Proc. the 22nd Pacific-Asia Conf. Knowledge Discovery and Data Mining, pp. 3–14. Springer (2018)
77. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., Lee, Y. K.: Discovering periodic-frequent patterns in transactional databases. In: Proc. 13th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, pp. 242–253. Springer (2009)
78. Truong, T., Duong, H., Le, B., Fournier-Viger, P.: Efficient Vertical Mining of High Average-Utility Itemsets based on Novel Upper-Bounds . IEEE Trans. Knowl. Data Eng. DOI:10.1109/TKDE.2018.2833478 (2018)
79. Tseng, V.S., Shie, B.-E., Wu, C.-W., Yu., P. S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. **25**(8), 1772–1786 (2013)
80. Tseng, V., Wu, C., Fournier-Viger, P., Yu, P.S.: Efficient Algorithms for Mining Top-K High Utility Itemsets. IEEE Trans. Knowl. Data Eng. **28**(1), 54–67 (2016)
81. Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In: Proc. ICDM'04 Workshop on Frequent Itemset Mining Implementations. CEUR (2004)
82. Yao, H., Hamilton, H. J.: Mining itemset utilities from transaction databases. Data and Knowledge Engineering **59**(3), 603–626 (2006)
83. Yao, H., Hamilton, H.J., Geng, L.: A Unified Framework for Utility-based Measures for Mining Itemsets. In: Proc. of ACM SIGKDD Workshop on Utility-Based Data Mining, pp. 28-37. ACM (2006)
84. Yin, J., Zheng, Z. and Cao, L.: USpan: an efficient algorithm for mining high utility sequential patterns. Proc. of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 660–668. ACM (2012)
85. Yun, U., D. Kim.: Mining of high average-utility itemsets using novel list structure and pruning strategy. Future Generation Computer Systems **68**, 346-360 (2016)
86. Yun, U., Ryang, H.:Incremental high utility pattern mining with static and dynamic databases. Applied Intelligence **42**(2), 323–352 (2015)

87. Yun, U., Ryang, H., Ryu, K.H.: High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Syst. Appl. **41**(8),3861–3878 (2014)
88. Wu, C.W., Fournier-Viger, P., Gu, J.-Y., Tseng, V.S.: Mining Closed+ High Utility Itemsets without Candidate Generation. in: Proc. 2015 Conf. on Technologies and Applications of Artificial Intelligence, pp. 187–194. IEEE (2015)
89. Wu, C.-W., Fournier-Viger, P., Gu, J.-Y., Tseng, V.-S.: Efficient algorithms for high utility itemset mining without candidate generation . In: Fournier-Viger et al. (eds). High-Utility Pattern Mining: Theory, Algorithms and Applications, to appear. Springer (2018)
90. Wu, C.-W., Fournier-Viger, P., Yu., P.S., Tseng, V.S.: Efficient Mining of a Concise and Lossless Representation of High Utility Itemsets. Proc. 11th IEEE Intern. Conf. on Data Mining, pp. 824–833. IEEE (2011)
91. Zaki, M.J.: Scalable Algorithms for Association Mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
92. Zhang, L., Fu, G., Cheng, F., Qiu, J., Su, Y.: A multi-objective evolutionary approach for mining frequent and high utility itemsets. Applied Soft Computing **62**, 974–986 (2018)
93. Zida, S., Fournier-Viger, P., Wu, C.-W., Lin, J.C.-W., Tseng, V.S.: Efficient Mining of High Utility Sequential Rules. In: Proc. 11th Intern. Conf. on Machine Learning and Data Mining, pp. 157–171. Springer (2015)
94. Zida, S., Fournier-Viger, P., Lin, J.C.-W., Wu, C.W., Tseng, V.S.: EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining. In: Proc. 14th Mexican Intern. Conf. Artificial Intelligence, pp. 530–546. Springer (2015)