# Discovering Frequent Patterns in Very Large Transactional Databases


**Jose M. Luna**

**Abstract** Finding frequent patterns in very large transactional databases is a challenging problem of great concern in many real-world applications. In this chapter, we first introduce the model of frequent patterns. Second, we describe the search space for finding the desired patterns. Third, we present four popular algorithms to find the patterns. Finally, we present the extensions of frequent patterns.

## 1 Introduction

Technological advances in the field of Information and Communication Technologies have enabled organizations to collect and store big data effectively. Useful knowledge that can empower users with the competitive information to achieve socioeconomic development lies in this data. The field of data analytics (or data mining) has emerged to discover the hidden knowledge in big data.

Frequent pattern mining (FPM) is an important knowledge discovery technique in data mining. It involves finding all frequently occurring patterns in a transactional database. A classic application is market basket analysis. It involves finding the itemsets that were frequently purchased by the customers in the data. An example of a frequent pattern is $\{Cheese, Beer\}$ $[support = 10\%]$, which provides the information that 10% of the customers have purchased the items 'Cheese' and 'Beer' together. Such an information may be found to be extremely useful to the users for various purposes, such as product recommendation and inventory management. Other applications of FPM may be found at [2].

The basic model of frequent pattern is as follows: Let $I = \{i_1, ..., i_n\}$ be the set of items. Let $X \subseteq I$ be a pattern (or an itemset). A pattern containing $k$ number of items is called as $k$-pattern. A transaction $t_{tid} = (tid, Y)$, where $tid \in \mathbb{R}^+$ represents the transaction identifier and $Y \subseteq I$ represents a pattern. A transactional database, $DB = \{t_1.t_2, \cdots, t_m\}$, $m \geq 1$, is a set of transactions. The $support$ of pattern $X$ in

J. M. Luna (✉)
Department of Computer Science and Numerical Analysis, Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Cordoba, 14071 Córdoba, Spain
e-mail: jmluna@uco.es



R. Uday Kiran et al. (eds.), *Periodic Pattern Mining*,
https://doi.org/10.1007/978-981-16-3964-7_2

$DB$, denoted as $sup(X)$, represents the number of transactions containing $X$ in $DB$. The pattern $X$ is said to be a frequent pattern if $sup(X) \geq minSup$, where $minSup$ represents the user-specified *minimum support* threshold value. The **problem definition** of frequent pattern mining is to find all patterns that have $support$ no less than $minSup$ in $DB$. (Please note that the $support$ of a pattern can also expressed in percentage of database size. However, we employ the former definition of $support$ throughout this book for ease of explanation.)

*Example 1* Let $I = \{a, b, c, d, e\}$ be a set of items. A hypothetical transactional database generated by the items in $I$ is shown in Table 1. The set of items $a$ and $b$, i.e., $\{a, b\}$ (or $ab$ in short) is a pattern. This pattern contains 2 items. Therefore, it is a 2-pattern. In Table 1, the pattern $ab$ appears in the transactions whose $tids$ are 2, 4, and 5. Thus, the $support$ of $ab$ in Table 1 is 3 or 60% $(= (3/5) \times 100)$. If the user-specified $minSup = 2$, then $ab$ is said to be a frequent pattern because $sup(ab) \geq minSup$. The complete set of frequent patterns generated from Table 1 is shown in Table 2.

It is important to remark that setting a minimum support value is a non-trivial task and generally requires a profound background in the application field. Inexpert and many expert users need to try different thresholds by guessing and re-executing the algorithms once and again until the results are good for them. As it has been demonstrated, a small change in the threshold value may lead to very few or an extremely large set of solutions.

**Table 1** Transactional database

| tid | Items |
|---|---|
| 1 | *ac* |
| 2 | *abc* |
| 3 | *bde* |
| 4 | *abe* |
| 5 | *abcd* |

**Table 2** Frequent patterns found in Table 1

| Pattern | Support | Pattern | Support |
|---|---|---|---|
| *a* | 4 | *ac* | 3 |
| *b* | 4 | *ae* | 2 |
| *c* | 3 | *bc* | 2 |
| *d* | 2 | *bd* | 2 |
| *e* | 2 | *abc* | 2 |
| *ab* | 3 | | |

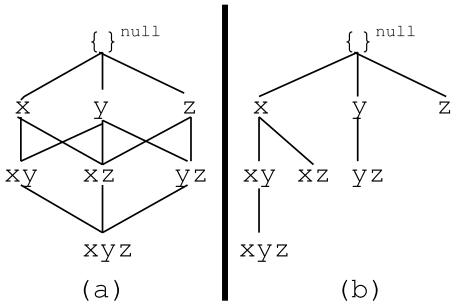


**Fig. 1** Search space representation methods. **a** Itemset lattice. **b** Set enumeration tree

## 2 Search Space

The space of items in a database gives rise to an itemset lattice (or a set enumeration tree[1]). This itemset lattice represents the search space for finding frequent patterns in a database. The size of this search space is $2^n - 1$, where $n$ represents the total number of distinct items in the database. Reducing this huge search space is a non-trivial and challenging task in frequent pattern mining. When confronted with this problem in the real-world applications, researchers employed *anti-monotonic property*[2] (see Property 1) to effectively reduce the search space. In other words, this property makes the frequent pattern mining practicable in the real-world applications.

*Example 2* Let $x$, $y$, and $z$ be three items in a hypothetical transactional database. The itemset lattice produced by the combinations of these three items is shown in Fig. 1a. The alternative representation of this lattice as a set enumeration tree is shown in Fig. 1b. The size of this lattice is 7 $(= 2^3 - 1)$. This lattice represents the search space for finding frequent patterns. Frequent pattern mining algorithms search this enormous lattice using the anti-monotonic property. For instance, a frequent pattern mining algorithm will check the interestingness of the pattern $xy$ if and only if all of its non-empty subsets, i.e., $x$ and $y$, are also frequent in the database.

**Property 1** ***(Anti-monotonic property.)*** *If $X \subset Y$, then $sup(X) \geq sup(Y)$. Thus, if $sup(X) \ngeq minSup$, then $\forall Y \supset Y$, $sup(Y) \ngeq minSup$.*

*Example 3* The *support* of pattern $cd$ in Table 1, i.e., $sup(cd) = 1$. Since $sup(cd) \ngeq minSup$, $cd$ is an infrequent pattern. Moreover, all supersets of $cd$ will also be infrequent patterns because their *supports* cannot also be more than $minSup$.

[1] The set enumeration tree is a high-performance data representation technique, which resembles the depth-first search on the itemset lattice

[2] Other names of this property are: apriori property and downward closure property.

## 3 Popular Algorithms

Several algorithms have been described in the literature to find frequent patterns. A recent survey on the past 25 years of frequent pattern mining may be found at [19]. In this chapter, we present four of the widely studied frequent pattern mining algorithms, namely Apriori [3], FP-Growth [13], Eclat [46], and LCM [37].

### *3.1 Apriori*

Apriori is one of the fundamental algorithms to find frequent patterns in a database. It is a breadth-first search algorithm that finds all frequent patterns by employing "*level-wise candidate-generate-and-test paradigm*." This paradigm briefly involves the following three steps: (i) find frequent $k$-patterns from the candidate $k$-patterns, (ii) generate candidate $k$-patterns by joining frequent $(k-1)$-patterns among themselves, and (iii) repeat the above two steps until no more candidate $k$-patterns can be generated. The pseudocode of this algorithm is presented in Algorithm 1.

**Algorithm 1** Pseudo-code of the Apriori algorithm.

**Require:** $I, DB, minSup$ {set of items, dataset and minimum support value}
**Ensure:** $F$
1: $F = \emptyset$
2: $L_1 = \{i \in I \mid support(i, DB) \geq minSup\}$
3: $F = F \cup L_1$
4: **for** $(k = 2; L_k \neq \emptyset; k++)$ **do**
5: $C =$ set of candidate patterns produced by $L_{k-1}$
6: $L_k = \{p \in C \mid support(p, DB) \geq minSup\}$
7: $F = F \cup L_k$
8: **end for**
9: **return** $F = \cup_k F_k$

Let us consider the sample transactional database $DB$ shown in Table 1. Let us also consider a minimum support value of three. The following is the set of frequent patterns that can be extracted from DB: $a$, $b$, $c$, $ab$, and $ac$. This set highly varies when the minimum support value is modified. Thus, considering a minimum support value of four, then the resulting set of frequent patterns is reduced: $a$ and $b$. Similarly, if the minimum support value is decreased, then the resulting set of frequent patterns is increased. The following is the set of frequent patterns that can be extracted from DB with a minimum support value of two: $a$, $b$, $c$, $d$, $e$, $ab$, $ac$, $ac$, $bc$, $bd$, $be$, and $abc$. Finally, it is important to remark that determining the exact minimum support value is not trivial and generally requires a profound background in the application field. Inexpert and many expert users need to try different thresholds by guessing and re-executing the algorithms once and again until the results are good for them. As it has been demonstrated, a small change in the threshold value may lead to very few or an extremely large set of solutions.

### 3.2 *Frequent Pattern-Growth*

The popular adoption and successful industrial application of the Apriori algorithm has been hindered by the following two limitations: (i) Apriori algorithm generates too many candidate patterns and (ii) Apriori algorithm requires multiple scans on the database. Han et al. [13] introduced Frequent Pattern-growth (FP-growth) algorithm to address the limitations of Apriori algorithm. It is a depth-first search algorithm that finds the desired patterns by employing the following two steps: (i) compress the given database into a $tree$ structure known as Frequent Pattern-tree (FP-tree) and (ii) find all frequent patterns by recursively mining the FP-tree.

A really efficient algorithm for mining frequent patterns was proposed by Han et al. [13]. This algorithm, named FP-Growth, achieves a high performance by representing the data as a tree structure [31] in which nodes denote items together with their frequency, and paths among nodes represent the patterns (set of connected nodes). This compressed representation of the input dataset drastically reduces the number of scans required to compute the patterns and their frequencies.

To understand how the FP-Growth algorithm works, let us explain first how to construct the tree structure (see Fig. 2) by taking the sample transactional dataset DB (see Table 1). The first step is to calculate the support value for each item: $support(a, DB) = 4$, $support(b, DB) = 4$, $support(c, DB) = 3$, $support$

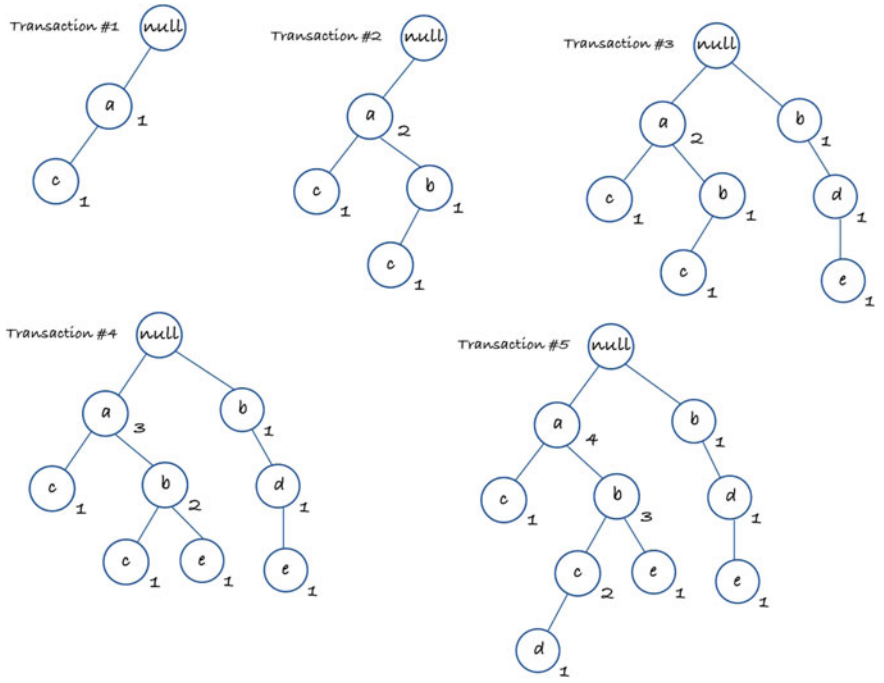


**Fig. 2** Building a tree structure for the sample dataset shown in Table 1

$(d, DB) = 2, support(e, DB) = 2$. Items are, therefore, sorted in descending order of support in each transaction: $a \prec b \prec c \prec d \prec e$. After that, the process starts with the empty node *null*, and each transaction of DB is analyzed. Taking the first transaction, it comprises items $ac$ and they are included as a branch from the root (the empty node). Each time a node is added to the tree, the value 1 is assigned to it as the frequency. If a transaction shares items of any existing branch in the tree, then the inserted transaction will be in the same path from the root to the common prefix. The values of the common nodes increase by one. Let us see the second transaction. The common path is $a$, so its value is increased by one (see transaction #2 in Fig. 2). The rest of the items is added as a new branch from the common path. The frequencies for the new nodes are initialized to one. The third transaction does not share any path with the tree, so it is added as a new branch from the root, that is, the *null* node. The process iterates till all the transactions are analyzed, and the resulting tree is obtained as a compressed data format. It is important to highlight that a fixed order among the items is required or the resulting tree structure will be meaningless.

The pseudocode of the FP-Growth algorithm is illustrated in Algorithm 2. To construct the tree, the algorithm takes frequent items (singletons) from data. Considering a minimum frequency value of 2, the following items are considered: $a$ appears four times, $b$ appears four times, $c$ appears three times, $d$ appears two times, and $e$ appears two times. Then, the tree is constructed following that order as previously described. Once the tree is obtained, FP-Growth first takes the lowest node, that is, item $e$, which occurs in 2 branches: $abe$ with frequency 1 and $bde$ with frequency 1. From these branches, only one frequent pattern can be obtained: $be$, with a frequency of 2. The process is repeated for each item. Taking now the item $d$, two branches are analyzed: $abcd$ and $bd$. Combining them, the frequent pattern $bd$ is obtained,

**Algorithm 2** Pseudo-code of the FP-Growth algorithm.

**Require:** $T, \alpha, minSup$ {Tree-structure, initial node, set of items and minimum support}
**Ensure:** $F$
$F = \emptyset$
**if** $T$ contains a single path $P$ **then**
  **for** each combination $\beta$ of nodes in $P$ **do**
    generate the pattern $X = \alpha \cup \beta$
    support of $X$ is the minimum support of nodes in $\beta$
  **end for**
**else**
  **for** each $a$ in the header of $T$ **do**
    generate the pattern $X = \alpha \cup a$
    support of $X$ is the support of $a$
    construct a conditional pattern base of $X$
    construct a conditional FP-Tree $T_X$ from $X$
    **if** $T_X$ is not empty **then**
      recursive call of FP-Growth($T_X, X, minSup$)
    **end if**
  **end for**
**end if**
**return** $F$

with a frequency of 2. As for the item $c$, two branches are analyzed: $ac$ and $abc$. The resulting frequent pattern is $ac$. The process iterates over all the items. As it is demonstrated, once the tree structure is built, no further passes over the dataset are necessary. Any frequent pattern can be obtained directly from the tree by exploring the tree from the bottom-up (considering the items as suffixes).

Research studies on FP-Growth determined that this algorithm is efficient and scalable. Its performance is calculated as an order of magnitude faster than Apriori. Its main drawback is building and traversing of the tree structure, which is not trivial in huge datasets.

## *3.3 ECLAT*

Eclat (Equivalent CLAss Transformation) was proposed in 1997 [46] as the first algorithm for mining frequent patterns that work on a vertical data representation. This data representation represents the items as lists including the transactions (tids) in which each item appears. Back to the sample transactional database DB shown in Table 1, a vertical data representation of DB is illustrated in Fig. 3. The list formed by item $a$ includes transactions number 1, 2, 4, and 5. As for the item $b$, it includes transactions number 2, 3, 4, and 5. Eclat considers the vertical data representation as a set of tidsets or pointers to the transaction tids including each item. This algorithm computes the support of each item by simply calculating the length of the tidsets. Thus, the items $a$ and $b$ have a support value of 4 since they appear in 4 transactions. The support of $c$ is 3, and the support of both $d$ and $e$ is 2.

Eclat works by combining tidsets, so for each frequent pattern of length $k$, a candidate pattern of length $k + 1$ is produced by adding the singleton that lexicographically follows according to those items included in the original frequent pattern (see Algorithm 3). A new tidset is obtained from the intersection of both tidsets (singleton and original frequent pattern) and its size is denoted as the frequency of the resulting pattern. The algorithm follows a breadth-first search strategy taking patterns and doing intersections with the next item in lexicographical order. For

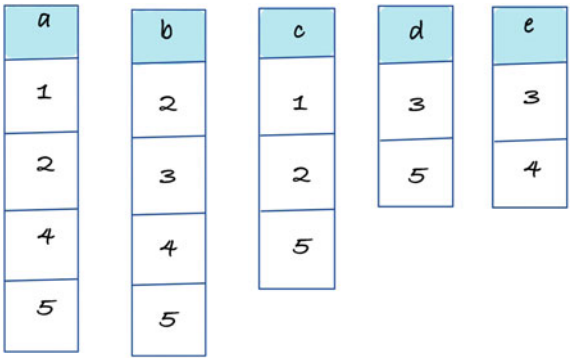


**Fig. 3** Tidsets for the sample dataset shown in Table 1

**Algorithm 3** Pseudo-code of the Eclat algorithm.

**Require:** $I, minSup$ {set of items, and minimum support value}
**Ensure:** $F$
1: $TidList =$ compute the tidsets of all the items in $I$
2: $L = \{l \in TidList \mid length(l) \geq minSup\}$
3: $F = F \cup L$
4: **for** $\forall\, l \in L$ **do**
5: $c = l$
6: **for** $\forall\, m \in L \mid m > j$ **do**
7: $c' = c \cap m$
8: **if** $length(c') \geq minSup$ **then**
9: $c = c'$
10: $F = F \cup c$
11: **end if**
12: **end for**
13: **end for**
14: **return** $F$

example, taking the item $a$ and considering a minimum frequency value of 2, it is combined with $a$ to form the pattern $ab$. The intersection of their tidsets is the new set $\{2, 4, 5\}$, and therefore, it is a frequent pattern since its frequency is 3 (length of the tidset). In an iterative process, the pattern $ab$ is combined with the next item in lexicographical order, that is, $c$. The resulting tidset is $\{2, 5\}$, so its frequency is 2. The resulting pattern $abc$ is now combined with $d$, obtaining the tidset $\{5\}$. Hence, this pattern is infrequent since its frequency is lower than 2. As a result, the process does not continue with the next item and it goes back to combine $a$ with $c$.

Finally, let us summarize the shortcomings of Eclat. This algorithm does not require to scan the dataset to find the frequency of new patterns as Apriori does. However, the bottleneck comes when the number of transactions increases, requiring huge memory and computational time for intersecting the tidsets.

## *3.4 LCM*

LCM (Linear Closed itemset Miner) [37] is a really fast algorithm proposed in 2003 for mining frequent patterns. This highly optimized algorithm won the FIMI2003 competition in which many efficient algorithms were proposed. LCM does not follow a single data representation but two of them, so it horizontally stores transactions and it also keeps the id of the transactions for each item (vertical data representation). What makes LCM be so fast and efficient is not the data representation but some alluring ideas that it includes. Authors of this algorithm clearly stated [37] that the pruning process usually included in FPM algorithms is not complete, and it is common to operate unnecessary frequent patterns. Taking it into consideration, they overcome the problem with a hypercube decomposition, also known as perfect extension pruning. An occurrence deliver schema is an additional promising

**Algorithm 4** Pseudo-code of the occurrence deliver schema.

**Require:** $P$, $DB_P$ {Pattern to be analyzed, conditional dataset}
**Ensure:** $Bucket$
1: **for** $\forall\, i \in P$ **do**
2: Set $Bucket[i] = \emptyset$
3: **end for**
4: **for** $\forall\, t \in DB_P$ **do**
5: **for** $\forall\, e \in t$ **do**
6: Insert $t$ into $Bucket[e]$
7: **end for**
8: **end for**

idea included in LCM to easily calculate the frequency of each pattern. The occurrence deliver schema (see Algorithm 4) takes as input a pattern $P$ and a conditional database given $P$. It creates the buckets for each item in $P$ and adds into buckets those transactions including the item. First, the algorithm orders the items based on their frequencies and removing infrequent items from data. To understand how this procedure works, let us take the transactional database DB shown in Table 1 in which items are sorted in ascending order of support in each transaction: $d \prec e \prec c \prec a \prec b$. Following the occurrence deliver schema illustrated in Algorithm 4, the following buckets are obtained (see Figure 4): $Bucket[a] = \{1, 2, 4, 5\}$, $Bucket[b] = \{2, 3, 4, 5\}$, $Bucket[c] = \{1, 2, 5\}$, $Bucket[d] = \{3, 5\}$, and $Bucket[e] = \{3, 4\}$. Thus, it is easy to know that the pattern $a$ is satisfied by 4 transactions, pattern $b$ by 4 transactions, pattern $c$ by 3 transactions, and so on. Taking the first item in the list of sorted items, that is, item $d$, a projection is performed and new buckets are obtained for that projected database (see Algorithm 4) as it is illustrated in Figure 4. Thus, given the prefix $d$, it is easy to check that the itemset $da$ appears once ($Bucket[a] = \{5\}$), whereas the itemset $db$ appears 2 times ($Bucket[b] = \{3, 5\}$). The process is repeated recursively (a depth-first search strategy is considered) for the remaining items so the conditional database are smaller and smaller.

As for the hypercube decomposition technique, it aims to improve the searching process by stopping the recursive process of forming patterns. Once a perfect extension is detected, then it is directly reported. A perfect extension of a pattern $I$ is an item $i$ that satisfies $i \notin I$, and the frequency of $I$ and $I \cup i$ is exactly the same. Perfect extensions have the following properties: If the item $i$ is a perfect extension of a pattern $I$, then $i$ is also a perfect extension of any pattern $Q$ such as $I \subseteq Q$ as long as $i \notin Q$; If $E$ is the set of all perfect extensions of the pattern $I$, then all sets $I \cup Q$ with $Q \in 2^E$ (the power set of the set $E$) have the same support as $I$. Back to the sample transactional dataset shown in Table 1, the item $a$ is a perfect extension of the pattern $bc$ since the frequency of $bc$ is exactly the same as $abc$, that is, 2. Consequently, the item $a$ is also a perfect extension of the pattern $b$ and $c$.

A new LCM version was proposed one year later [39], and it was granted for a second year in a row with the best implementation award in FIMI2004 competition. The new version introduced an improvement in runtime by reducing data. The algorithm deletes an item if it is not present in at least $\alpha$ data records or if it is present in

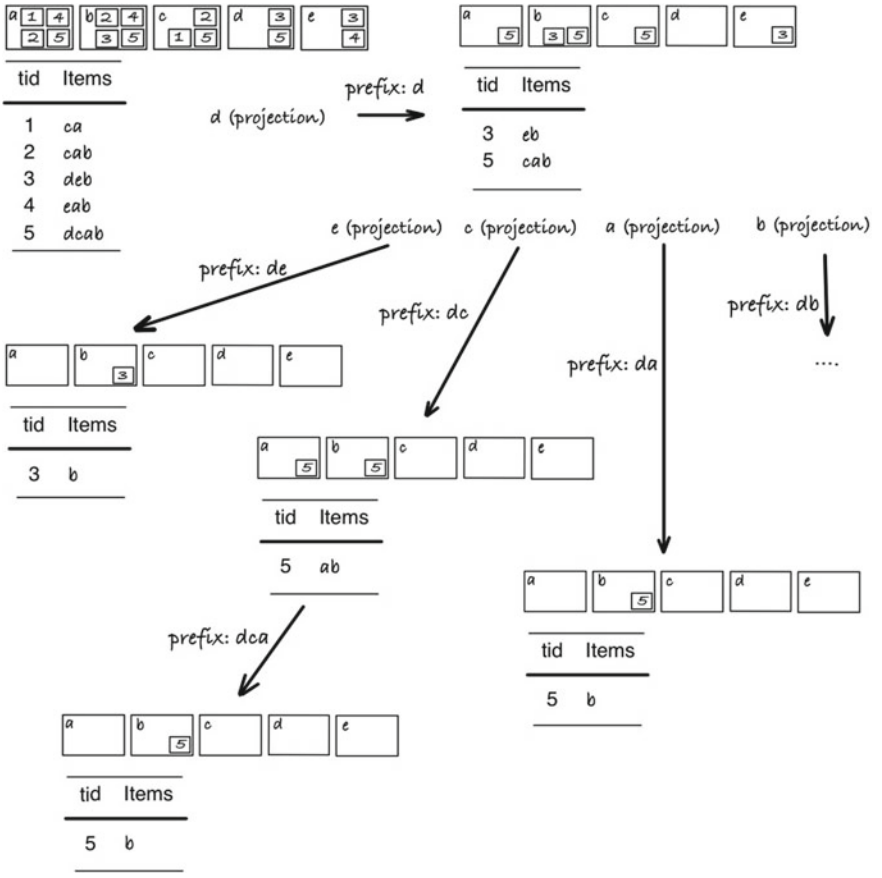


**Fig. 4** Sample occurrence deliver using buckets on dataset shown in Table 1

all data records. Additionally, identical data records are merged into a single one. A third LCM version was also proposed one year later [38], and it is considered as the fastest version up to date. This version includes really efficient structures in the FPM task: bitmaps, array lists, and prefix trees. A bitmap is efficient for dense datasets since it enables fast intersections/unions to be performed with not so much memory consumption. The $k$ most frequent items are kept, known as *k-items machine* in the literature. Those items not included in the bitmap representation are stored as array lists and considered to build a prefix tree. It is essential to highlight that this third version also comprises an occurrence deliver technique, the database reduction, and the hypercube decomposition technique proposed in previous versions.

# 4 Types of Patterns

The growing interest in pattern mining has encouraged the definition of a new type of pattern according to the analysis required by experts in different application fields.

## *4.1 Maximal Frequent Patterns and Closed Frequent Patterns*

Since the objective of FPM is to find all frequently occurring patterns in the database, this model often produces too many patterns most of which may be uninteresting to the users. Moreover, the computational cost of finding these huge number of frequent patterns may not be non-trivial.

*Example 4* The frequent pattern model not only finds $abc$ as a frequent pattern in Table 1, but also finds all of its non-empty subsets, i.e., $a$, $b$, $c$, $ab$, $ac$, and $bc$ as frequent patterns. Thus, producing too many patterns, most of which are uninteresting or redundant to the user.

When encountered with this problem in real-world applications, researchers have tried to find a reduced set of frequent patterns, namely *maximal frequent patterns* and *closed frequent patterns*. We now briefly discuss both of these patterns.

**Definition 1 (Maximal frequent pattern $X$.)** A frequent pattern $X$ is said to be a *maximal frequent pattern* if $sup(X) \geq minSup$ and $\forall Y \supset X,\ sup(Y) \ngeq minSup$.

*Example 5* The frequent pattern $abc$ in Table 1 is a maximal frequent pattern because all of its supersets are infrequent patterns. Moreover, all non-empty subsets of $abc$ cannot be maximal frequent patterns. Thus, maximal frequent pattern mining significantly reduces the number of patterns being discovered in a database.

Maximal frequent pattern mining helps us to find long patterns in a database effectively. However, they lead to a loss of information as they do not record the $support$ information of its subsets. This motivated researchers to find closed frequent patterns in a database. A closed frequent pattern is a frequent pattern that is not strictly included in another pattern having the same frequency. Thus, closed frequent patterns are lossless by nature as they preserve the $support$ information of all frequent patterns in a database.

**Definition 2 (Closed frequent pattern $X$.)** Let $X$ and $Y$ be two frequent patterns such that $X \subset Y$. The frequent pattern $X$ is said to be a closed frequent pattern if $sup(X) \neq sup(Y)$, $sup(X) \geq minSup$.

*Example 6* Consider the frequent patterns $a$, $c$, and $ac$ in Table 2. Since $sup(a) \neq sup(ac)$, $a$ is a closed frequent pattern. In contrast, $c$ is not a closed frequent pattern because $sup(c) = sup(ac)$.

The relationship between the set of frequent patterns ($F$), the set of closed frequent patterns ($C$), and the set of maximal frequent patterns ($M$) is $F \supseteq C \supseteq M$.

### 4.2 Condensed Patterns

As previously stated, the mining of frequent patterns [1] is the keystone in data analysis and the extraction of useful patterns from data. Many efficient approaches have been proposed but it is still expensive to find the complete set of solutions (frequent patterns). The daunting process may be eased by computing a small subset of frequent patterns that can be used to approximate the frequency values of arbitrary frequent patterns. These frequent patterns used to approximate further solutions are known as condensed patterns [25]. The mining of these patterns drastically reduces the runtime but, sometimes, the error in the approximation is not enough from the domain/problem point of view. On some occasions, this process (considering a maximal error bound) is enough even when no full precision is achieved.

The support approximations are calculated through a function $\mathcal{F}$ defined on a transactional database DB. The following can be an approximation function for any pattern $P$: $\mathcal{F}(P) = 0$ if there exists no superset $P' \supseteq P$ such as $P'$ is defined as a condensed pattern; whereas $\mathcal{F}(P) = [support(P', DB) - 3, support(P', DB)]$ being $support(P')$ the minimum support for any $P' \subset P$ such as $P'$ is defined as a condensed pattern. Considering such a function, the support of any pattern can be estimated. Back to the sample transactional database DB (see Table 1), $\mathcal{F}(abcde) = 0$ since there is no $P' \in DB|abcde \subseteq P'$. On the contrary, $\mathcal{F}(ac) = [3 - 3, 3] = [0, 3]$ since the minimum support of any subset of $ac$ is $support(c, DB) = 3$.

This way of dealing with patterns is sometimes preferable to the mining of all the patterns. The following are the main reasons: (1) When dealing with really large datasets, small deviations often have minor effects on the analysis; (2) Computing condensed patterns leads to more efficient approaches since it is only required to operate with and access to a small portion of frequent patterns [29].

### 4.3 Top-k Patterns

In the task of frequent pattern mining, it is common to consider a frequency threshold value to split the search space into useful and useless solutions (patterns). This frequency threshold mainly depends on the users' expectations and the data themselves. Thus, it is the user who has to specify the boundary to consider a pattern a valid solution or not. This boundary will produce a large set of patterns (if the frequency threshold is too low) or a reduced set of patterns (if the frequency threshold is high enough). However, the problem to properly specify such a threshold value is not easy and it also depends on the data distribution. Some datasets may not produce any pattern for a specific frequency value, whereas other datasets may produce tons of patterns for the same frequency value. As a result, it is not trivial to know beforehand the right boundary value for each dataset. Additionally, two main dangers of working with frequency-based algorithms are: 1) setting up an incorrect threshold value may cause an algorithm to fail in finding the interesting patterns; 2) the algorithm may report spurious patterns that do not really exist or are not interesting at all.

To precisely control the output size and discover the patterns with the highest frequency are of vital importance in many application fields [27]. In this regard, it is necessary to extract not only frequent patterns but those having a high significance and low redundancy. Salam et al. [28] proposed an algorithm for mining frequent patterns without any minimum frequency threshold. To achieve this, they proposed two novel algorithms for mining top-most and top-*k* frequent patterns. The top-*k* parameter has been differently used by researchers: Wang et al. [41] employed a top-*k* parameter to extract frequent closed patterns; Tzvetkov et al. [36] considered sequential databases; Cheung et al. [43] retrieved the *k* most frequent patterns; and Chuang et al. [7] used a top-*k* parameter for mining top few significant maximal frequent patterns. Fournier-Viger et al. [35] proposed to redefine the task of mining high utility patterns as mining top-*k* high utility patterns. Luna et al. [23] also proposed a free-parameter algorithm for mining patterns in the form of association rules.

### *4.4 User-Centric Patterns*

Up to date, it is possible to find many research studies in the specialized literature related to different and efficient ways of speeding up the mining of frequent patterns [19]. Currently, more and more research studies are paying attention to extract patterns according to the users' needs or expectations. The user, as the final consumer of the data insights, should play a relevant role in the mining process and it is not just enough to find any pattern that overcomes some minimum frequency values, but those patterns that provide useful information to the user. Here, it is essential to highlight comprehensibility and flexibility as two challenging research issues in the pattern mining field. Comprehensibility describes the degree by which a user can understand the provided information. This is a subjective measure since a pattern can be little comprehensible for a specific user and, at the same time, too much comprehensible for others. Nevertheless, it is possible to define this metric as an objective measure with a fixed formula: the fewer the number of items included in the extracted pattern, the more comprehensible the pattern is. Flexibility, on the contrary, refers to the ability to adapt the solutions to the users' requirements by introducing subjective knowledge into the mining process. These two features, namely comprehensibility and flexibility, have been widely studied and some authors [24] have proposed the use of grammars in pattern mining to introduce subjective knowledge in the mining process and to produce more flexible and expressive results.

Recently, there are some signs of progress in supplying existing pattern mining approaches [19] with methods to extract more actionable insights [40]. Thus, the user is playing a crucial role in the mining process restricting the search space with various constraints based on his/her subjective knowledge of the problem [18]. Other proposals are focused on using new and more flexible forms of information [12]. New methods are also being considered to handle context-sensitive concepts to avoid any discriminative behavior [21], as well as to extract exceptional behavior [22]. As a result, more and more researchers are paying special attention to the extraction of the appropriate knowledge type [20], accomplishing the users' aims or requirements.

## 4.5 Weighted Frequent Patterns

In traditional frequent pattern mining, items within the solutions are uniformly treated and present the same degree of importance. However, in a real-world scenario, items usually have different importance and it is, therefore, required to weigh them. A simple example is related to the market basket analysis, where expensive items may contribute a large portion of overall revenue even though it does not appear in many data records. Weighted frequent pattern mining [45] has been suggested as a promising task to find important frequent patterns by considering the weights of patterns.

Based on the previous ideas, multiple weighted frequent pattern mining algorithms have been proposed up to date. First proposals, such as MINWAL [6], WARM [33], and WAR [42], were proposed taking the Apriori [3] algorithm as a baseline. All these algorithms follow a breadth-first search methodology, considering a level-wise paradigm in which all the candidate patterns of length $k+1$ are obtained by using all the extracted patterns of length $k$. A major drawback of these algorithms is they require multiple database scans, giving rise to poor performance. *Yun et al.* [44] proposed a more efficient algorithm, named WFIM (Weighted Frequent Itemset Mining), that was the first weighted frequent pattern algorithm based on an FP-tree. This algorithm considered a minimum weight value and a weight range, and the FP-Tree is arranged in weight ascending order. A similar algorithm, known as WCloset, was also proposed but instead of mining weighted frequent patterns, it aimed to extract closed weighted frequent patterns. Ahmed et al. [4] proposed an approach to keep track of the varying weights of each item in a prefix tree. They proposed the DWFPM (dynamic weighted frequent pattern mining) algorithm, which is able to handle dynamic weights during the mining process.

Recently, Uday et al. [14] introduced two pattern-growth algorithms: Sequential Weighted Frequent Pattern-growth and Parallel Weighted Frequent Pattern-growth. These two algorithms, which were designed to discover weighted frequent patterns efficiently, employ three novel pruning techniques to reduce the computational cost effectively. The first technique, called cutoff-weight, prunes uninteresting items in the database. The second pruning technique, called conditional pattern base elimination, eliminates the construction of conditional pattern bases if a suffix item is an uninteresting item. The third pruning technique, called pattern-growth termination, proposes a new terminating condition for the pattern-growth technique.

## 4.6 High Utility Patterns

Frequent pattern mining is a widely known and useful task, but it has three essential limitations. First, the item purchase quantities are not considered in the data records. As a result, any item is considered equally important than buying a single unit. In market basket analysis, this is a useless analysis since customers tend to buy more than a single unit. Second, all items are considered as equally important, which does

not show a real situation if a real-world problem is considered. It is more important to sell a laptop than a keyboard since the former yields a much higher profit. Third, the frequency of the patterns could be useless for the user, which is more interested in the amount of profit. High utility pattern mining has recently emerged to address all these limitations [34].

The high utility problem is, therefore, a generalization of frequent pattern mining, considering that every single item has a different utility or relative importance. Additionally, each item in the high utility problem can appear more than once in a data record or transaction. The aim of high utility pattern mining is to find all patterns having a utility higher than a predefined threshold value. Research studies have proposed really interesting and efficient approaches [9]. However, all of them have an important limitation since in the high utility pattern mining problem, the anti-monotonicity property does not hold for the utility measure. The first algorithm for mining high utility patterns is called Two-Phase [10]. This algorithm is an extension of the Apriori algorithm [3] considering an upper-bound on the utility, called Transaction Weighted Utilization (TWU), which is anti-monotonic. After that, really efficient algorithms were proposed such as UP-Growth [34]. However, these algorithms overestimate their utility, so it is computationally hard to calculate their real utility. To overcome such limitations, different researchers have proposed to ease the utility calculation: HUI-Miner [17], FHM+ [8], and EFIM [47], to list a few. Recently, high utility pattern mining algorithms are considering the time at which transactions were made [11]. Additionally, some research studies [16] focus on finding high utility patterns that periodically appear in data.

## *4.7 Periodic-Frequent Patterns*

Periodic-frequent patterns were introduced by Tanbeer et al. [32] as a way of determining the interestingness of frequent patterns in terms of the shape of occurrence. In other words, whether frequent patterns occur periodically, irregularly, or mostly in the specific time interval in the dataset. For example, the shopkeeper in a retail market may be interested only in those products that were regularly sold compared to the rest.

A frequent pattern is defined as a periodic-frequent pattern if it appears and maintains a similar period/interval in a database. To put it in another way, a frequent pattern is said to be periodic-frequent if it occurs at regular intervals specified by the user in data. More technically, a pattern is called a periodic-frequent pattern if it satisfies both of the following two criteria: 1) its periodicity is no greater than a user-given maximum periodicity threshold value; 2) its support is no less than a user-given minimum support threshold value. As a result, the periodic-frequent pattern mining problem is to discover the complete set of periodic-frequent patterns in a database satisfying the two aforementioned criteria.

Tanbeer et al. [32] proposed the first algorithm for mining periodic-frequent patterns. This algorithm, known as PFP-growth, is a pattern-growth approach that

generates periodic-frequent patterns by applying depth-first search in the pattern lattice. From a single item $i$ that satisfies the requirements (frequency and periodicity), PFP-growth obtains larger patterns by adding one item at a time. Additionally, thanks to the periodicity measure, which ensures the anti-monotonic property, the algorithm is quite efficient in discovering the complete set of periodic-frequent patterns. After this first approach, many research studies have paid attention to this interesting task giving rise to really efficient approaches. Kiran et al. [15] and Surana et al. [30] enhanced the PFP-growth algorithm to address the problem of rare or infrequent pattern mining. Amphawan et al. [5] proposed an efficient algorithm for mining top-$k$ periodic-frequent patterns, reducing the resulting set and making it more understandable for the end-user. Rashid et al. [26] employed standard deviation of periods as an alternative criterion to assess the periodic behavior of frequent patterns. They considered extensions of the well-known PFP-growth algorithm [32] to obtain a resulting set of solutions known as regular frequent patterns.

## References

1. C.C. Aggarwal, J. Han, *Frequent Pattern Mining* (Springer International Publishing, Berlin, 2014)
2. C.C. Aggarwal, J. Han, *Frequent Pattern Mining* (Springer Publishing Company, Berlin, 2014)
3. R. Agrawal, T. Imielinski, and A.N. Swami, Mining association rules between sets of items in large databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD Conference'93 (ACM, New York, 1993), pp. 207–216
4. C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong and Y.-K. Lee, Handling dynamic weights in weighted frequent pattern mining. *IEICE Trans. Inf. Syst.*, **91-D**(11), 2578–2588 (2008)
5. K. Amphawan, P. Lenca and A. Surarerks, Mining top-k periodic-frequent pattern from transactional databases without support threshold. Advances in Information Technology—Third International Conference, IAIT 2009 (Springer, Berlin, 2009), pp. 18–29
6. C.H. Cai, A.W.C. Fu, C.H. Cheng and W.W. Kwong, Mining association rules with weighted items. Proceedings of the 1998 International Database Engineering and Applications Symposium, IDEAS 1998 (IEEE, New York, 1998), pp. 68–77
7. Kun-Ta. Chuang, Jiun-Long. Huang, Ming-Syan. Chen, Mining top-k frequent patterns in the presence of the memory constraint. The VLDB Journal **17**(5), 1321–1344 (2008)
8. Philippe Fournier-Viger, Jerry Chun-Wei Lin, Quang-Huy Duong, and Thu-Lan Dam. FHM + : Faster high-utility itemset mining using length upper-bound reduction. In *Trends in Applied Knowledge-Based Systems and Data Science - 29th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2016, Morioka, Japan, August 2-4, 2016, Proceedings*, pages 115–127, 2016
9. Philippe Fournier-Viger, Jerry Chun-Wei Lin, Roger Nkambou, Bay Vo, and Vincent S. Tseng. *High-Utility Pattern Mining: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2019
10. Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S. Tseng. FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In *Foundations of Intelligent Systems - 21st International Symposium, ISMIS 2014, Roskilde, Denmark, June 25-27, 2014. Proceedings*, pages 83–92, 2014
11. Philippe Fournier-Viger and Souleymane Zida. FOSHU: faster on-shelf high utility itemset mining - with or without negative unit profit. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 857–864, 2015

12. J. Han, Y. Fu, Mining multiple-level association rules in large databases. IEEE Transactions on Knowledge and Data Engineering **11**(5), 798–804 (1999)
13. Jiawei Han, Jian Pei, Yiwen Yin, Mining frequent patterns without candidate generation. SIGMOD Rec. **29**(2), 1–12 (2000)
14. R. Uday Kiran, Amulya Kotni, P. Krishna Reddy, Masashi Toyoda, Subhash Bhalla, and Masaru Kitsuregawa. Efficient discovery of weighted frequent itemsets in very large transactional databases: A re-visit. In Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He, and Jeffrey S. Saltz, editors, *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 723–732. IEEE, 2018
15. R. Uday Kiran and P. Krishna Reddy. Towards efficient mining of periodic-frequent patterns in transactional databases. In *Database and Expert Systems Applications, 21th International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part II*, pages 194–208, 2010
16. Yu-Feng. Lin, Wu. Cheng-Wei, Chien-Feng. Huang, Vincent S. Tseng, Discovering utility-based episode rules in complex event sequences. Expert Syst. Appl. **42**(12), 5303–5314 (2015)
17. Mengchi Liu and Jun-Feng Qu. Mining high utility itemsets without candidate generation. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 55–64, 2012
18. J.M. Luna, J.R. Romero, S. Ventura, Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules. Knowledge and Information Systems **32**(1), 53–76 (2012)
19. José María Luna, Philippe Fournier-Viger, and Sebastián Ventura. Frequent itemset mining: A 25 years review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 9(6), 2019
20. J.M. Luna, P. Fournier-Viger and S. Ventura, Extracting user-centric knowledge on two different spaces: Concepts and records. *IEEE Access* **8**, 134782–134799 (2020)
21. José María Luna, Mykola Pechenizkiy, María José del Jesus, and Sebastián Ventura. Mining context-aware association rules using grammar-based genetic programming. *IEEE Trans. Cybern.*, 48(11):3030–3044, 2018
22. José María Luna, Mykola Pechenizkiy, Wouter Duivesteijn, and Sebastián Ventura. Exceptional in so many ways - discovering descriptors that display exceptional behavior on contrasting scenarios. *IEEE Access*, 8:200982–200994, 2020
23. José María Luna, José Raúl Romero, Cristóbal Romero, and Sebastián Ventura. Reducing gaps in quantitative association rules: A genetic programming free-parameter algorithm. *Integr. Comput. Aided Eng.*, 21(4):321–337, 2014
24. José María Luna, José Raúl Romero, and Sebastián Ventura. Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules. *Knowl. Inf. Syst.*, 32(1):53–76, 2012
25. J. Pei, G. Dong, W. Zou, J. Han, Mining Condensed Frequent-Pattern Bases. Knowledge and Information Systems **6**(5), 570–594 (2004)
26. Md. Mamunur Rashid, Md. Rezaul Karim, Byeong-Soo Jeong, and Ho-Jin Choi. Efficient mining regularly frequent patterns in transactional databases. In *Proceedings of the 17th International Conference on Database Systems for Advanced Applications - Volume Part I*, DASFAA'12, page 258-271, Berlin, Heidelberg, 2012. Springer-Verlag
27. Cristóbal Romero, Amelia Zafra, José María Luna, and Sebastián Ventura. Association rule mining using genetic programming to provide feedback to instructors from multiple-choice quiz data. *Expert Syst. J. Knowl. Eng.*, 30(2):162–172, 2013
28. Abdus Salam and M. Sikandar Hayat Khayal. Mining top-k frequent patterns without minimum support threshold. *Knowledge and Information Systems*, 30(1):57-86, 2012
29. A. Soulet, B. Crémilleux, Adequate condensed representations of patterns. Data Mining and Knowledge Discovery **17**(1), 94–110 (2008)
30. Akshat Surana, R. Uday Kiran, and P. Krishna Reddy. An efficient approach to mine periodic-frequent patterns in transactional databases. In *New Frontiers in Applied Data Mining - PAKDD 2011 International Workshops, Shenzhen, China, May 24-27, 2011, Revised Selected Papers*, pages 254–266, 2011

31. P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005
32. Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. Discovering periodic-frequent patterns in transactional databases. In *Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, 2009, Proceedings*, pages 242–253, 2009
33. Feng Tao, Fionn Murtagh, and Mohsen Farid. Weighted association rule mining using weighted support and significance framework. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 661-666, New York, NY, USA, 2003. Association for Computing Machinery
34. Vincent S. Tseng, Bai-En. Shie, Wu. Cheng-Wei, SYu. Philip, Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. **25**(8), 1772–1786 (2013)
35. Vincent S. Tseng, Wu. Cheng-Wei, Philippe Fournier-Viger, SYu. Philip, Efficient algorithms for mining top-k high utility itemsets. IEEE Trans. Knowl. Data Eng. **28**(1), 54–67 (2016)
36. Petre Tzvetkov, Xifeng Yan, Jiawei Han, TSP: mining top-k closed sequential patterns. Knowl. Inf. Syst. **7**(4), 438–457 (2005)
37. Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. Lcm: An efficient algorithm for enumerating frequent closed item sets. In *Fimi*, volume 90. Citeseer, 2003
38. Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 77–86, 2005
39. Takeaki Uno, Masashi Kiyomi, Hiroki Arimura, et al. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Fimi*, volume 126, 2004
40. S. Ventura and J. M. Luna. *Pattern Mining with Evolutionary Algorithms*. Springer International Publishing, 2016
41. Jianyong Wang, Jiawei Han, Lu. Ying, Petre Tzvetkov, TFP: an efficient algorithm for mining top-k frequent closed itemsets. IEEE Trans. Knowl. Data Eng. **17**(5), 652–664 (2005)
42. Wei Wang, Jiong Yang, SYu. Philip, WAR: weighted association rules for item intensities. Knowl. Inf. Syst. **6**(2), 203–229 (2004)
43. Yin-Ling Cheung and Ada Wai-Chee Fu, Mining frequent itemsets without support threshold: with and without item constraints. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1052–1069 (2004)
44. Unil Yun and John J. Leggett. WFIM: weighted frequent itemset mining with a weight range and a minimum weight. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005*, pages 636–640, 2005
45. Unil Yun and Keun Ho Ryu, Approximate weighted frequent pattern mining with/without noisy environments. Knowl. Based Syst. **24**(1), 73–82 (2011)
46. Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. Parallel algorithms for discovery of association rules. *Data mining and knowledge discovery*, 1(4):343–373, 1997
47. Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent S. Tseng. EFIM: A highly efficient algorithm for high-utility itemset mining. In *Advances in Artificial Intelligence and Soft Computing - 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015, Proceedings, Part I*, pages 530–546, 2015