Advanced Review

A survey of itemset mining

Philippe Fournier-Viger,[1*] Jerry Chun-Wei Lin,[2] Bay Vo,[3,4] Tin Truong Chi,[5] Ji Zhang[6] and Hoai Bac Le[7]

Itemset mining is an important subfield of data mining, which consists of discovering interesting and useful patterns in transaction databases. The traditional task of frequent itemset mining is to discover groups of items (itemsets) that appear frequently together in transactions made by customers. Although itemset mining was designed for market basket analysis, it can be viewed more generally as the task of discovering groups of attribute values frequently cooccurring in databases. Because of its numerous applications in domains such as bioinformatics, text mining, product recommendation, e-learning, and web click stream analysis, itemset mining has become a popular research area. This study provides an up-to-date survey that can serve both as an introduction and as a guide to recent advances and opportunities in the field. The problem of frequent itemset mining and its applications are described. Moreover, main approaches and strategies to solve itemset mining problems are presented, as well as their characteristics are provided. Limitations of traditional frequent itemset mining approaches are also highlighted, and extensions of the task of itemset mining are presented such as high-utility itemset mining, rare itemset mining, fuzzy itemset mining, and uncertain itemset mining. This study also discusses research opportunities and the relationship to other popular pattern mining problems, such as sequential pattern mining, episode mining, subgraph mining, and association rule mining. Main open-source libraries of itemset mining implementations are also briefly presented.

How to cite this article:
WIREs Data Mining Knowl Discov 2017, e1207. doi: 10.1002/widm.1207

INTRODUCTION

The goal of data mining is to predict the future or to understand the past.[1,2] Techniques used for predicting the future such as neural networks are often designed to behave as black boxes because the goal is generally to obtain a model that is as accurate as possible rather than to obtain a model that is explanatory. However, several data mining techniques aim at discovering patterns in data that are understandable by humans. Approaches for discovering patterns in data can be classified by the types of patterns that they discover. Some common types of patterns found in databases are clusters, itemsets, trends, and outliers.[2] This study is a survey that focuses on the discovery of itemsets in databases, a popular data mining task for analyzing symbolic data.

The task of discovering itemsets in databases was introduced in 1993 by Agrawal and Srikant[3] as *large itemset mining*, but it is nowadays called

*Correspondence to: philfv8@yahoo.com

[1]School of Natural Science and Humanities, Harbin Institute of Technology, Shenzhen Graduate School, Shenzhen, China

[2]School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen Graduate School, Shenzhen, China

[3]Faculty of Information Technology, Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam

[4]College of Electronics and Information Engineering, Sejong University, Seoul, Republic of Korea

[5]Department of Mathematics and Informatics, University of DaLat, DaLat, Vietnam

[6]Faculty of Health, Engineering and Sciences, University of Southern Queensland, Toowoomba, Australia

[7]Faculty of Information Technology, Ho Chi Minh City University of Science, Ho Chi Minh City, Vietnam

Conflict of interest: The authors have declared no conflicts of interest for this article.

frequent itemset mining (FIM). The task of FIM is defined as follows. Given a database of customer transactions, FIM consists of discovering groups of items (*itemsets*) that are frequently purchased together by customers. For example, one may analyze a customer transaction database and discover that many customers buy taco shells with peppers. Discovering associations between items is useful to understand customer behavior. For instance, a retail store manager can use this knowledge to take strategic marketing decisions, such as copromoting products or putting them closer on the shelves.

Although FIM was originally proposed for analyzing customer data, it is now viewed as a general data mining task that is applicable to many domains. In fact, a customer transaction database can be more generally viewed as a database of instances describing objects (the transactions), where each object is described using nominal attribute values (the items). Thus, FIM can be equivalently defined as the task of finding attribute values that frequently co-occur in a database. Because many types of data can be represented as transaction databases, FIM has many applications in a wide range of domains, such as bioinformatics,[4] image classification,[5] network traffic analysis,[6,7] analyzing customer reviews,[8] activity monitoring,[9] malware detection,[10] and e-learning,[11] to name just a few. FIM has also been extended in many ways to address specific needs. For example, some extensions of FIM are to discover rare patterns,[12] correlated patterns,[13–15] patterns in sequences[16,17] and graphs,[18] and patterns that generate a high profit.[19–25]

The field of itemset mining is a very active research field, where hundreds of new algorithms are proposed every year. This study provides an up-to-date survey that can serve both as an introduction and as a guide to recent advances and opportunities in the field. The article is organized as follows: first, describes the problem of FIM, and the main techniques employed in FIM; second, discusses popular extensions of the problem of FIM and other problems in data mining that are closely related to FIM; third, discusses research opportunities and open-source implementations for itemset mining; and finally, a conclusion is drawn.

FREQUENT ITEMSET MINING

The problem of FIM is formally defined as follows.[3] Let there be a set of items (symbols) $I = \{i_1, i_2, \ldots i_m\}$. A transaction database $D = \{T_1, T_2 \ldots T_n\}$ is a set of transactions such that each transaction $T_q \subseteq I (1 \leq q \leq m)$ is a set of distinct items, and each transaction T_q has a unique identifier q called its Transaction IDentifier (TID). For example, consider the transaction database shown in Table 1. This database contains five transactions, where the letters a, b, c, d, e represent items bought by customers. For example, the first transaction T_1 represents a customer that has bought the items a, c, and d.

TABLE 1 | Transaction Database

TID	Transaction
T_1	{*a*, *c*, *d*}
T_2	{*b*, *c*, *e*}
T_3	{*a*, *b*, *c*, *e*}
T_4	{*b*, *e*}
T_5	{*a*, *b*, *c*, *e*}

An *itemset* X is a set of items such that $X \subseteq I$. Let the notation $|X|$ denotes 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 ($|X| = k$). The goal of itemset mining is to discover interesting itemsets in a transaction database, i.e., interesting associations between items. In general, in itemset mining, various measures can be used to assess the interestingness of patterns. In FIM, the interestingness of a given itemset is traditionally defined using a measure called the *support*. The support (or *absolute support*) of an itemset X in a database D is denoted as $sup(X)$ and defined as the number of transactions containing X, i.e., $sup(X) = |\{T | X \subseteq T \wedge T \in D\}|$. For example, the support of the itemset $\{a, b\}$ is 2 because this itemset appears in two transactions (T_3 and T_5). Note that some authors prefer to define the support of an itemset X as a ratio. This definition called the *relative support* is $relSup(X) = sup(X)/|D|$. For example, the relative support of the itemset $\{a, b\}$ is 0.4.

The *task of FIM*[3] consists of discovering all *frequent itemsets* in a given transaction database. An itemset X is *frequent* if it has a support that is no less than a given minimum support threshold *minsup* set by the user (i.e., $sup(X) \geq minsup$). For example, if we consider the database shown in Table 1 and that the user has set $minsup = 3$, the task of FIM is to discover all groups of items appearing in at least three transactions. In this case, there are exactly nine frequent itemsets: $\{a\} : 3$, $\{b\} : 4$, $\{c\} : 4$, $\{e\} : 4$, $\{a, c\} : 3$, $\{b, c\} : 3$, $\{b, e\} : 4$, $\{c, e\} : 3$, and $\{b, c, e\} : 3$, where the number besides each itemset indicates its support.

FIM is an enumeration problem. The goal is to enumerate all patterns that meet the minimum support constraint specified by the user. Thus, there is

always a single correct answer to an FIM task. FIM is a difficult problem. The naive approach to solve this problem is to consider all possible itemsets to then output only those meeting the minimum support constraint specified by the user. However, such a naive approach is inefficient for the following reason. If there are m distinct items in a transaction database, there are $2^m - 1$ possible itemsets. If a database has, e.g., 1000 distinct items, the number of possible itemsets is $2^{1000} - 1$ itemsets, which is clearly unmanageable using a naive approach. It is important to note that the FIM problem can be very difficult even for a small database. For example, a database containing a single transaction with 100 items, with *minsup* = 1 still generates a search space of 2^{100} itemsets. Thus, the number of itemsets in the search space generally matters more than the size of the data in FIM. But what influences the number of itemsets in the search space? The number of itemsets depends on how similar the transactions are in the database, and also on how low the *minsup* threshold is set by the user.

To discover frequent itemsets efficiently, it is thus necessary to design algorithms that avoid exploring the search space of all possible itemsets and that process each itemset in the search space as efficiently as possible. Several efficient algorithms have been proposed for FIM. Some of the most commonly used are Apriori,[3] FP-Growth,[26] Eclat,[27] H-Mine,[28] and LCM.[29] All of these algorithms have the same input and the same output. However, the difference is the strategies and data structures that these algorithms employ to discover frequent itemsets efficiently. More specifically, FIM 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 count the support of itemsets to determine if they satisfy the minimum support constraint.

In the rest of this section, we first explain the concept of breadth-first search and depth-first search in itemset mining. Then, we provide an overview of the key techniques used by some of the most popular FIM algorithms and explain their advantages and limitations. These techniques are very important as they have inspired numerous algorithms in the field of pattern mining.

Breadth-First Search and Depth-First Search

Most of the existing itemset mining algorithms can be described as either using a breadth-first search or a depth-first search. Assume that there are m items in a database. A breadth-first search algorithm (also called a level-wise algorithm) such as Apriori explores the search space of itemsets by first considering 1-itemsets, then 2-itemsets, 3-itemsets, and lastly m-itemsets. For example, Figure 1 depicts the search space of all possible itemsets for the running example. In this figure, the search space is represented as a Hasse diagram.[a] A breadth-first search algorithm will first consider 1-itemsets $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$, and $\{e\}$. Then, it will generate 2-itemsets such as $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, and then 3-itemsets, and so on, until it generates the itemset $\{a, b, c, d, e\}$ containing all items. However, depth-first search algorithms such as FPGrowth, H-Mine, and LCM start 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

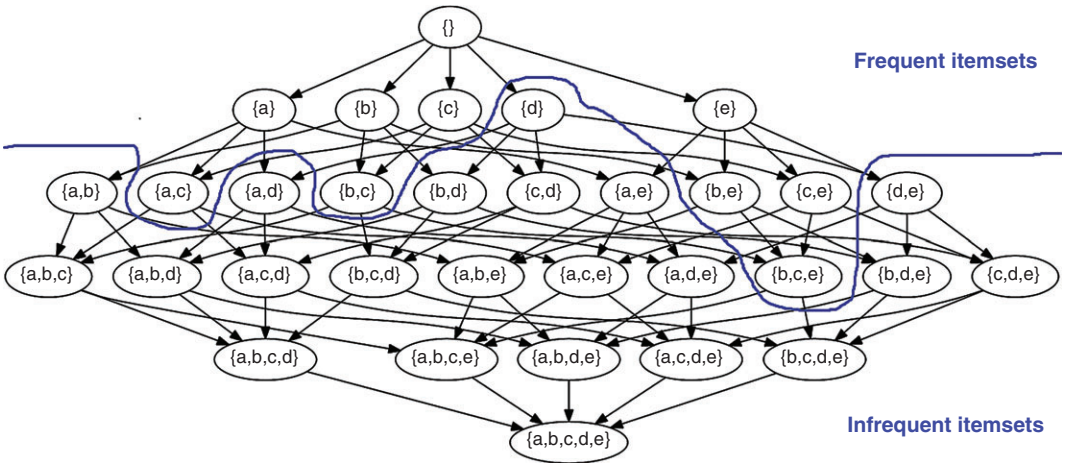

FIGURE 1 | The search space for $I = \{a, b, c, d, e\}$.

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*}.

To design an efficient FIM algorithm, it is important that the algorithm avoid exploring the whole search space of itemsets because the search space can be very large. To reduce the search space, *search space pruning techniques* are used. In FIM, the key observation for reducing the search space is that the support is a monotone measure, i.e., for any itemsets X and Y such that $X \subset Y$, it follows that $sup(X) \geq sup(Y)$.[3] This has the implication that if an itemset is infrequent, all its supersets are also infrequent, and thus do not need to be explored. For example, assuming that *minsup* = 3 and that the support of the itemset {*a*, *b*} is 2, it can be concluded that {*a*, *b*} and all its supersets are infrequent and do not need to be considered. This property, called the *downward-closure property*,[3] *anti-monotonicity property*, or *Apriori property* is very powerful and can greatly reduce the search space. For example, it can be observed in Figure 1 that only 9 itemsets are frequent of 31 possible itemsets (excluding the empty set). Thus, by applying the above property, many itemset can potentially be avoided when exploring the search space.

Apriori: An Horizontal Breadth-First Search Algorithm

Apriori[3] is the first FIM algorithm. Apriori takes a transaction database and the *minsup* threshold as input. Apriori uses a standard database representation, as shown in Table 1, also called a *horizontal database*. The pseudocode of Apriori is given in Algorithm 1. Apriori first scans the database to calculate the support of each item, i.e., 1-itemset (line 1). Then, Apriori uses this information to identify the set of all frequent items, denoted as F_1 (line 2). Then, Apriori performs a breadth-first search to find larger frequent itemsets (lines 4–10). During the search, Apriori uses the frequent itemsets of a given length $k-1$ (denoted as F_{k-1}) to generate potentially frequent itemsets of length k (denoted as C_k). This is performed by combining pairs of items of length k that share all but one item (line 5). For example, if the frequent 1-itemsets are {*a*}, {*b*}, {*c*}, and {*e*}, Apriori will combine pairs of these itemsets to obtain the following candidate 2-itemsets: {*a*, *b*}, {*a*, *c*}, {*a*, *e*}, {*b*, *c*}, {*b*, *e*}, and {*c*, *e*}. After generating candidates of length k, Apriori checks if the $(k-1)$ subsets of each candidate are frequent. If a candidate itemset X has an infrequent $(k-1)$ subset, X cannot be frequent (it would violate the downward-closure property) and it is thus removed from the set of candidate k-itemsets. Then, Apriori scans the database to calculate the support of all remaining candidate itemsets in C_k (line 7). Each candidate having a support not less than *minsup* is added to the set F_k of frequent k-itemsets (line 8). This process is repeated until no candidates can be generated. The set of all frequent itemsets is then returned to the user (line 11).

Algorithm 1: The Apriori algorithm

input : D: a horizontal transaction database, *minsup*: a user-specified threshold

output: the set of frequent itemsets

1 Scan the database to calculate the support of all items in I;
2 $F_1 = \{i | i \in I \wedge sup(\{i\}) \geq minsup\}$; // F_1 : frequent 1-itemsets
3 $k = 2$;
4 **while** $F_k \neq \emptyset$ **do**
5 $C_k =$ CandidateGeneration (F_{k-1}) ; // C_k : candidate k-itemsets
6 Remove each candidate $X \in C_k$ that contains a $(k-1)$-itemset that is not in F_{k-1};
7 Scan the database to calculate the support of each candidate $X \in C_k$;
8 $F_k = \{X | X \in C_k \wedge sup(X) \geq minsup\}$; // F_k : frequent k-itemsets
9 $k = k + 1$;
10 **end**
11 return $\bigcup_{k=1...k} F_k$;

Apriori is an important algorithm as it has inspired many other algorithms. However, it suffers from important limitations. The first one is that because Apriori generates candidates by combining itemsets without looking at the database, it can generate some patterns that do not even appear in the database. Thus, it can spend a huge amount of time processing candidates that do not exist in the database. The second limitation is that Apriori has to repeatedly scan the database to count the support of candidates, which is very costly. The third limitation is that the breadth-first search approach can be quite costly in terms of memory as it requires at any moment to keep in the worst case all k and $k-1$ itemsets in memory (for $k > 1$). In terms of complexity, a very detailed complexity analysis of the Apriori algorithm has been done by Hegland.[30] Briefly, the time complexity is $O(m^2 n)$, where m is the number of distinct items and n is the number of transactions.

Eclat: A Vertical Depth-First Search Algorithm

The Eclat[27] algorithm improves upon the Apriori approach by using a depth-first search to avoid keeping many itemsets in memory. Contrarily to Apriori, Eclat utilizes what is called a *vertical database representation*. A vertical database representation indicates the list of transactions where each item appears.

TABLE 2 | Vertical Representation of the Database of Table 1

Item (x)	TID-Set ($tid(x)$)
a	$\{T_1, T_3, T_5\}$
b	$\{T_2, T_3, T_4, T_5\}$
c	$\{T_1, T_2, T_3, T_5\}$
d	$\{T_1\}$
e	$\{T_2, T_3, T_4, T_5\}$

For an itemset i, the list of transactions containing the item i is called its TID-list, and it is denoted as $tid(X)$. For example, the vertical representation of the database presented in Table 1 is shown in Table 2.

This vertical representation can be obtained by scanning the original horizontal database only once. Furthermore, note that it is also possible to regenerate a horizontal database from a vertical database. The vertical representation is very useful in itemset mining because it possesses the following two properties. First, for any itemsets X and Y, the TID-list of the itemset $X \cup Y$ can be obtained without scanning the original database by intersecting the TID-lists of X and Y, i.e., $tid(X \cup Y) = tid(X) \cap tid(Y)$. Second, the TID-list of an itemset X allows to directly derive its support without scanning the database, by using the property that $sup(X) = |tid(X)|$. For example, the TID-list of $\{a, c\}$ can be calculated as $tid(\{a, c\}) = tid(a) \cap tid(c) = \{T_1, T_3, T_5\}$, and it can thus be derived that the support of $\{a, c\}$ is $|tid(\{a, c\})| = 3$. Thus, using these two properties, *vertical algorithms* such as Eclat can explore the search space by scanning the database only once to create the initial TID-lists. Candidate generation and support counting are carried out directly without scanning the database.

The pseudocode of Eclat is shown in Algorithm 2. The Eclat algorithm takes as input a vertical database (a set R containing all items annotated with their tidsets, as shown in Table 2) and the *minsup* threshold. The Eclat algorithm performs a loop where it considers each itemset X in R that is frequent according to its tidset (lines 2–10). The itemset X is first output. Then, a search is performed to find frequent itemsets extending X with one item. This is performed by attempting to combine each itemset Y in R that shares all but the last item with X to obtain an itemset $X \cup Y$ (lines 4–10). For example, if $X = \{a\}$, Eclat will attempt to combine X with $\{b\}$, $\{c\}$, $\{d\}$, and $\{e\}$ to generate the extensions $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, and $\{a, e\}$, respectively. During this process, the tidset of an extension $X \cup Y$ is calculated as $tid(X) \cap tid(Y)$ (line 6). Then, if $X \cup Y$ is frequent according to its tidset, $X \cup Y$ is added to a set E of frequent extensions of X (lines 6 and 7). After that the Eclat algorithm is recursively called with E to explore all extensions of $X \cup Y$. Then, the loop of lines 1–10 is repeated for other itemsets in R. When the algorithm terminates, all frequent itemsets have been output.

Algorithm 2: The Eclat algorithm

input : R: a set of itemsets with their tidsets, *minsup*: a user-specified threshold

output: the set of frequent itemsets

```
1  foreach itemset X ∈ R such that |tid(X)| ≥ minsup do
2      Output X;                                          // X is a frequent itemset
3      E = ∅;                          // frequent itemsets that are extensions of X
4      foreach itemset Y ∈ R sharing all but the last item with X do
5          tid(X ∪ Y) = tid(X) ∩ tid(Y);          // calculate the tidset of X ∪ Y
6          if |tid(X ∪ Y)| ≥ minsup then                  // if X ∪ Y is frequent
7              E = E ∪ {X ∪ Y};          // add X ∪ Y to frequent extensions of X
8          end
9          Eclat (E, minsup);                             // recursive call using E
10     end
11 end
```

The Eclat algorithm is considered to be a depth-first search algorithm as it outputs all frequent itemsets according to the depth-first search order. Eclat is generally much faster than Apriori as it does not perform multiple database scans. However, Eclat still has some drawbacks. First, because Eclat also generates candidates without scanning the database, it can spend time considering itemsets that do not exist in the database. Second, although TID-lists are useful, they can consume a lot of memory especially for dense datasets (datasets where all items appear in almost all transactions). It is important to note however, that there has been some work to reduce the size of TID-lists using an improved structure called diffsets.[31] Also other proposed improvements are, e.g., to encode TID-lists as bit vectors[31,32] to reduce the memory usage and the speed of intersecting TID-lists, on dense datasets. TID-lists can also be used in breadth-first search algorithms. For example, Apriori-TID is a version of Apriori that relies on TID-lists to calculate the support of itemsets.[3]

Pattern-Growth Algorithms

To address the main limitation of algorithms, such as Apriori and Eclat, a major advance in the field has been the development of *pattern-growth algorithms*, such as FP-Growth,[26] H-Mine,[28] and LCM.[29] The main idea of pattern-growth algorithms is to scan a database to find itemsets, and thus avoid generating candidates that do not appear in the database. Furthermore, to reduce the cost of scanning the database, pattern-growth algorithms have introduced the concept of *projected database* to reduce the size of

databases as an algorithm explore larger itemsets with its depth-first search.

The pseudocode of a typical pattern-growth algorithm is shown in Algorithm 3. It takes as input a transaction database D, the empty set, and the *minsup* threshold. Without loss of generality, assume that there exists a total order on items $\prec$ such as the lexicographical order ($a \prec b \prec c \prec d \prec e$). A pattern-growth algorithm explores the search space using a depth-first search by recursively appending items according to the $\prec$ order to frequent itemsets, to obtain larger frequent itemsets. At the beginning, a pattern-growth algorithm considers that the current itemset X is the empty set. A pattern-growth algorithm scans the database D to find the set Z of all frequent items in D (line 1). Then, for each such item z, the itemset $X \cup \{z\}$ is output as it is a frequent itemset (line 3). 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 5). 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 4) and will use this database to perform the depth-first search (line 5). This will allow reducing the cost of scanning the database. After recursively performing the depth-first search for all items, the set of all frequent itemsets will have been output.

Algorithm 3: A pattern-growth algorithm

input : D: a transaction database, X: the current itemset (initially $X = \emptyset$),
minsup: a user-specified threshold

output: the set of frequent itemsets

1 Scan the database D to find the set Z of all frequent items in D;
2 **foreach** item $z \in Z$ **do**
3 Output $X \cup \{z\}$; // $X \cup \{z\}$ is a frequent itemset
4 $D' = \text{Projection}(D, z)$; // Create projected database of $X \cup \{z\}$
5 PatternGrowth $(D, X \cup \{z\}, minsup)$; // recursive call to extend $X \cup \{z\}$
6 **end**

Now, let us illustrate these steps in more details with an example. Assume that *minsup* = 3. By scanning the database of Table 1, it can be found that the frequent 1-itemsets are a, b, c, and e. The algorithm will first consider the item a to try to find larger frequent itemsets starting with the prefix $\{a\}$. The algorithm will thus build the projected database of $\{a\}$ as shown in Table 3. 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 frequent itemsets starting with $\{a\}$ containing one more item, the algorithm will scan the projected database of $\{a\}$ and count the support of all items appearing in that database. For example, the support of items in the projected database of $\{a\}$ are: $\{b\} : 2$, $\{c\} : 3$, and $\{e\} : 1$. This means that the support of $\{a, b\}$ is 2, that the support of $\{a, c\}$ is 3, and that the support of $\{a, e\}$ is 1. Thus, only the itemset $\{a, c\}$ is frequent (recall that we assume that *minsup* = 3 in the running example). Then the algorithm will pursue the depth-first search to find frequent itemsets starting with the prefix $\{a, c\}$. The algorithm will build the projected database of $\{a, c\}$ from the projected database of $\{a\}$. The projected database of $\{a, c\}$ is shown in Table 4. Then, the algorithm will scan the projected database of $\{a, c\}$ to find frequent items in that database. This process will continue until all frequent itemsets have been explored by the depth-first search.

TABLE 3 | Projected Database of $\{a\}$

TID	Transaction
T_1	$\{c, d\}$
T_3	$\{b, c, e\}$
T_5	$\{b, c, e\}$

TABLE 4 | Projected Database of $\{a, c\}$

TID	Transaction
T_1	$\{d\}$
T_3	$\{e\}$
T_5	$\{e\}$

A major advantage of pattern-growth algorithms is that they only explore the frequent itemsets in the search space thus avoiding considering many itemsets not appearing in the database, or infrequent itemsets. Besides, the concept of projected database is also useful to reduce the cost of database scans. 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.[28,29] For example, Figure 2 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

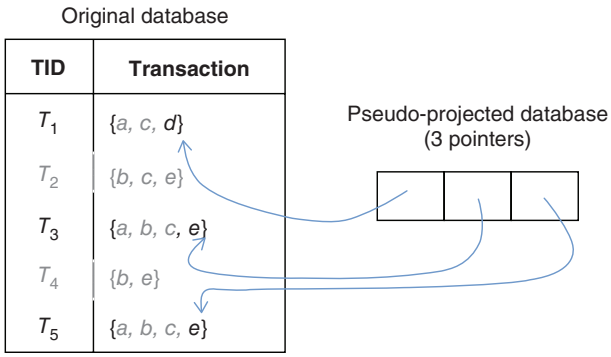

FIGURE 2 | The pseudo-projected database of $\{a,c\}$.

optimizations can also be integrated in pattern-growth algorithms. For example, LCM[29] also integrates a mechanism to merge identical transactions in projected databases to further reduce their size, and an efficient array-based support counting technique called *occurrence-delivery*. The FP-Growth[26] and H-Mine[28] algorithms respectively introduce a prefix-tree structure and a hyper-structure for representing projected-databases to also reduce memory usage.

Table 5 provides a summary of the characteristics of the algorithms discussed in this section. In recent years, a lot of research has been carried on further improving the performance of algorithms for FIM because it is a computationally expensive task. These improvements have been in terms of proposing novel algorithms with additional optimizations,[33–36] and also to design FIM algorithms that can run on GPU processors,[37] on multicore processors,[38] and cloud platforms such as Hadoop[39] and Spark.[40]

VARIATIONS OF THE ITEMSET MINING PROBLEM

Although the task of FIM has numerous applications, it can also be viewed as having limitations in terms of the assumptions that it makes. This section reviews some of the most important limitations of FIM and discusses extensions to the task of itemset mining that are designed to address these limitations.

One the most important limitation of FIM is that an algorithm may find a huge amount of itemsets, depending on how the minimum support threshold is set. Discovering too many patterns makes it difficult for a human to analyze the patterns found. Moreover, itemsets can be viewed as containing a lot of redundancy, because, e.g., if an itemset is frequent, all its subsets are also frequent. To reduce the number of itemsets found and present more meaningful itemsets to the user, researchers have designed algorithms to extract *concise representations* of frequent itemsets. A concise representation is a set of frequent itemsets that is smaller and summarize the whole set of all frequent itemsets. In practice, the size of concise representations can sometimes be several orders of magnitude smaller than the set of frequent itemsets. Moreover, discovering these concise representations is often faster than discovering the full set of frequent itemsets.[29,32,41–47] It was also shown that for task such as classification, using concise representations of items can increase classification accuracy.[48] Let FI denotes the set of frequent itemsets. The most popular concise representations of frequent itemsets are the following.

- *Closed itemsets*[29,32,41–44] are the frequent itemsets that have no superset having the same support, i.e., $CI = \{X | X \in FI \wedge \nexists Y \in FI$ such that $X \subset Y \wedge sup(X) = sup(Y)\}$. In the example of Table 1, of nine frequent itemsets, only four of them are closed: $\{c\}$, $\{e\}$, $\{a, c\}$, and $\{b, c, e\}$. Thus, a considerable reduction in terms of number of itemsets is achieved by discovering only closed itemsets. An interesting property of closed itemsets is that they are a *lossless* representation of all frequent itemsets. In other words, using closed itemsets the information about all frequent itemsets, including their support, can be recovered without scanning the database.[29,32,41–44] In market basket analysis, closed itemsets are also interesting because they represent the largest sets of items common to groups of customers.

TABLE 5 | Summary of the Five Representative FIM Algorithms

Algorithm	Type of Search	Database Representation
Apriori[3]	Breadth-first (candidate generation)	Horizontal
Eclat[27]	Depth-first (candidate generation)	Vertical (TID-lists, diffsets)
FP – Growth[26]	Depth-first (pattern-growth)	Horizontal (prefix-tree)
H – Mine[28]	Depth-first (pattern-growth)	Horizontal (hyperlink structure)
LCM[29]	Depth-first (pattern-growth)	Horizontal (with transaction merging)

- *Maximal itemsets*[29] are the set of frequent itemsets that do not have supersets that are frequent, i.e., $MI = \{X | X \in FI \wedge \nexists X \in FI$ such that $X \subset Y\}$. In other words, maximal itemsets are the largest frequent itemsets. The representation of maximal itemsets is a subset of the representation of closed itemsets ($MI \subseteq CI \subseteq FI$) , and thus can further reduce the number of itemsets presented to the user. However, maximal itemsets are not a lossless representation. Maximal itemsets can be used to recover all frequent itemsets, but they cannot be used to recover the support of all frequent itemsets. In the example of Table 1, there are only two maximal itemsets: {*a*, *c*} and {*b*, *c*, *e*}.
- *Generator itemsets (key itemsets)*[45–47] are the set of frequent itemsets that have no subsets having the same support, i.e., $GI = \{X | X \in FI \wedge \nexists X \in FI$ such that $Y \subset X \wedge sup(X) = sup(Y)\}$. The set of generator itemsets is always of equal size as or larger than the set of closed itemsets and the set of maximal itemsets. But the set of generators is interesting according to the minimum description length principle,[49] as it represents the smallest sets of items that are common to sets of transactions. For example, in market basket analysis, generator itemsets represent the smallest sets of items common to group of customers.

Another popular way of reducing the number of patterns found by itemset mining algorithms is to introduce constraints to filter less interesting patterns. Many different types of more or less complex constraints have been introduced. The naive way of applying constraints in FIM is to apply them as a postprocessing step after all the frequent itemsets have been found. However, this approach may suffer from important performance issues as discovering all patterns can be time-consuming and require a lot of space. A better way of applying constraints is to push them deep in the mining process, i.e., to use the constraints to reduce the search space, and thus improve the performance of the algorithms. Algorithms that use constraints to reduce the search space can sometimes be orders of magnitude faster, because constraints can greatly reduce the size of the search space, and produce orders of magnitude less patterns. Some of the first itemset mining algorithms to integrate constraints are the Reorder and Direct algorithms, which are Apriori-based algorithms that let the user specify boolean constraints on items that can appear in frequent itemsets.[50] Another example of constraint is the use of the *occupancy*,[51] a measure used to find itemsets that occupy a large portion of transactions where they appear. Measures have also been presented to assess how correlated each itemset is.[13–15] For example, the *bond*[13] of an itemset $X = \{i_1, i_2, \ldots i_p\}$ is defined as $bond(X) = |tid(X)|/|tid(i_1) \cap tid(i_2) \cap \ldots tid(i_p)|$. Thus, a frequent itemset having a high bond is not just frequent but also contains items that often cooccur. Many other alternative interestingness measures have also been proposed in the itemset mining literature,[52] such as the affinity,[53] all-confidence,[54] coherence and mean,[14,15] each having different advantages and limitations. The interested reader may refer to the study of Geng and Hamilton[52] for more details.

In terms of constraints that can be used in itemset mining, not all constraints can be pushed deep into the mining process to reduce the search space. Some studies[55–57] have categorized constraints into several categories such as monotone, anti-monotone, succinct, convertible and other constraints. *Anti-monotone constraints* such as the minimum support threshold are some of the easiest and most beneficial to integrate in an FIM mining algorithm, as they can be used to prune the search space by applying the downward closure property. *Convertible constraints* are constraints that are neither monotone nor anti-monotone but that can be converted to anti-monotone constraints if some additional strategies are applied by the FIM algorithm. For example, assume that a weight is assigned to each item. A constraint that is neither anti-monotone nor monotone is the maximum average weight of an itemset. But this constraint can be easily converted into an anti-monotone constraint if items are sorted by increasing order of weights, and itemsets are explored by following that order.[55] A *succinct constraint* is a constraint that can be checked for an itemset by only looking at the single items that it contains. For example, the constraint that the sum of the weights of an itemset should be less than a given value can be checked by simply adding the weights of its items. This constraint is both succint and anti-monotone. For more information about the use of constraints, the reader may refer to the referenced studies.[55–57]

Another limitation of traditional FIM is that it assumes that all items are equal. But in real-life applications, items are often very different from each other.[58] For example, two items such as bread and caviar do not have the same selling frequencies in a retail store, and one should not expect these items to have the same frequencies because bread is a very common type of product whereas caviar is a specialized and expensive product. Thus, some items have naturally more chance of being frequent than others. This leads to the *rare item problem*,[58] which means

that some items are much less likely to appear in frequent itemsets than others. To address this issue, researchers have developed algorithms to find frequent itemsets using multiple minimum support thresholds, such as MSApriori,[58] CFPGrowth,[59] and CFPGrowth++.[60] These algorithms let the user select a different minimum support threshold for each item. As a result, these algorithms can find frequent itemsets containing rare items and/or frequent items. For example, a user could assign a lower minimum support threshold for the item caviar than for the item bread. A related problem with databases having a skewed item support distribution is that patterns involving both frequent and infrequent items may be spurious as infrequent items may appear with frequent items simply because those latter are frequent. Measures have thus been designed to eliminate such patterns, called *cross-support patterns*.[53]

A lot of research has also been carried on finding rare itemsets in databases[12,61,62] instead of frequent itemsets, as frequent itemsets may not always be the most interesting in real-life applications. The problem of discovering rare patterns is quite challenging as there is generally much more rare patterns than frequent patterns. As a result, various definitions of rare patterns have been proposed, each providing different restrictions on what is a rare pattern to avoid discovering a huge number of rare patterns. For example, the AprioriInverse[63] algorithm finds *perfectly rare itemsets*. A perfectly rare itemset is an itemset that has a support no less than a minimum support threshold and not higher than a maximum support threshold. Furthermore, it is required that all subsets of a perfectly rare itemset also have a support not greater than the maximum support threshold. Another type of rare itemsets is the *minimal rare itemsets*.[61,62] A minimal rare itemset is an itemset that is infrequent according to the minimum support threshold and that all its subsets are frequent. Thus, minimal rare itemsets can be viewed as the itemsets that are almost frequent. Both the perfectly rare itemsets and minimal rare itemsets are subsets of the set of all infrequent itemsets.

Another drawback of traditional FIM algorithms is that they are designed to be applied as batch algorithms, i.e., they are designed to be run only once. This is a problem if FIM algorithms are applied in a dynamic environment. For example, if a transaction database is updated, a user will need to apply an FIM algorithm again to get an updated view of the patterns in the database. This is inefficient because sometimes only small changes are made to a database. In these cases, it would be preferable to not recompute all frequent itemsets from scratch. Various algorithms have been designed to provide updated results to a user when a database changes. There are three main types of algorithms as follows:

- *Incremental mining algorithms* are designed to update the set of frequent itemsets when new transactions are inserted, deleted, or modified[64–67] in a transaction database. An example of strategy used by this type of algorithms to avoid recalculating all frequent itemsets from scratch is to keep a buffer of almost-frequent itemsets in memory. Using a buffer, it is unnecessary to perform a database scan to calculate the support of itemsets, when a database is only slightly changed.
- *Stream mining algorithms* are designed to cope with a potentially infinite stream of transactions. They assume that transactions may arrive at a very high speed. Thus, these algorithms are optimized to process transactions as quickly as possible and generally to calculate an approximate set of frequent itemsets rather than an exact set of frequent itemsets.[68] Some popular algorithms for mining frequent itemsets in streams are estDec[69] and estDec+.[70] These algorithms use a tree structure for maintaining information about frequent itemsets, and also include a recency constraint to discover itemsets that are recently frequent, but that may have been infrequent in the past, and calculate upper-bounds on the approximation error of support calculation for frequent itemsets. Similarly, algorithms have also been designed to find and maintain closed[71] and maximal itemsets[72] in streams.
- *Interactive mining algorithms* propose a different solution to the problem of handling dynamic databases. The idea is that instead of mining and updating a large number of itemsets that may not all be useful, one could mine the itemsets that are needed on-the-fly, when they are needed. For example, the Itemset-Tree[73] and the improved Memory-Efficient Itemset-Tree[74] are tree-based data structures that can be created from a transaction database, and that can be updated incrementally with new transactions. These structures are optimized for quickly answering queries about itemsets on-the-fly, such as (1) computing the support of a given itemset, (2) finding all supersets of a given itemset that has a support greater than a given minimum support threshold, and (3) finding all association rules having

a given itemset as antecedent. Interactive algorithms can be very efficient. For example, it was reported that the above structure can process more than 10,000 queries in just a few seconds for some datasets, on a standard personal computers.[74]

Another important limitation of traditional FIM algorithms is the database format. As previously explained, FIM assumes that the input database only contains binary attributes (items). But in real-life applications, this assumption does not always hold. Thus, several extensions of FIM have been proposed to handle richer database types. Some of the most important ones are the following:

- *Weighted itemset mining* is an extension of FIM where weights are associated to each item to indicate their relative importance.[75–77] The goal of weighted itemset mining is to find itemsets that have a minimum weight. A popular variation of this problem is to mine infrequent weighted itemsets.[78]
- *High-utility itemset mining* (HUIM) is an extension of weighted itemset mining where not only weights are considered but also purchase quantities in transactions.[19–25] In traditional FIM, purchase quantities are assumed to be binary, i.e., either an itemset appears in a transaction or not. In HUIM, the number of units bought for each item is indicated in transactions. For example, a transaction could indicate that a customer has bought four breads and two bottles of wine, which is different than a customer having bought one bread and four bottles of wine. In HUIM, weights can be viewed as the unit profit of items (how much profit is generated by each unit sold of a product). The goal of HUIM is to find all itemsets that have a *utility* higher than a given threshold in a database (i.e., itemsets generating a high profit). A major challenge in HUIM is that the utility measure is neither monotone nor anti-monotone.[19] Hence, the utility measure cannot be directly used to prune the search space. To solve this problem, HUIM algorithms have introduced the concept of upper-bound. For example, the Two-Phase algorithm[19] uses an upper-bound, called the TWU, on the utility of itemsets that is monotone to reduce the search space. A major challenge in HUIM has been to develop tighter upper-bounds on the utility to be able to prune a larger part of the search space, and improve the performance of HUIM algorithms.[23–25,66] One of the current fastest HUIM algorithm is EFIM.[24] Various extensions of the utility-mining problem have also been proposed to consider for example the shelf-time periods of items,[79] discount strategies applied in retail stores,[80] and also to discover the top-k most profitable itemsets.[81,82]
- *Uncertain itemset mining* is another popular extension of FIM, designed to consider uncertainty about the data.[83–87] Uncertainty play a role in several real-life applications because data collected is often imperfect, inaccurate, or may be collected through noisy sensors. Two main models have been proposed for uncertain FIM.[85] The first model is the *expected-support model*.[84,86,87] It considers that each item i appearing in a transaction T_q is associated to an expected support value $e(i, T_q)$ representing the certainty that this item appeared in the transaction (a value in the [0,1] interval). For example, consider a transaction database where items represent symptoms and transactions represent hospital patients. A symptom such as stomach pain may be associated to a patient with an expected-support of 0.75 (representing a probability of 75%) based on medical tests or discussion with the patient. The expected-support of an itemset X in a transaction is defined as the product of the expected-support of its items in the transaction, i.e., $exp(X, T_q) = \prod_{i \in X} e(i, Tq)$. The expected-support of an itemset in a database D is the sum of its expected-support in all transactions where X appears, i.e., $exp(X,D) = \sum_{T_q \in D \wedge X \subseteq T_q} exp(X,Tq)$. The task of uncertain itemset mining in the expected support model is to discover all itemsets that are expected to be frequent. The second model is the *probabilistic itemset model*.[84] It utilizes the same database format, but it considers two thresholds: the minimum support threshold *minsup*, as well as a second threshold called the minimum probability threshold *minprob*. An itemset is considered a *probabilistic frequent itemset* if the calculated probability that it appears in more than *minsup* transactions by considering possible worlds is greater than *minprob*.
- *Fuzzy itemset mining*[88–91] is also a well-studied extension of itemset mining. In fuzzy itemset mining, quantitative values are assigned to each item in transactions and fuzzy membership functions are defined for each attribute

(item) to map these values to nominal values. For example, an attribute of a transaction could be the 'height' of a person and a corresponding fuzzy membership function could be defined to map a height to nominal values such as short, average, or tall. Fuzzy FIM algorithms discover itemsets where each item is associated with a nominal value and a membership percentage. For example, a fuzzy FIM algorithm can discover itemsets such as 'height (tall = 80%) and age (young = 60%).' Some of the most important applications of fuzzy itemset mining are in text mining, including text clustering.[90]

Another limitation of FIM is that it is traditionally applied to find itemsets in a single set of transactions. However, in real-life applications, it is often useful to discover patterns that are different or vary greatly in two or more sets of transactions. For example, one may wish to discover patterns that explain the difference between the shopping behavior of adults and teenagers. For this purpose, algorithms have been designed to mine *emerging patterns*. An emerging pattern is a pattern that is significantly more frequent in a set of transactions than in another.[92] It was shown that emerging patterns are useful for tasks such as classification.[93]

OTHER PATTERN MINING PROBLEMS RELATED TO ITEMSET MINING

The previous section has described popular extensions of the problem of FIM to address some of its limitations. This section describes some other important pattern mining problems that are closely related to itemset mining and are solved using similar techniques.

- *Association rule mining*[3,94,95] is the task of discovering rules of the form $X \rightarrow Y$ in a transaction database, where X and Y are itemsets such that $X \cap Y = \emptyset$. The interestingness of a rule is traditionally assessed by the *support* measure $sup(X \rightarrow Y) = sup(X \cup Y)$ and the *confidence* measure $conf(X \rightarrow Y) = sup(X \cup Y)/sup(X)$.[3] The confidence is a very useful measure because it assesses how 'strong' an association between some items is. The confidence represents the conditional probability $P(Y|X)$. Thus, for any rule $X \rightarrow Y$, if follows that $conf(X \rightarrow Y)$ is not necessarily equal to $conf(Y \rightarrow X)$. To discover frequent and confident association rules, a user has to specify a minimum support threshold *minsup* and a minimum confidence threshold *minconf*. An association rule mining algorithm then finds all the valid association rules, i.e., those having a support and confidence respectively no less than these thresholds. For example, for the database of Figure 1 and *minsup* = 3 and *minconf* = 1 (which means 100% confidence), the valid association rules are $\{a\} \rightarrow \{c\}$, $\{e\} \rightarrow \{b\}$, $\{b\} \rightarrow \{e\}$, $\{c, e\} \rightarrow \{b\}$, and $\{b, c\} \rightarrow \{e\}$, which all have a confidence of 100%, and respectively have a support of 3, 4, 4, 3, and 3 transactions. Using the confidence measure, association rule mining algorithms can discover patterns representing strong associations between items. This addresses a limitation of traditional FIM, which is that it may find many frequent itemsets that are weakly correlated. For example, $\{a, c\}$ is a frequent itemset for *minsup* = 3. But the rule $\{a\} \rightarrow \{c\}$ has a higher confidence (100%) than the rule $\{c\} \rightarrow \{a\}$ (75%). Thus, association rules can provide a more detailed information about this itemset. Note that more than 20 alternative interestingness measures have been proposed for association mining beside the support and confidence. The interested reader may refer to Lenca et al.[95] for a survey of association rule measures. Association rules are typically generated in two steps by first discovering frequent itemsets using a standard FIM algorithm and then generating the rules using the frequent itemsets.[3]
- *Sequential pattern mining*[16,17,96–99] consists of discovering sequences frequently appearing in a set of sequences. The problem is similar to the problem of FIM except that the input database is a sequence database, where each sequence is a sequence of transactions. The output of a sequential pattern mining algorithm is a set of *frequent sequential patterns*, i.e., subsequences that appear in no less than *minsup* sequences in the database, where *minsup* is the minimum threshold specified by the user. Applications of sequential pattern mining include analyzing the genome, analyzing web-click stream, and analyzing alarm data in telecommunications.[96] A variation of the sequential pattern mining problem is to discover *sequential rules*[100,101] of the form $X \rightarrow Y$ indicating that if some items X appear in a sequence it will be followed by some other items Y with a given confidence. A sequential rule $X \rightarrow Y$ can be viewed as an association rule that respect the restriction that X must appear before Y. Sequential rules have

been reported as more effective than sequential patterns for some tasks involving prediction.[100] They have been applied to various applications such as e-learning, web page prefetching, anti-pattern detection, alarm sequence analysis, and restaurant recommendation.[101] Another variation is to discover *frequent partial orders*[102,103] rather than sequential patterns or rules. Items in a frequent partial orders are only partially ordered sequentially. Thus, a frequent partial order can summarize several sequential patterns.[102]

- *Episode mining*[104,105] is similar to the problem of sequential pattern mining except that patterns are mined in a single sequence rather than in a set of sequences. The goal is to discover itemsets or *episode rules* of the form $X \rightarrow Y$ that appears many times in a sequence of transactions. An itemset or rule occurs in a sequence if it appears within a time window set by the user. Episode mining can be used to analyze various types of data such as web-click streams, telecommunication data, sensor readings, sequences of events on an assembly line, and network traffic data.[104,105]
- *Periodic pattern mining*[106–108] is the problem of discovering patterns in a single sequence of transactions. The goal of periodic pattern is not just to find patterns that regularly appear in a sequence, but that also appear periodically. The time elapsed or number of transactions between two occurrences of an itemset is called the period length. A user typically has to set parameters on the maximum, minimum or average period lengths to discover periodic patterns.[107] For example, in the database of Table 1, the itemset $\{a, c\}$ could be considered as a periodic pattern, since it appears periodically every two transactions. Applications of periodic pattern mining include stock market analysis, market analysis, and bioinformatics.[106]
- *Subgraph mining*[109–111] is another problem with many similarities to the problem of FIM. The difference is that the goal is to discover *frequent subgraphs* in a database of graphs rather than frequent itemsets in transactions. The traditional problem of subgraph mining requires the user to set a minimum support threshold as in FIM. Some of the key challenges in subgraph mining is that the search space is generally very large and that it is necessary to design strategies to check if different generated graphs are isomorphic (if generated graphs contain the same number of vertices connected in the same manner).[109] As for FIM, several extensions of the problem of subgraph mining have also been proposed, e.g., to mine closed and maximal frequent subgraphs.[110] Applications of subgraph mining are varied and include for example the analysis of chemical compounds.[109,110] *Formal concept analysis* (FCA)[112] is a problem that consists of extracting *formal concepts* from a table of objects (transactions) described using binary attributes (items). A formal concept is the equivalent of a closed itemset in FIM, obtained using *minsup* = 0, and annotated with the set of objects (transactions) where it appears. An interesting aspect of FCA is that it organizes formal concepts as a lattice based on the subset relation, which can be visualized. For example, Figure 3 shows the lattice of formal concepts extracted from the database of Table 1. FCA has been extensively studied and there exist many variations of the FCA problem.[112]

RESEARCH OPPORTUNITIES

Itemset mining and related pattern mining problems have been an active research topic for more than 20 years, and there are still countless opportunities for research in this area. We here provide a classification of the main types of research opportunities in this field as follows:

- *Novel applications*. The first research opportunity is to apply existing pattern mining algorithms in new ways in terms of application

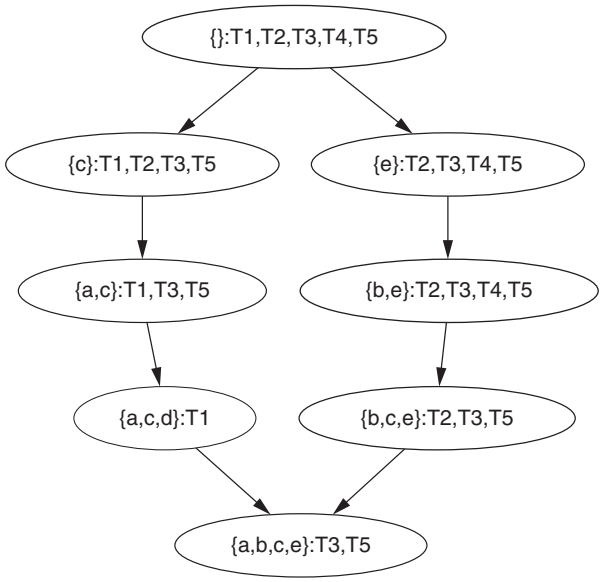

FIGURE 3 | The lattice of formal concepts for the database of Table 1.

domains. As pattern mining algorithms are quite general, they can be applied to 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, and sensor networks provides several novel possibilities in terms of applications.

- *Enhancing the performance of pattern mining algorithms*. As pattern mining can be quite time-consuming, especially on dense databases, large databases, or databases containing many long transactions, a lot of research is carried on developing more efficient algorithms. This is an important problem especially for new extensions of the pattern mining problem, such as uncertain itemset mining or high-utility itemset mining, which have been less explored. Many opportunities also lie in distributed, GPU, multicore, 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 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. Recent researches have for example focused on mining spatial patterns.[113]
- *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 interestingness measures,[95] because it is also key to ensure that the most interesting or useful patterns are found.

OPEN-SOURCE SOFTWARE

An issue in the frequent itemset and pattern mining research community is that most researchers do not release the source code of their algorithms, or even their binary files. This has led to the following problems. First, several researchers need to reimplement algorithms proposed by other researchers to compare their algorithm performance thus spending a great amount of time reimplementing algorithms. Second, in these cases, it remains unsure if reimplemented algorithms are as efficient as the original implementations. Besides, even when binary files are released it has been noted in studies such as the one of Goethal[114] that results may vary greatly depending on the compiler used and the machine architecture used for running performance comparison. Third, if algorithm implementations are not released by researchers, fewer people may use them in real applications, as they would need to be reimplemented.

The main solution to this issue is that researchers release their algorithms as open-source implementations, ideally in a common programming language to facilitate comparison. Currently, the largest collection of open-source implementations of algorithms for pattern mining is by far the *SPMF data mining library*[115] (http://www.philippe-fournier-viger.com/spmf/). It offers more than 110 algorithms for mining patterns such as itemsets, sequential patterns, sequential rules, periodic patterns, and association rules. It is a multiplatform 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. The other main collection of open-source implementation is the FIMI 2004 competition repository website (http://fimi.ua.ac.be/), which provides about 20 C++ implementations of itemset mining algorithms published before 2005. Datasets used for benchmarking itemset and pattern mining algorithms can be found on the SPMF and FIMI websites.

Itemset mining algorithms are also offered as part of some general purpose open-source data mining platform or libraries, such as Weka (http://www.cs.waikato.ac.nz/ml/weka/), Mahout (http://mahout.apache.org/), and Knime (http://www.knime.org/), implemented in Java, and R (https://www.r-project.org/). These softwares offer a wide range of data mining techniques. However, they offer a very limited set of algorithms for itemset mining, offering mostly implementations of Apriori, FP-Growth, and association rule generation. Some specialized platforms offering itemset mining algorithms are Coron (http://coron.loria.fr/) and LUCS-KDD (http://cgi.csc.liv.ac.uk/~frans/KDD/Software/), developed in Java, and Illimine (http://illimine.cs.uiuc.edu/), developed in C++. However, the source code of Coron is not public, Illimine provides the source code of only one of its pattern mining algorithms and LUCS-KDD source code cannot be used for commercial purposes.

Related to the importance of sharing implementations of algorithms in the research community, another issue is that researchers in the field of pattern mining do not always compare the performance of new algorithms with the previous best algorithms because of the lack of public implementations or other reasons. To illustrate this problem, consider the task of subgraph mining. Various algorithms have

been proposed for this task. Among them, the GASTON, FFSM, and SPIN algorithms have been shown to outperform GSPAN.[110] Furthermore, SPIN and MARGIN were shown to outperform FFSM.[110] And recently, FPGraphMiner was shown to outperform MARGIN and FFSM.[116] However, to our knowledge no algorithm has been compared to GASTON, even though it one of the first proposed algorithms for frequent subgraph mining. And SPIN, GASTON, and FPGraphMiner have also never been compared with each other. Moreover, can we really make the transitive inference that an algorithm like FPGraphMiner is faster than GSPAN, because it is faster than MARGIN and FFSM? Thus, in this type of cases, it remains uncertain, which algorithm is the most efficient.

CONCLUSIONS

Itemset mining is an active field of research having numerous applications. This study has presented the problem of FIM, discussed the main techniques for exploring the search space of itemsets, employed by itemset mining algorithms. Then, the study has discussed several extensions of the basic FIM problem to overcome some of its limitations, e.g., to handle dynamic databases, uncertain data, and the use of various constraints. The paper has also briefly discussed pattern mining problems that are closely related to the problem of itemset mining such as sequential pattern mining, sequential rule mining, association rule mining, periodic pattern mining, and episode mining. Finally, the study has discussed research opportunities and open-source implementations of pattern mining algorithms.

NOTE

[a] A Hasse diagram draws an arrow from an itemset X to another itemset Y if and only if $X \subseteq Y$ and $|X| + 1 = |Y|$.

ACKNOWLEDGMENT

The research of the corresponding author is supported by the Youth 1000 talent program from the National Science Foundation of China.

REFERENCES

1. Aggarwal CC. *Data Mining: The Textbook*. Heidelberg: Springer; 2015.
2. Han J, Pei J, Kamber M. *Data Mining: Concepts and Techniques*. Amsterdam: Elsevier; 2011.
3. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, Santiago de Chile, Chile, 12–15 September, 1994, 487–499).
4. Naulaerts S, Meysman P, Bittremieux W, Vu TN, Berghe WV, Goethals B, Laukens K. A primer to frequent itemset mining for bioinformatics. *Brief Bioinform* 2015, 16:216–231.
5. Fernando B, Elisa F, Tinne T. Effective use of frequent itemset mining for image classification. In: *European Conference on Computer Vision*, Florence, Italy, 7–13 October, 2012, 214–227.
6. Glatz E, Mavromatidis S, Ager B, Dimitropoulos X. Visualizing big network traffic data using frequent pattern mining and hypergraphs. *Computing* 2014, 96:27–38.
7. Brauckhoff D, Dimitropoulos X, Wagner A, Salamatian K. Anomaly extraction in backbone networks using association rules. *IEEE/ACM Trans Netw* 2012, 20:1788–1799.
8. Mukherjee A, Liu B, Glance N. Spotting fake reviewer groups in consumer reviews. In: *Proceedings of the 21st International Conference on World Wide Web*, Lyon, France, 16–20 April, 2012, 191–200.
9. Liu Y, Zhao Y, Chen L, Pei J, Han J. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. *IEEE Trans Parallel Distrib Syst* 2012, 23:2138–2149.
10. 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: *Proceedings of the 2015 I.E. International Conference on Communications*, London, UK, 8–12 June, 2015, 5691–5696.
11. Mwamikazi E, Fournier-Viger P, Moghrabi C, Baudouin R. A dynamic questionnaire to further reduce questions in learning style assessment. In: *Proceedings of the 10th International Conference on Artificial Intelligence Applications and Innovations*, Rhodes, Greece, 19–21 September, 2014, 224–235.
12. Koh YS, Ravana SR. Unsupervised rare pattern mining: a survey. *ACM Trans Knowl Discov Data* 2016, 10:1–29.
13. Fournier-Viger P, Lin JCW, Dinh T, Le HB. Mining correlated high-utility itemsets using the bond measure.

In: *Proceedings of the International Conference on Hybrid Artificial Intelligence Systems*, Seville, Spain, 18–20 April, 2016, 53–65).

14. Barsky M, Kim S, Weninger T, Han J. Mining flipping correlations from large datasets with taxonomies. *VLDB Endowment* 2011, 5:370–381.
15. Soulet A, Raissi C, Plantevit M, Cremilleux B. Mining dominant patterns in the sky. In: *Proceedings of the 11th IEEE International Conference on Data Mining*, Vancouver, Canada, 11–14 December, 2011, 655–664.
16. Mabroukeh NR, Ezeife CI. A taxonomy of sequential pattern mining algorithms. *ACM Comput Surv* 2010, 43:3.
17. Fournier-Viger P, Gomariz A, Campos M, Thomas R. Fast vertical mining of sequential patterns using co-occurrence information. In: *Proceedings of the 18th Pacific-Asia Conf. Knowledge Discovery and Data Mining*, Tainan, Taiwan, 13–16 May, 2014, 40–52.
18. Yan X, Han J. 2002. gspan: graph-based substructure pattern mining. In: *Proceedings of the 2002 International Conference on Data Mining*, Maebashi City, Japan, 9–12 December, 2002, 721–724.
19. Liu Y, Liao WK, Choudhary AN. A two-phase algorithm for fast discovery of high utility itemsets. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, 18–20 May, 2005, 689–695.
20. Lin CW, Hong TP, Lu WH. An effective tree structure for mining high utility itemsets. *Expert Syst Appl* 2011, 38:7419–7424.
21. Lin YC, Wu CW, Tseng VS. Mining high utility itemsets in big data. In: *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Ho Chi Minh City, Vietnam, 19–22 May, 2015, 649–661.
22. Liu M, Qu. J. Mining high utility itemsets without candidate generation. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, Maui, HI, USA, 29 October–2 November, 2012, 55–64.
23. Fournier-Viger P, Wu CW, Zida S, Tseng VS. FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, Roskilde, Denmark, 25–27 June, 2014, 83–92.
24. Zida S, Fournier-Viger P, Lin JCW, Wu WW, Tseng VS. EFIM: a highly efficient algorithm for high-utility itemset mining. In: *Proceedings of the 14th Mexican International Conference on Artificial Intelligence*, Cuernavaca, Mexico, 25–31 October, 2015, 530–546.
25. Yun U, Ryang H, Ryu KH. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. *Expert Syst Appl* 2014, 41:3861–3878.
26. Han J, Pei J, Ying Y, Mao R. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Min Knowl Discov* 2004, 8:53–87.
27. Zaki MJ. Scalable algorithms for association mining. *IEEE Trans Knowl Data Eng* 2000, 12:372–390.
28. Pei J, Han J, Lu H, Nishio S, Tang S, Yang D, H-mine: hyper-structure mining of frequent patterns in large databases. In: *Proceedings of the 2001 I.E. International Conference on Data Mining*, San Jose, CA, USA, 29 November–2 December, 2001, 441–448.
29. Uno T, Kiyomi M, Arimura H. LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: *Proceedings of the ICDM'04 Workshop on Frequent Itemset Mining Implementations*. Aachen, Germany: CEUR; 2004.
30. Hegland M. The apriori algorithm tutorial. In: Goh SS, Ron A, Shen Z, eds. *Mathematics and Computation in Imaging Science and Information Processing*, vol. 11. Singapore: World Scientific; 2005, 209–262.
31. Zaki MJ, Gouda K. Fast vertical mining using diffsets. In: *Proceedings of the 9th ACM SIGKDD International Conference Knowledge Discovery and Data Mining*, Washington, DC, USA, 24–27 August, 2003, 326–335.
32. Lucchese C, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets. *IEEE Trans Knowl Data Eng* 2006, 18:21–36.
33. Chen J, Xiao K. BISC: a bitmap itemset support counting approach for efficient frequent itemset mining. *ACM Trans Knowl Discov Data* 2010, 4:12.
34. Deng ZH, Lv SL. PrePost+: an efficient N-lists-based algorithm for mining frequent itemsets via children parent equivalence pruning. *Expert Syst Appl* 2015, 42:5424–5423.
35. Pyun G, Yun U, Ryu KH. Efficient frequent pattern mining based on linear prefix tree. *Knowl-Based Syst* 2014, 55:125–139.
36. Vo B, Le T, Coenen F, Hong TP. Mining frequent itemsets using the N-list and subsume concepts. *Int J Mach Learn Cybern* 2016, 7:253–265.
37. Zhang F, Zhang Y, Bakos JD. Accelerating frequent itemset mining on graphics processing units. *J Supercomput* 2013, 66:94–117.
38. Schlegel B, Karnagel T, Kiefer T, Lehner W. Scalable frequent itemset mining on many-core processors. In: *Proceedings of the 9th International Workshop Data Management on New Hardware*, New York, USA, 24 June, 2013, paper 3.
39. Moens S, Aksehirli E, Goethals B. Frequent itemset mining for big data. In: *2013 I.E. International Conference on Big Data*, Santa Clara, CA, USA, 6–9 October, 2013, 111–118.
40. Qiu H, Gu R, Yuan C, Huang Y. Yafim: a parallel frequent itemset mining algorithm with spark. In: *Proceedings of the 2014 I.E. International Parallel and*

Distributed Processing Symposium Workshops, Phoenix, AZ, USA, 19–23 May, 2014, 1664–1671.

41. Zaki MJ, Hsiao CJ. CHARM: an efficient algorithm for closed itemset mining. In: *Proceedings of the 12th SIAM International Conference on Data Mining*, Anaheim, CA, USA, 26–28 April, 2012, 457–473.
42. Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules. In: *Proceedings of the International Conference on Database Theory*, Jerusalem, Israel, 10–12 January, 1999, 398–416.
43. Aliberti G, Colantonio A, Di Pietro R, Mariani R. EXPEDITE: EXPress closED ITemset enumeration. *Expert Syst Appl* 2015, 42:3933–3944.
44. Vo B, Hong TP, Le B. DBV-miner: a dynamic bit-vector approach for fast mining frequent closed itemsets. *Expert Syst Appl* 2012, 39:7196–7206.
45. Soulet A, Rioult F. Efficiently depth-first minimal pattern mining. In: *Proceedings of the 18th Pacific-Asia Conf. Knowledge Discovery and Data Mining*, Tainan, Taiwan, 13–16 May, 2014, 28–39.
46. Fournier-Viger P, Wu CW, Tseng VS. Novel concise representations of high utility itemsets using generator patterns. In: *Proceedings of the International Conference on Advanced Data Mining and Applications*, Guilin, China, 19–21 December, 2014, 30–43.
47. Szathmary L, Valtchev P, Napoli A, Godin R, Boc A, Makarenkov V. A fast compound algorithm for mining generators, closed itemsets, and computing links between equivalence classes. *Ann Math Artif Intell* 2014, 70:81–105.
48. Antonie L, Zaiane OR, Holte RC. Redundancy reduction: does it help associative classifiers? In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, Pisa, Italy, 4–8 April, 2016, 867–874
49. Barron A, Rissanen J, Yu B. The minimum description length principle in coding and modeling. *IEEE Trans Inf Theory* 1998, 44:2743–2760.
50. Srikant R, Vu Q, Agrawal R. Mining association rules with item constraints. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, Newport Beach, CA, USA, 14–17 August, 1997, 67–73.
51. Tang L, Zhang L, Luo P, Wang M. Incorporating occupancy into frequent pattern mining for high quality pattern recommendation. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, Maui, HI, USA, 29 October–2 November, 2012, 75–84.
52. Geng L, Hamilton HJ. Interestingness measures for data mining: a survey. *ACM Comput Surv* 2006, 38:9.
53. Xiong H, Tan PN, Kumar V. Mining strong affinity association patterns in data sets with skewed support distribution. In: *Proceedings of the 2003 I.E. International Conference on Data Mining*. Melbourne, FL, USA, 19–22 December, 2003, 387–394.
54. Omiecinski E. Alternative interest measures for mining associations in databases. *IEEE Trans Knowl Discov Data Eng* 2003, 15:57–69.
55. Pei J, Han J, Lakshmanan LV. Mining frequent itemsets with convertible constraints. In: *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, 2–6 April, 2001, 433–442.
56. Pei J, Han J, Lakshmanan LV. Pushing convertible constraints in frequent itemset mining. *Data Min Knowl Discov* 2004, 8:227–252.
57. Bonchi F, Lucchese C. Pushing tougher constraints in frequent pattern mining. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, 18–20 May, 2005, 114–124.
58. Liu B, Hsu W, Ma Y. Mining association rules with multiple minimum supports. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, USA, 15–18 August, 1999, 337–341.
59. Hu YH, Chen YL. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decis Support Syst* 2006, 42:1–24.
60. Kiran RU, Reddy PK. Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In: *Proceedings of the 14th International Conference on Extending Database Technology*, Uppsala, Sweden, 21–24 March, 2011, 11–20.
61. Szathmary L, Napoli A, Valtchev P. Towards rare itemset mining. In: *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, Patras, Greece, 29–31 October, 2007, 305–312.
62. Szathmary L, Valtchev P, Napoli A, Godin R. Efficient vertical mining of minimal rare itemsets. In: *Proceedings of the 9th International Conference on Concept Lattices and Their Applications*, Fuengirola, Spain, 11–14 October, 2012, 269–280.
63. Koh YS, Rountree N. Finding Sporadic Rules Using Apriori-Inverse. In: *Proceedings of the 9th Pacific-Asia Conference, PAKDD 2005*, Hanoi, Vietnam, 18–20 May, 2005, 97–106.
64. Koh JL, Shieh SF. An efficient approach for maintaining association rules based on adjusting FP-tree structures. In: *Proceedings of the 9th International Conference on Database Systems for Advanced Applications*, Jeju Island, Korea, 17–19 March, 2004, 417–424.
65. Leung CK, Khan QI, Li Z, Hoque T. CanTree: a canonical-order tree for incremental frequent-pattern mining. *Knowl Inf Syst* 2007, 11:287–311.

66. Lin CW, Hong TP, Lu WH. The Pre-FUFP algorithm for incremental mining. *Expert Syst Appl* 2009, 36:9498–505.

67. Nath B, Bhattacharyya DK, Ghosh A. Incremental association rule mining: a survey. *Wiley Interdiscip Rev Data Min Knowl Discov* 2013, 3:157–169.

68. Calders T, Dexters N, Gillis JJ, Goethals B. Mining frequent itemsets in a stream. *Inf Syst* 2014, 39:233–55.

69. Chang JH, Lee WS. Finding recent frequent itemsets adaptively over online data streams. In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, 24–27 August, 2003, 487–492.

70. Shin SJ, Lee DS, Lee WS. CP-tree: an adaptive synopsis structure for compressing frequent itemsets over online data streams. *Inf Sci* 2014, 10:559–576.

71. Chi Y, Wang H, Philip SY, Muntz RR. Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowl Inf Syst* 2006, 10:265–294.

72. Farzanyar Z, Kangavari M, Cercone N. Max-FISM: mining (recently) maximal frequent itemsets over data streams using the sliding window model. *Comput Math Appl* 2012, 64:1706–18.

73. Kubat M, Hafez A, Vijay Raghavan V, Lekkala JR, Chen WK. Itemset trees for targeted association querying. *IEEE Trans Knowl Data Eng* 2003, 15:1522–1534.

74. Fournier-Viger P, Mwamikazi E, Gueniche T, Faghihi U. Memory efficient itemset tree for targeted association rule mining. In: *Proceedings of the 9th International Conference on Advanced Data Mining and Applications*, Hangzhou, China, 14–16 December, 2013, 95–106.

75. Torres-VerdÃn C, Chiu KY, Vasudeva Murthy AS. WFIM: weighted frequent itemset mining with a weight range and a minimum weight. In: *Proceedings of the 2005 SIAM International Conference on Data Mining*, Newport Beach, CA, USA, 21–23 April, 2005, 636–640.

76. Yun U. Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. *Inform Sci* 2007, 177:3477–3499.

77. Yun U. On pushing weight constraints deeply into frequent itemset mining. *Intell Data Anal* 2009, 13:359–383.

78. Cagliero L, Garza P. Infrequent weighted itemset mining using frequent pattern growth. *IEEE Trans Knowl Data Eng* 2014, 26:903–915.

79. Fournier-Viger P, Zida S. FOSHU: faster on-shelf high utility itemset mining with or without negative unit profit. In: *Proceedings of the 30th Symposium on Applied Computing*. Salamanca, Spain, 13–17 April, 2015, 857–864.

80. Lin JC, Gan W, Fournier-Viger P, Hong TP, Tseng VS. Fast algorithms for mining high-utility itemsets with various discount strategies. *Adv Eng Inf* 2016, 30:109–126.

81. Tseng V, Wu C, Fournier-Viger P, Yu PS. Efficient algorithms for mining top-K high utility itemsets. *IEEE Trans Knowl Data Eng* 2016, 28:54–67.

82. Duong QH, Liao B, Fournier-Viger P, Dam TL. An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. *Knowl-Based Syst* 2016, 104:106–122.

83. Bernecker T, Kriegel HP, Renz M, Verhein F, Zuefle A. Probabilistic frequent itemset mining in uncertain databases. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, June 28–July 1, 2009, 119–128.

84. Chui CK, Kao B, Hung E. Mining frequent itemsets from uncertain data. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Nanjing, China, 22–25 May, 2007, 47–58.

85. Tong Y, Chen L, Cheng Y, Yu PS. Mining frequent itemsets over uncertain databases. *VLDB Endowment* 2012, 5:1650–1661.

86. Leung CKS, MacKinnon RK. BLIMP: a compact tree structure for uncertain frequent pattern mining. In: *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery*, Munich, Germany, 2–4 September, 2014, 115–123.

87. Lin JCW, Gan W, Fournier-Viger P, Hong TP, Tseng VS. Weighted frequent itemset mining over uncertain databases. *Appl Intell* 2015, 44:232–250.

88. Chen CH, Li AF, Lee YC. Actionable high-coherent-utility fuzzy itemset mining. *Soft Comput* 2014, 18:2413–2424.

89. Hong TP, Kuo CS, Wang SL. A fuzzy AprioriTid mining algorithm with reduced computational time. *Appl Soft Comput* 2004, 5:1–10.

90. Chen CL, Tseng FS, Liang T. Mining fuzzy frequent itemsets for hierarchical document clustering. *Inf Process Manage* 2010, 46:193–211.

91. Lin JCW, Tin L, Fournier-Viger P, Hong TP. A fast algorithm for mining fuzzy frequent itemsets. *J Intell Fuzzy Syst* 2015, 9:2373–2379.

92. Dong G, Li J. Efficient mining of emerging patterns: discovering trends and differences. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, USA, 15–18 August, 1999, 43–52.

93. García-Borroto M, Martinez-Trinidad JF, Carrasco-Ochoa JA. A survey of emerging patterns for supervised classification. *Artif Intell Rev* 2014, 42:705–721.

94. Fournier-Viger P, Wu CW, Tseng VS. Mining top-K association rules. In: *Proceedings of the 25th Canadian*

Conference on Artificial Intelligence, Toronto, Canada, 28–30 May, 2012, 61–73.

95. Lenca P, Vaillant B, Meyer P, Lallich S. Association rule interestingness measures: experimental and theoretical studies. In: *Proceedings of the Quality Measures in Data Mining Workshop*, 2007, 51–76.
96. Mooney CH, Roddick JF. Sequential pattern mining–approaches and algorithms. *ACM Comput Surv* 2013, 45:1–19.
97. Ayres J, Flannick J, Gehrke J, Yiu T. Sequential pattern mining using a bitmap representation. In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, 23–26 July, 2002, 429–435.
98. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu MC. Mining sequential patterns by pattern-growth: the prefixspan approach. *IEEE Trans Knowl Data Eng* 2004, 16:1424–1440.
99. Gouda K, Hassaan M, Zaki MJ. Prism: an effective approach for frequent sequence mining via prime-block encoding. *J Comput Syst Sci* 2010, 76:88–102.
100. Fournier-Viger P, Wu CW, Tseng VS, Cao L, Nkambou R. Mining partially-ordered sequential rules common to multiple sequences. *IEEE Trans Knowl Data Eng* 2015, 27:2203–2216.
101. Fournier-Viger P, Gueniche T, Zida S, Tseng VS. ERMiner: sequential rule mining using equivalence classes. In: *Proceedings of the 13th International Symposium on Intelligent Data Analysis*, Leuven, Belgium, 30 October 30–1 November, 2014, 108–119.
102. Fabregue M, Braud A, Bringay S, Le Ber F, Teisseire M. Mining closed partially ordered patterns, a new optimized algorithm. *Knowl-Based Syst* 2015, 79:68–79.
103. Pei J, Wang H, Liu J, Wang K, Wang J, Yu PS. Discovering frequent closed partial orders from strings. *IEEE Trans Knowl Data Eng* 2006, 18:1467–1481.
104. Mannila H, Toivonen H, Verkamo AI. Discovery of frequent episodes in event sequences. *Data Min Knowl Discov* 1997, 1:259–89.
105. Zimmermann A. Understanding episode mining techniques: benchmarking on diverse, realistic, artificial data. *Intell Data Anal* 2014, 18:761–91.
106. Fournier-Viger P, Lin CW, Duong QH, Dam TL. PHM: mining periodic high-utility itemsets. In: *Proceedings of the 16th Industrial Conference on Data Mining*, New York, USA, 13–17 July, 2016, 64–79.
107. Tanbeer SK, Ahmed CF, Jeong BS, Lee YK. Discovering periodic-frequent patterns in transactional databases. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Bangkok, Thailand, 27–30 April, 2009, 242–253.
108. Kiran U, Venkatesh JN, Fournier-Viger P, Toyoda M, Reddy PK, Kitsuregawa M. Discovering periodic patterns in non-uniform temporal databases. In: *Proceedings of the 21st Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Jeju, Korea, 23–27 May, 2017.
109. Yan X, Han J. gSpan: graph-based substructure pattern mining. In: *Proceedings of the 2002 I.E. International Conference on Data Mining*, Maebashi City, Japan, 9–12 December, 2002, 721–724.
110. Jiang C, Coenen F, Zito M. A survey of frequent subgraph mining algorithms. *Knowl Eng Rev* 2013, 28:75–105.
111. Bhuiyan MA, Al HM. An iterative MapReduce based frequent subgraph mining algorithm. *IEEE Trans Knowl Data Eng* 2015, 27:608–20.
112. Poelmans J, Kuznetsov SE, Ignatov DI, Dedene G. Formal concept analysis in knowledge processing: a survey on models and techniques. *Expert Syst Appl* 2013, 40:6601–6623.
113. Sengstock C, Gertz M. Spatial itemset mining: a framework to explore itemsets in geographic space. In: *Proceedings of the East European Conference on Advances in Databases and Information Systems*, Genoa, Italy, 1–4 September, 2013, 148–161.
114. Goethals B. *Survey on Frequent Pattern Mining*. Helsinki, Finland: University of Helsinki; 2003.
115. Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu CW, Tseng VS. SPMF: a Java open-source pattern mining library. *J Mach Learn Res* 2014, 15:3389–3393.
116. Vijayalakshmi R, Nadarajan R, Roddick JF, Thilaga M, Nirmala P. FP-GraphMiner—a fast frequent pattern mining algorithm for network graphs. *J Graph Algorithms Appl* 2011, 15:753–776.

