Overview

Frequent item set mining

Christian Borgelt*

Frequent item set mining is one of the best known and most popular data mining methods. Originally developed for market basket analysis, it is used nowadays for almost any task that requires discovering regularities between (nominal) variables. This paper provides an overview of the foundations of frequent item set mining, starting from a definition of the basic notions and the core task. It continues by discussing how the search space is structured to avoid redundant search, how it is pruned with the *a priori* property, and how the output is reduced by confining it to closed or maximal item sets or generators. In addition, it reviews some of the most important algorithmic techniques and data structures that were developed to make the search for frequent item sets as efficient as possible.

How to cite this article:
WIREs Data Mining Knowl Discov 2012, 2: 437–456 doi: 10.1002/widm.1074

INTRODUCTION

It is hardly an exaggeration to say that the popular research area of *data mining* was started by the tasks of frequent item set mining and association rule induction. At the very least, these tasks have a strong and long-standing tradition in data mining and *knowledge discovery in databases*, and triggered an abundance of publications in data mining conferences and journals. The huge research efforts devoted to these tasks had considerable impact and led to a variety of sophisticated and efficient algorithms to find frequent item sets. Among the best-known methods are Apriori,[1,2] Eclat,[3–5] FP-Growth (Frequent Pattern Growth),[6–9] and LCM (Linear time Closed item set Miner)[10–12]; but there is also an abundance of alternatives. A curious historical aspect is that researchers in the area of neurobiology came very close to frequent item set mining as early as 1978 with the accretion algorithm,[13] thus preceding Apriori by as much 15 years.

This paper surveys some of the most important ideas, algorithmic concepts, and data structures in this area. The material is structured as follows: *Basic Notions* introduces the basic notions such as *item base*, *transaction*, and *support*; formally defines the frequent item set mining problem; shows how the search space can be structured to avoid redundant search; and reviews how the output can be reduced by confining it to *closed* or *maximal item sets* or *generators*. *Item Set Enumeration* derives the general top-down search scheme for item set enumeration from the fundamental properties of the support measure, resulting in *breadth-first* and *depth-first search*, with the subproblem and item order providing further distinctions. *Database Representations* reviews different data structures by which the initial as well as conditional transaction databases can be represented and how these are processed in the search. *Advanced Techniques* collects several advanced techniques that have been developed to make the search maximally efficient, including perfect extension pruning, conditional item reordering, the *k*-items machine, and special output schemes. *Intersecting Transactions* briefly surveys intersecting transactions as an alternative to item set enumeration for finding closed (and maximal) item sets, which can be preferable in the presence of (very) many items. *Extensions* discusses selected extensions of the basic approaches, such as *association rule induction*, alternatives to item set support, association rule and item set ranking and filtering methods, and *fault-tolerant item sets*. Finally, *Summary* summarizes this survey.

*Correspondence to: christian@borgelt.net
European Centre for Soft Computing, Edificio de Investigación, Calle Gonzalo Gutiérrez Quirós s/n, Mieres, Asturias, Spain. Email: christian@borgelt.net, christian.borgelt@softcomputing.es; WWW: http://www.borgelt.net/, http://www.softcomputing.es/

DOI: 10.1002/widm.1074

BASIC NOTIONS

Frequent item set mining is a data analysis method that was originally developed for market basket

analysis. It aims at finding regularities in the shopping behavior of the customers of supermarkets, mail-order companies, and online shops. In particular, it tries to identify sets of products that are frequently bought together. Once identified, such sets of associated products may be used to optimize the organization of the offered products on the shelves of a supermarket or the pages of a mail-order catalog or Web shop, or may give hints which products may conveniently be bundled. However, frequent item set mining may be used for a much wider variety of tasks, which share that one is interested in finding regularities between (nominal) variables in a given data set.

Problem Definition

Formally, frequent item set mining is the following task: we are given a set $B = \{i_1, \ldots, i_n\}$ of *items*, called the *item base*, and a database $T = (t_1, \ldots, t_m)$ of *transactions*. An item may, for example, represent a product. In this case, the item base represents the set of all products offered by a supermarket. The term *item set* refers to any subset of the item base B. Each transaction is an item set and may represent, in the supermarket setting, a set of products that has been bought by a customer. As several customers may have bought the same set of products, the total of all transactions must be represented as a vector (as above) or as a multiset. Alternatively, each transaction may be enhanced by a *transaction identifier (tid)*. Note that the item base B is usually not given explicitly, but only implicitly as the union of all transactions, that is, $B = \cup_{k \in \{1,\ldots,m\}} t_k$.

The *cover* $K_T(I) = \{k \in \{1, \ldots, m\} \mid I \subseteq t_k\}$ of an item set $I \subseteq B$ indicates the transactions it is contained in. The *support* $s_T(I)$ of I is the number of these transactions and hence $s_T(I) = |K_T(I)|$. Given a user-specified *minimum support* $s_{\min} \in \mathbb{N}$, an item set I is called *frequent* (in T) iff $s_T(I) \geq s_{\min}$. The goal of frequent item set mining is to find all item sets $I \subseteq B$ that are frequent in the database T and thus, in the supermarket setting, to identify all sets of products that are frequently bought together. Note that frequent item set mining may be defined equivalently based on the *(relative) frequency* $\sigma_T(I) = s_T(I)/m$ of an item set I and a corresponding lower bound $\sigma_{\min}$.

As an illustration, Figure 1 shows a simple transaction database with 10 transactions over the item base $B = \{a, b, c, d, e\}$. With a minimum support of $s_{\min} = 3$, a total of 16 frequent item sets can be found in this database, which are shown, together with their support values, in the table on the right in Figure 1. Note that the empty set is often discarded (not reported) because it is trivially contained in all transactions and thus not informative.

(a) Transactions

0: {a, d, e}
1: {b, c, d}
2: {a, c, e}
3: {a, c, d, e}
4: {a, e}
5: {a, c, d}
6: {b, c}
7: {a, c, d, e}
8: {b, c, e}
9: {a, d, e}

(b) Frequent item sets (with support) (minimum support: $s_{\min} = 3$)

0 items	1 item	2 items	3 items
∅: 10	{a}: 7	{a, c}: 4	{a, c, d}: 3
	{b}: 3	{a, d}: 5	{a, c, e}: 3
	{c}: 7	{a, e}: 6	{a, d, e}: 4
	{d}: 6	{b, c}: 3	
	{e}: 7	{c, d}: 4	
		{c, e}: 4	
		{d, e}: 4	

FIGURE 1 | (a) A simple example database with 10 transactions (market baskets, shopping carts) over the item base $B = \{a, b, c, d, e\}$ and (b) the frequent item sets that can be found in it if the minimum support is chosen to be $s_{\min} = 3$ (the numbers state the support of these item sets).

Search Space and Support Properties

It is immediately clear that simply generating every candidate item set in the power set 2^B, determining its support, and filtering out the infrequent sets is computationally infeasible, because even small supermarkets usually offer thousands of different products. To make the search efficient, one exploits a fairly obvious property of item set support, namely that it is *antimonotone*: $\forall I \subseteq J \subseteq B$: $s_T(I) \geq s_T(J)$, regardless of the transaction database T. In other words, *if an item set is extended* (if another item is added to it), *its support cannot increase*. Together with the user-specified minimum support, we immediately obtain the *Apriori property*[1,2]: $\forall I \subseteq J \subseteq B$: $s_T(I) < s_{\min} \Rightarrow s_T(J) < s_{\min}$, that is, *no superset of an infrequent item set can be frequent*. It follows that the set $\mathcal{F}_T(s_{\min})$ of item sets that are frequent in a database T w.r.t. minimum support $s_{\min}$ is *downward closed*: $\forall I \in \mathcal{F}_T(s_{\min}) : J \subseteq I \Rightarrow J \in \mathcal{F}_T(s_{\min})$.

As a consequence, the search space is naturally structured according to the subset relationships between item sets, which form a *partial order* on 2^B. This partial order can be nicely depicted as a Hasse diagram, which is essentially a graph, in which each item set $I \subseteq B$ forms a node and there is an edge between the nodes for two sets I, J with $I \subset J$ if $\nexists K : I \subset K \subset J$. As an example, Figure 2(a) shows a Hasse diagram for the partial order of 2^B for $B = \{a, b, c, d, e\}$, the item base underlying Figure 1.

Due to the *Apriori* property (or the fact that the set of frequent item sets is downward closed), the frequent item sets are dense at the top of such a Hasse diagram (see Figure 2b). Thus frequent item sets are naturally found by a top-down search in this structure.

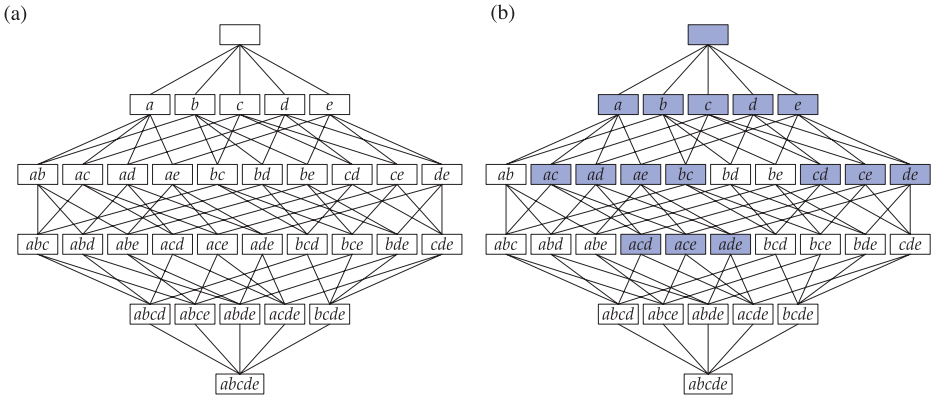

FIGURE 2 | (a) Hasse diagram for the partial order induced by $\subseteq$ on $2^{\{a,b,c,d,e\}}$ and (b) frequent item sets for the database shown in Figure 1 and $s_{\min} = 3$.

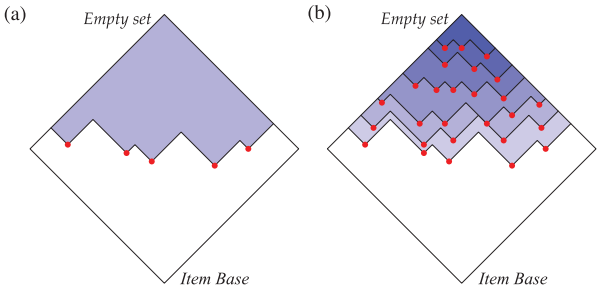

FIGURE 3 | Schematic illustration of maximal (a) and closed item sets (b) demonstrating their relation.

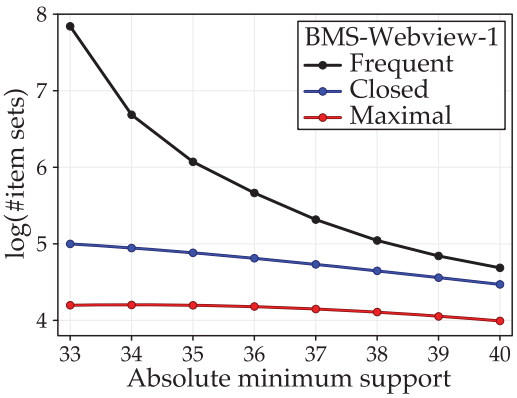

FIGURE 4 | The number of frequent, closed, and maximal item sets on a common benchmark data set (BMS-Webview-1).

Closed and Maximal Item Sets and Generators

An annoying problem in frequent item set mining is that the number of frequent item sets is often huge and thus the output can easily exceed the size of the transaction database to mine. To mitigate this problem, several restrictions of the set of frequent item sets have been suggested. A frequent item set $I \in \mathcal{F}_T(s_{\min})$ is called

- a *maximal (frequent) item set*[14–18] iff $\forall J \supset I$: $s_T(J) < s_{\min}$;
- a *closed (frequent) item set*[19–24] iff $\forall J \supset I$: $s_T(J) < s_T(I)$;
- a *(frequent) generator*[19,25–29] iff $\forall J \subset I$: $s_T(J) < s_T(I)$.

Due to the contraposition of the *Apriori* property, that is, $\forall I \subseteq J \subseteq B : s_T(J) \geq s_{\min} \Rightarrow s_T(I) \geq s_{\min}$, the set of all frequent item sets can easily be recovered from the set $\mathcal{M}_T(s_{\min})$ of maximal item sets as $\mathcal{F}_T(s_{\min}) = \bigcup_{I \in \mathcal{M}_T(s_{\min})} 2^I$. However, for the support of nonmaximal item sets it only preserves a lower bound: $\forall I \in \mathcal{F}_T(s_{\min}) : s_T(I) \geq \max_{J \in \mathcal{M}_T(s_{\min}) \wedge J \supseteq I} s_T(J)$. As an intuitive illustration, the schematic Hasse diagram in Figure 3(a) shows the maximal item sets as red dots, whereas all frequent item sets are shown as a blue region.

The set $\mathcal{C}_T(s_{\min})$ of all closed item sets, however, also preserves knowledge of all support values according to $\forall I \in \mathcal{F}_T(s_{\min}) : s_T(I) = \max_{J \in \mathcal{C}_T(s_{\min}) \wedge J \supseteq I} s_T(J)$, because any frequent item set is either closed or possesses a uniquely determined closed superset. Note that maximal item sets are obviously also closed, but not vice versa. The exact relationship of closed and maximal item sets is: $\mathcal{C}_T(s_{\min}) = \bigcup_{s \in \{s_{\min}, s_{\min}+1, \ldots, m\}} \mathcal{M}_T(s)$, where m is the total number of transactions. This relationship is illustrated schematically in Figure 3(b).

As an illustration of the huge savings that can result from restricting the output to closed or even maximal item sets, Figure 4 shows the number of frequent, closed, and maximal item sets for a common benchmark data set (BMS-Webview-1, see Ref 30; note the logarithmic scale).

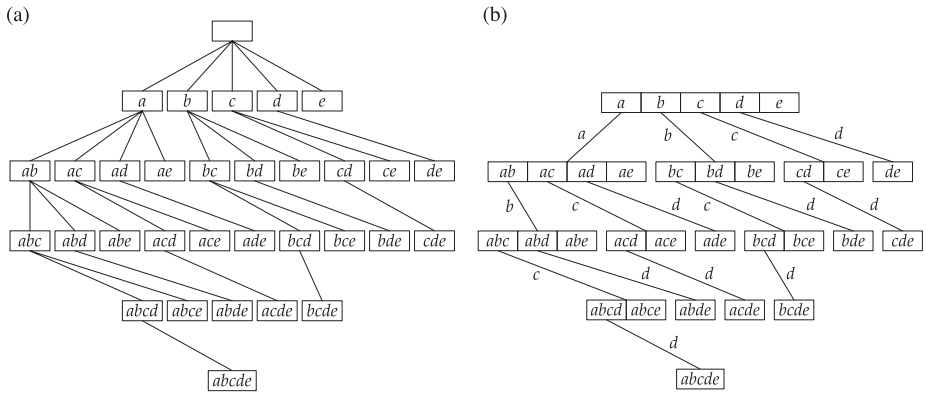

FIGURE 5 | (a) A subset tree that results from assigning a unique parent to each item set and (b) a prefix tree in which sibling nodes with the same prefix are merged.

The set $\mathcal{G}_T(s_{\min})$ of generators (or *free item sets*) has the convenient property that it is downward closed, that is, $\forall I \in \mathcal{G}_T(s_{\min}) : J \subseteq I \Rightarrow J \in \mathcal{G}_T(s_{\min})$. Unfortunately, though, $\mathcal{G}_T(s_{\min})$ does not preserve knowledge which item sets are frequent. However, because every generator possesses (like any frequent item set) a uniquely determined closed superset with the same support, one may report with each generator the difference to this closed superset. In this way, one preserves the same knowledge as with the set of closed item sets. Note, however, that $|\mathcal{G}_T(s_{\min})| \geq |\mathcal{C}_T(s_{\min})|$, although recovering the support values from a generator-based output may be somewhat easier or more efficient than recovering them from the set of closed item sets. Note also that generators are often induced alone (i.e., without reporting the difference to their closed supersets), because they are seen as more useful features for classification purposes, as they contain fewer items than closed sets.

Alternative lossless compressions of the set of frequent item sets, which can reduce the output even more, are, for example, *nonderivable item sets*[31] and *closed nonderivable item sets*.[32] However, the better the compression, the more complex it usually becomes to derive the support values of item sets that are not directly reported.

ITEM SET ENUMERATION

Due to the *Apriori* property, most frequent item set mining algorithms search top-down in (the Hasse diagram representing) the partial order $\subseteq$ on 2^B, thus enumerating the (frequent) item sets. A notable alternative (intersecting transactions) is discussed in *Intersecting Transactions*.

Organizing the Search

Searching the partial order $\subseteq$ on 2^B top-down means growing item sets from the empty set or from single items toward the item base B. However, a naive implementation of this scheme leads to redundant search because the same item set can be constructed multiple times by adding its items in different orders. To eliminate this redundancy, the Hasse diagram representing the partial order is reduced to a tree by assigning a unique parent set $\pi(I)$ to every item set $I \subseteq B$. This is achieved by choosing an arbitrary, but fixed order of the items and setting $\pi(I) = I - \{\max(I)\}$. As an example, Figure 5(a) shows the resulting item subset tree for $B = \{a, b, c, d, e\}$ and the item order $a < b < c < d < e$. Clearly, this tree is a prefix tree because sibling sets differ only in their last item. Thus it is often convenient to combine these siblings into one node as shown in Figure 5(b). This structure is used in the following to explain and illustrate the different search schemes.

Breadth-First/Levelwise Search

The most common ways to traverse the nodes of a tree are breadth-first and depth-first. The former is used in the Apriori algorithm,[1,2] which derives its name from the *Apriori* property. It is most conveniently implemented with a data structure representing a prefix tree such as the one shown in Figure 5(a). This tree is built level by level. A new level is added by creating a child node for every frequent item set on the current level. From these child nodes all item sets are deleted that possess an infrequent subset (*a priori* pruning). Then the transaction database is accessed to count the support of the remaining candidate item sets. This is usually done by traversing the transactions and constructing, with a doubly recursive procedure, all

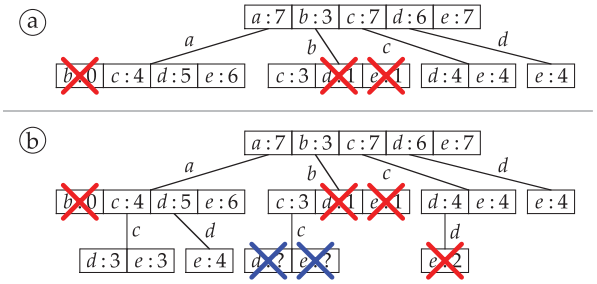

FIGURE 6 | Two steps of the Apriori algorithm: (a) adding the second level; (b) adding the third level (blue: *a priori* pruning, red: *a posteriori* pruning for $s_{\min} = 3$).

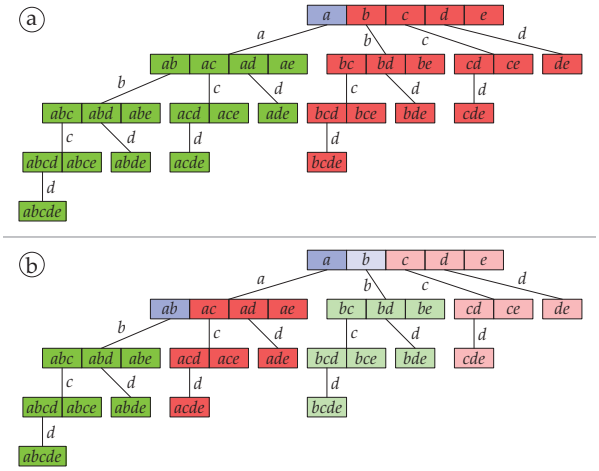

FIGURE 7 | Illustration of the divide-and-conquer approach to find frequent item sets: (a) first split, (b) second split; blue: split item/prefix, green: first subproblem (include split item), red: second subproblem (exclude split item).

subsets of the size that corresponds to the depth of the new tree level. Afterward, all item sets that are found to be infrequent are eliminated (*a posteriori* pruning). Technical and optimization details can be found, for example, in Refs 33–37.

As an example, Figure 6 shows two steps of the Apriori algorithm for the transaction database shown in Figure 1, namely, adding the second and the third level (item sets with two and three items). *A posteriori* pruning is indicated by red crosses, and *Apriori* pruning by blue ones.

It should be noted, though, that nowadays the Apriori algorithm is mainly of historical interest as one of the first frequent item set mining and association rule induction algorithms, because its performance (in terms of both speed and memory consumption) usually cannot compete with that of state-of-the-art depth-first approaches.

Depth-First Search

Although a depth-first version of Apriori has been suggested,[38] depth-first search is usually known from algorithms such as Eclat,[3–5] FP-Growth,[6–9] LCM,[10–12] and many others. The general approach can be seen as a simple *divide-and-conquer* scheme. For a chosen item i, the problem to find all frequent item sets is split into two subproblems: (1) find all frequent item sets containing i and (2) find all frequent item sets *not* containing i. Each subproblem is then further divided based on another item j: find all frequent item sets containing (1.1) both i and j, (1.2) i, but not j, (2.1) j, but not i, (2.2) neither i nor j, and so on. This division scheme is illustrated in Figure 7.

All subproblems occurring in this recursion can be defined by a *conditional transaction database* and a *prefix*. The prefix is a set of items that has to be added to all frequent item sets that are discovered in the conditional transaction database. Formally, all subproblems are pairs $S = (C, P)$, where C is a conditional database and $P \subseteq B$ is a prefix. The initial problem, with which the recursion is started, is $S = (T, \emptyset)$, where T is the given transaction database and the prefix is empty.

A subproblem $S_0 = (C_0, P_0)$ is processed as follows: choose an item $i \in B_0$, where B_0 is the set of items occurring in C_0. This choice is, in principle, arbitrary, but often follows some predefined order of the items. If $s_{C_0}(\{i\}) \geq s_{\min}$, then report the item set $P_0 \cup \{i\}$ as frequent with the support $s_{C_0}(\{i\})$, and form the subproblem $S_1 = (C_1, P_1)$ with $P_1 = P_0 \cup \{i\}$. The conditional database C_1 comprises all transactions in C_0 that contain the item i, but with the item i removed. This also implies that transactions that contain no other item than i are entirely removed: no empty transactions are ever kept. If C_1 is not empty, process S_1 recursively. In any case (i.e., regardless of whether $s_{C_0}(\{i\}) \geq s_{\min}$ or not), form the subproblem $S_2 = (C_2, P_2)$, where $P_2 = P_0$. The conditional database C_2 comprises all transactions in C_0 (including those that do not contain the item i), but again with the item i (and resulting empty transactions) removed. If C_2 is not empty, process S_2 recursively.

Note that in this search scheme, due to the conditional transaction databases, one needs only to count the support of individual items (or singleton sets). As an illustration, Figure 8 shows the (top levels of) a subproblem tree resulting from this divide-and-conquer search. The indices of the transaction databases T indicate which (split) items have been included (no bar) or excluded (bar).

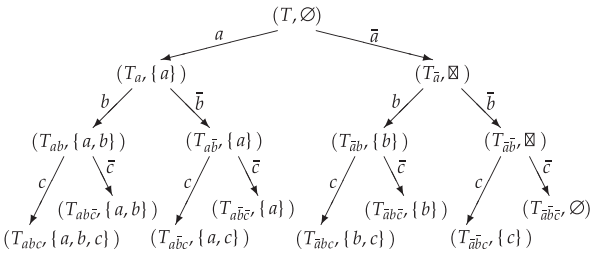

FIGURE 8 | (Top levels of) the subproblem tree of the divide-and-conquer scheme (with a globally fixed item order).

Order of the Subproblems

Whereas it is irrelevant in which order the child nodes are traversed in a breadth-first/levelwise scheme (since the whole next level is needed before the support counting can start), the depth-first search scheme allows for a choice whether the subproblem S_1 or the subproblem S_2 is processed first. In the former case, the item sets are considered in lexicographic order (w.r.t. the chosen item order), in the latter case this order is reversed. At first sight, this does not seem to make much of a difference. However, depending on how conditional transaction databases are represented, processing subproblem S_2 first can be advantageous, because then one may be able to use (part of) the memory storing the conditional database C_0 to store the conditional databases C_1 and C_2.

Intuitively, this can be understood as follows: C_2 is essentially C_0, only that the split item is eliminated or ignored. Hence it may not require extra memory, as C_0 (properly viewed) may be used instead (examples are provided in *Vertical Representation* and *Hybrid Representations*). C_1 is never larger than C_2 as it is a subset of the transactions in C_2, namely, those that contain the split item in C_0. Therefore, one may reuse C_2's (and thus C_0's) memory for representing and processing C_1. Hence, processing subproblem S_2 before S_1 can lead to effectively constant memory requirements (all computations are carried out on the memory storing the initial transaction database). This is demonstrated by LCM[10–12] and the so-called top-down version of FP-Growth.[39] However, if S_1 is solved before S_2, representing the transaction selection needed for S_1 may need extra memory, because the transaction database without this selection (i.e., all transactions) needs to be maintained for later solving S_2.

The traversal order can also be relevant if the output is to be confined to closed or maximal item sets, namely, if this restriction is achieved by a repository of already found closed item sets, which is queried for a superset with the same support: in this case S_1 must be processed before S_2. Analogously, if one filters for generators with a repository that is queried for a subset with the same support, S_2 must be processed before S_1.

Order of the Items

Apart from the order in which the subproblems are processed, the order in which the items are used to split the subproblems (or the order used to assign unique parents to single out a prefix tree from the Hasse diagram, see Figure 5) can have a considerable impact on the time needed for the search. Experimentally, it was determined very early that it is usually best to process the items in the order of increasing frequency or, even better, in the order of increasing size sum of the transactions they are contained in. The reason for this behavior is that the average size of the conditional transaction databases tends to be smaller if the items are processed in this order. This observation holds for basically all database representations (see *Vertical Representation* for more details).

Note, however, that the order of the items influences only the search time, *not* the result of the algorithms. To emphasize this fact, in the following some algorithms will be illustrated with a default alphabetic order of the items, others with items reordered according to their frequency in the given transaction database.

DATABASE REPRESENTATIONS

Algorithms that enumerate (frequent) item sets with the divide-and-conquer scheme outlined in *Depth-First Search* (such as Eclat, FP-growth, LCM, and so on) differ in how conditional transaction databases are represented: *horizontally* (*Horizontal Representation*), *vertically* (*Vertical Representation*), or in a *hybrid* fashion (*Hybrid Representations*), which combines a horizontal and a vertical representation. With the general algorithmic scheme of subproblem splits (see *Depth-First Search*), all that is needed in this section to obtain a concrete algorithm is to specify how the conditional transaction databases are constructed that are needed for the two subproblems.

Horizontal Representation

In a horizontal representation, a transaction database is stored as a list (or array) of transactions, each of which lists the items contained in it. This is the most obvious form of storing a transaction database, which was used in the problem definition in *Problem Definition* and for the example in Figure 1. The Apriori algorithm as well as some transaction intersection

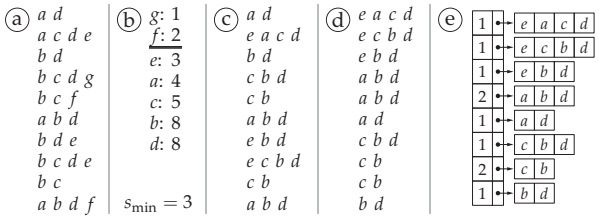

FIGURE 9 | Preprocessing a transaction database: (a) original form; (b) item frequencies; (c) transactions with sorted items and infrequent items eliminated ($s_{\min} = 3$); (d) lexicographically sorted reduced transactions; (e) data structure used in the SaM (Split and Merge) algorithm.

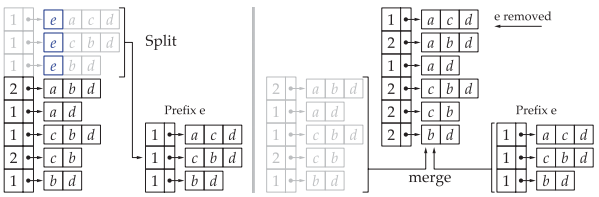

FIGURE 10 | The basic operations of the SaM (Split and Merge) algorithm: split (left; first subproblem, include split item) and merge (right; second subproblem, exclude split item).

approaches naturally use this representation. However, it can also be used in a divide-and-conquer/depth-first algorithm, as is demonstrated by the SaM (Split and Merge) algorithm.[40]

The SaM algorithm requires some preprocessing of the transaction database, which usually includes reordering the items according to their frequency in the transaction database. The steps of this preprocessing are illustrated for a simple example database in Figure 9, together with the final data structure, which is a simple array of transactions. Note, however, that equal transactions are merged and their multiplicity is kept in a counter.

How this list is processed in a subproblem split is demonstrated in Figure 10: The left part shows splitting off the transactions starting with the split item *e* (first subproblem; note that due to the lexicographic sorting done in the preprocessing all of these transactions are consecutive). The right part shows how the split-off transactions, with the split item removed, are merged with the rest of the transaction list (second subproblem). The merge operation is essentially a single phase of the *mergesort* sorting algorithm, with the only difference that it combines equal transactions (or transaction suffixes). It also ensures that the transactions (suffixes) are lexicographically sorted for the next subproblem split.

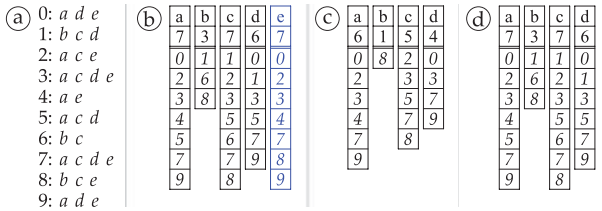

FIGURE 11 | Split into subproblems in the Eclat algorithm: (a) transaction database; (b) vertical representation with split item in blue; (c) conditional transaction database after intersection with split item list (first subproblem: include split item); (d) transaction index list of split item removed (second subproblem: exclude split item).

Vertical Representation

In a *vertical representation* of a transaction database, the items are first referred to with a list (or array) and for each item the transactions containing it are listed. As a consequence, a vertical scheme essentially represents the item covers $K_T(\{i\})$ for all $i \in B$. As an example, Figure 11(b) shows the vertical representation of the example transaction database of Figure 11(a) (the second row states the support values of the items).

A vertical representation is used in the Eclat algorithm[3] and its variants. The conditional transaction database for the first subproblem is constructed by intersecting the list of transaction indices for the split item with the lists of transaction indices for the other items. This is illustrated in Figure 11(b), in which the transaction index list for the split item is highlighted in blue, and Figure 11(c), which shows the result of the intersections and thus the conditional transaction database for the first subproblem. Constructing the conditional transaction database for the second subproblem is particularly simple: one merely has to delete the transaction index list for the split item, see Figure 11(d).

The execution time of the Eclat algorithm depends mainly on the length of the transaction index lists: the shorter these lists, the faster the algorithm. This explains why it is advantageous to process the items in the order of increasing frequency (see *Order of the Items*): if a split item has a low frequency, it has a short transaction index list. Intersecting this list with the transaction index lists of other items cannot yield a list that is longer than the list of the split item (this is another form of the *Apriori* property). Therefore, if the first split item has a low support, the transaction index lists that have to be processed in the recursion for the first problem have a low average length. Hence one should strive to use items with a low support as early as possible, that is, when there are still many other items, so that the lists of many items are reduced

by the intersection. Only later, when fewer items remain (in the branches for the second subproblem), items with higher support, which yield longer intersection lists, are processed, thus reducing the number of long lists. In this way the average length of the transaction lists is reduced. This insight, applied to the general average size of conditional transaction databases, can be transferred to other algorithms as well.

Especially, if the transaction database to mine is dense (that is, if the average transaction size (number of items) is large relative to the size of the item base), it can happen that intersecting two transaction index lists removes fewer transaction indices than it keeps. This observation led to the idea to represent a conditional transaction database not by covers, but by so-called *diffsets* (short for *difference sets*).[4] Diffsets are defined as follows: $\forall I \subseteq B : \forall a \in B - I : D_T(a \mid I) = K_T(I) - K_T(I \cup \{a\})$. In other words, $D_T(a \mid I)$ contains the indices of the transactions that contain I but not a. With diffsets, the support of direct supersets of I can be computed as $\forall I \subseteq B : \forall a \in B - I : s_T(I \cup \{a\}) = s_T(I) - |D_T(a \mid I)|$, and the diffsets for the next level (subproblems) can be computed with the help of $\forall I \in B : \forall a, b \in B - I, a \neq b : D_T(b \mid I \cup \{a\}) = D_T(b \mid I) - D_T(a \mid I)$.

As an alternative, Eclat can be improved for dense transaction databases by transferring certain elements of the FP-Growth algorithm. This extension, which uses lists of transaction ranges instead of plain lists of transaction indices, is described in the following section after the FP-Growth algorithm has been discussed.

Hybrid Representations

Although algorithms that use a purely horizontal or purely vertical transaction database representation are attractive (because they are, to some degree, conceptually simpler), they are often outperformed by algorithms that use a hybrid data structure, exploiting elements of both vertical and horizontal representations. The simplest form of such a hybrid structure can be found in the LCM algorithm[10–12]: it employs a purely vertical and a purely horizontal representation in parallel. Its processing scheme is best understood by viewing it as a variant of the Eclat algorithm, in which the intersection of transaction index lists is replaced by a scheme called *occurrence deliver*. This scheme accesses the horizontal representation to fill the transaction index lists as illustrated in Figure 12 for the same subproblem split used to illustrate the Eclat algorithm in Figure 11. The transaction index list of the split item is traversed and for each index

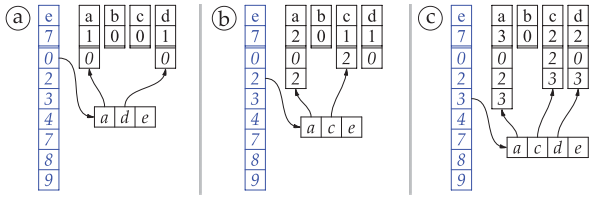

FIGURE 12 | Occurrence deliver scheme used by LCM (Linear time Closed item set Miner) to find the conditional transaction database for the first subproblem (needs a horizontal representation in parallel).

the corresponding transaction is retrieved. The item list of this transaction is then traversed (up to the split item) and the transaction index is added to the lists of the items that are encountered in the transaction.

LCM has the advantage that to construct the conditional transaction database (for the first subproblem) it only reads memory linearly and stores the transaction indices through direct access. In contrast to this, the intersection procedure of the standard Eclat algorithm needs if-then-else statements, which are difficult to predict by modern processors (see Ref 41 for details). However, LCM has the disadvantage that it is more difficult to eliminate infrequent and other removable items; even though their vertical representations can be eliminated, removing them also from the horizontal representation requires a special projection operation. Nevertheless, with state-of-the-art optimizations, LCM is one of the fastest frequent item set mining algorithms: it won (in version 2[11]) the second Frequent Item Set Mining Implementations (FIMI) Workshop competition[42] and version 3[12] is even faster.

Note that due to its hybrid transaction database representation, LCM can be implemented in such a way that only an amount of memory linear in the size of the transaction database is needed: if the second subproblem is processed before the first, the transaction index lists can be reused for the conditional transaction databases.

A more sophisticated hybrid structure is used by the FP-Growth algorithm,[6–9] namely, a so-called FP-Tree (Frequent Pattern Tree), which combines a horizontal and a vertical representation. The core idea is to represent a transaction database by a prefix tree, thus combining transactions with the same prefix. At the same time an FP-Tree keeps track of the transactions an item is contained in by linking the prefix tree nodes referring to the same item into a list. This structure is enhanced by a header table, each entry of which refers to one item and contains the head of the item list as well as the item support.

An FP-Tree is best explained by how it is constructed from a transaction database, which is

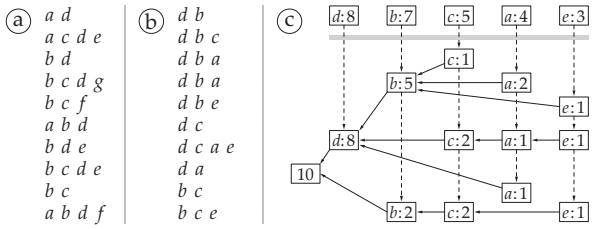

FIGURE 13 | Building an FP-tree (Frequent Pattern Tree): (a) original transaction database; (b) lexicographically sorted transactions (infrequent items eliminated, other items sorted according to descending frequency); (c) resulting FP-tree structure.

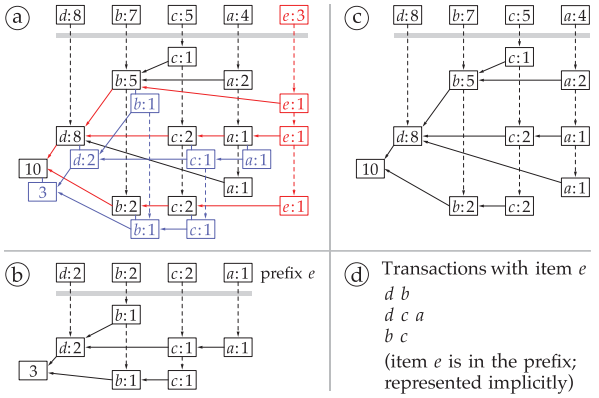

FIGURE 14 | Subproblem split in the FP-growth (Frequent Pattern Growth) algorithm: (a) projecting an FP-tree (Frequent Pattern Tree) (to item *e*); (b) detached projection (FP-tree of conditional transaction database); (c) remaining FP-tree after split item level is removed; (d) conditional transaction database in horizontal representation.

illustrated in Figure 13 for a simple example, shown in Figure 13(a). This transaction database is first preprocessed in a similar manner as for the SaM algorithm (see Figure 10) by sorting the items according to their frequency and then the transactions lexicographically. However, in contrast to the SaM algorithm, for an FP-Tree the items should be ordered according to descending frequency in the transactions. This transforms the transaction database into the form shown in Figure 13(b). From this database, the FP-Tree can be built directly by representing it as a prefix tree, see Figure 13(c). Note that sorting the items in the order of descending frequency is essential for obtaining a compact tree (even though it cannot be guaranteed that the tree needs less memory than a simple horizontal representation due to the overhead for the support and the pointers). Note also that the prefix tree is essentially a (compressed) horizontal representation, whereas the links between the branches (shown as dashed lines in Figure 13) are a vertical representation.

An FP-Tree is processed in the subproblem split as follows: the rightmost item is chosen as the split item and its list is traversed, see Figure 14(a). This selects all transactions that contain the split item. From nodes on the split item list, the parent pointers of the prefix tree are followed to recover the rest of the transactions containing the split item. To build the FP-Tree for the conditional transaction database, the encountered nodes are copied and linked. Note, however, that the support values are derived from the support values in the nodes for the split item: only these state in how many transactions the split item is contained. After the split item list has been fully traversed, the created FP-Tree is detached (see Figure 14b) and processed recursively (first subproblem). Note, however, that for an implementation it may be easier to extract the transactions one by one, see Figure 14(d) and to insert them into a new (and initially empty) FP-Tree than to copy nodes. Of course, in this approach no full horizontal database representation is created, but each extracted transaction is immediately inserted into a new tree and then discarded again, which can be achieved with a single transaction buffer. For the second subproblem, the nodes on the list for the split item are simply discarded (or ignored; note that an explicit deletion is not necessary) as shown in Figure 14(c).

The advantages of the FP-Growth algorithm are that its data structure allows for an easy removal of infrequent and other eliminated items and that it reduces, for some data sets, the amount of memory needed to store the transaction database (due to the combination of transactions with the same prefix). Its disadvantages are that the FP-Tree structure is comparatively complex and that the overhead for the support values and the pointers in the nodes can also have the effect that an FP-Tree is larger than a purely horizontal representation of the same database. Nevertheless, in a state-of-the-art implementation FP-Growth is usually one of the fastest frequent item set mining algorithms, as exemplified by the fact that the version of Ref 7 won the first FIMI Workshop competition.[43] An even faster version, which represents the FP-Tree very cleverly with the help of only two integer arrays was presented in Ref 8.

Variants of FP-Growth include a version that uses Patricia trees to improve the compression[44] and a compressed coding of a standard FP-Tree to process particularly large data sets.[45]

As already mentioned in *Vertical Representation*, certain aspects of the FP-Growth algorithm can be transferred to Eclat to improve the performance on dense data sets. The core idea is that Eclat also allows to combine transaction prefixes if one uses

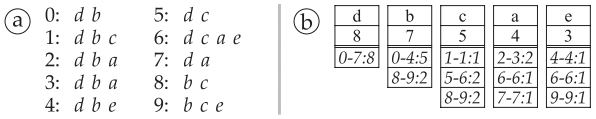

FIGURE 15 | Eclat with transaction ranges: (a) lexicographically sorted transactions; (b) transaction range lists.

transaction range lists instead of plain transaction index lists. This is illustrated in Figure 15 for the same example database used to illustrate FP-Growth (a range is represented as start−end:support). Clearly, using ranges can reduce the length of the lists considerably, especially for dense data sets, and can thus reduce the processing time, even though intersecting transaction range lists is more complex than intersecting plain transaction index lists.

ADVANCED TECHNIQUES

The basic algorithms as defined above by a general divide-and-conquer/depth-first approach and a representation form and processing scheme for conditional transaction databases can be enhanced by various optimizations to increase their speed. The following sections list the most important and most effective ones. Note, however, that none of these can change the fact that the asymptotic time complexity of frequent item set mining is essentially linear in the number of item sets, and thus potentially exponential in the number of items.

Reducing the Transaction Database

One of the simplest and most straightforward optimizations consists in eliminating all infrequent items from the initial transaction database. This not only removes items that cannot possibly be elements of frequent item sets (due to the *Apriori* property), but also increases the chances to find equal transactions in the database, which can be combined keeping their multiplicity in a counter. We already saw this technique for the SaM algorithm (see Figure 10) and it is implicit in the construction of an FP-Tree (see Figure 13), but it can also be applied for Eclat-style algorithms (including LCM). This makes the support counting slightly more complex in these algorithms, because it requires an additional array holding the transaction weights (multiplicities), but it can significantly improve speed for certain data sets and minimum support values and thus is worth the effort.

Perfect Extension Pruning

The basic divide-and-conquer/depth-first scheme can easily be improved with so-called *perfect extension pruning*: an item $i \notin I$ is called a *perfect extension* of an item set I (or a *parent equivalent item* or a *full support item*), iff I and $I \cup \{i\}$ have the same support. Perfect extensions have the following properties: (1) if an item i is a perfect extension of an item set I, then it is also a perfect extension of any item set $J \supseteq I$ as long as $i \notin J$ and (2) if K is the set of all perfect extensions of an item set I, then all sets $I \cup J$ with $J \in 2^K$ (power set of K) have the same support as I. These properties can be exploited by collecting in the recursion not only prefix items, but also, in a third element of a subproblem description, perfect extension items. These items are also removed from the conditional databases and are only used when an item set is reported to generate all supersets of the reported set that have the same support.

The correctness of this approach can also be understood by realizing that if a perfect extension item is chosen as the split item, the conditional transaction databases C_1 and C_2 are identical (since there are no transactions in C_0 that do not contain the split item). The only difference between the subproblems is the prefix, which contains the split item for the first subproblem, whereas it is missing in the second subproblem. As a consequence, the frequent item sets reported in the solutions of the two subproblems are essentially the same, only that the item sets reported for the first subproblem contain the split item, whereas those of the second subproblem do not. Or formally: let i be the split item and $\mathcal{S}(S_k)$ the frequent item sets in the solution of the kth subproblem, $k = 1, 2$. Then $I \in \mathcal{S}(S_2) \Leftrightarrow I \cup \{i\} \in \mathcal{S}(S_1)$.

Note that closed item sets (see *Closed and Maximal Item Sets and Generators*) may be defined as item sets that do not possess a perfect extension. However, note also that using perfect extension pruning is not sufficient to restrict the output to closed item sets (see *Closed and Maximal Item Set Filtering* for the additional requirements).

Few Items

One of the core issues of making an item set enumeration approach efficient is that in the recursion the set of items in the conditional transaction databases is reduced. This can make it possible to combine (projected) transactions that differed initially in some item, but became equal w.r.t. the reduced item base. We have seen a direct example of this in the illustration of the SaM algorithm (see Figure 10) and the FP-growth algorithm does essentially the same.

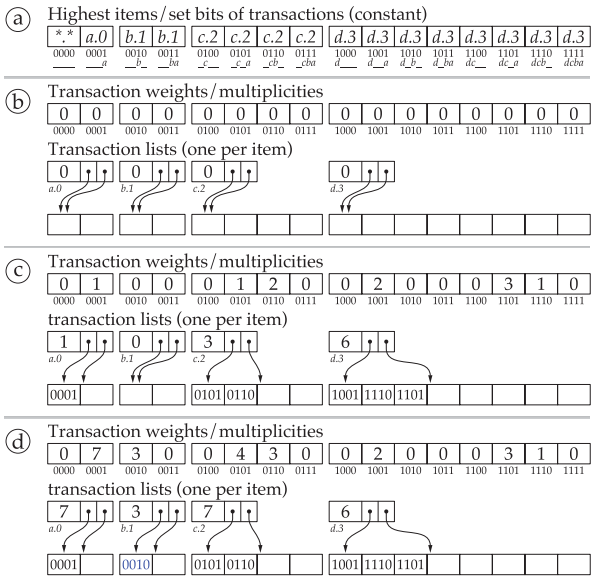

FIGURE 16 | Illustration of a 4-items machine: (a) table of highest set bits; (b) empty 4-items machine (no transactions); (c) after inserting the transactions of Figure 1 (without *e*); (d) after propagating the transaction lists left-/downward.

However, the computational overhead for this transaction combination procedure can be considerably reduced with a special data structure if the set of remaining items becomes sufficiently small (as it will necessarily be at some point in the recursion).

The core idea is to represent a certain number k of the most frequent items in a horizontal representation as bit arrays, in which each bit stands for an item. A set bit indicates that the item is contained in a transaction, a cleared bit that it is missing. Provided that k is sufficiently small, we can then represent all possible (projected) transactions with k items as indices of an integer array, which simply holds the multiplicities of these transactions. Together with transaction lists holding the transactions containing an item (in a similar style as for the LCM algorithm), we arrive at what will be called a k-items machine in the following. This data structure was proposed in Ref 12 for the LCM algorithm (version 3). However, the same technique can equally well be used with other algorithms, including SaM, Eclat (with plain transaction index lists as well as with transaction ranges), FP-Growth, and many others.

To illustrate this method, Figure 16 shows the elements of a 4-items machine as well as different states. Figure 16(b) shows the main components of the 4-items machine (in an empty state, i.e., without any transactions). The top array records the multiplicities of transactions, which are identified by their index (see Figure 16a for the correspondence of bit patterns and item sets). The bottom arrays record, for each item, the transactions it is contained in (in a bit representation). Note that these data structures explain why k needs to be small for a k-items machine to be feasible: the multiplicities array at the top has 2^k elements, which is also the total (maximum) size of the transaction lists.

Bit-represented transactions are entered into a k-items machine by simply indexing the transaction weights array to check whether an equal transaction is already contained: if the corresponding weight is zero, the transaction is not yet represented, otherwise it is. The principle underlying this aspect is essentially that of *binsort*. If a transaction is not yet represented, the array shown in Figure 16(a) is used to determine the highest set bit (and the item corresponding to it; in the array denoted as $i.b$, where i is the item and b the corresponding bit index). Then the transaction is appended to the list for this item. Executing this procedure for all transactions in the database of Figure 1 (without item *e*) yields Figure 16(c).

However, after this step only the list for the highest item (in this case *d*) lists all the transactions containing it, because a transaction has been added only to the list for its highest item. To complete the assignment, the transaction lists are traversed from the highest item downward, the bit corresponding to the list is masked out, and the (projected) transactions are, if necessary, assigned to the list for the next highest bit. If the transaction is already contained in this list (determined from the entry in the weights arrays as above), only the transaction weight/multiplicity is updated. This finally yields Figure 16(d), in which new list entries created in the second step are shown in blue and which can then be processed recursively in the usual way.

Conditional Item Reordering

Although the differences are often fairly small, in certain transaction databases the frequency order of the items in the conditional transaction databases can differ considerably from the global frequency order. Because the order in which split items are chosen is, in principle, arbitary, it can be beneficial to reorder the items to reduce the average size of the conditional transaction databases in the deeper levels of the recursion. This is one of the core features underlying the speed of the FP-growth implementations of Refs 7, 9. However, the same idea can be used (and even much more easily) with an Eclat-based scheme (although it is slightly more difficult to implement for the LCM variant). It should be noted, though, that the cost of reordering can also slow down the search

unnecessarily if the frequency orders do not differ much. It depends heavily on the data set whether item reordering is advantageous or not.

Generating the Output

In the described divide-and-conquer/depth-first scheme the item sets are enumerated in lexicographic order (if the first subproblem is processed before the second) or in reverse lexicographic order (if the second is processed before the first). This can be exploited to optimize the output, by avoiding the need to generate the textual description of each item set from scratch. Due to the lexicographic order, it will often be the case that the next item set to be reported shares a prefix with the last one reported. The output, and with it the whole search, can be made considerably faster by using an output buffer for the textual description of an item set. It is then recorded which part is still valid and this part is reused. That this should be relevant may sound surprising, but the benchmark results of Ref 41 clearly indicate that a well-designed output routine can be orders of magnitude faster than a basic one and thus can decide which implementation wins a speed competition.

Closed and Maximal Item Set Filtering

All techniques discussed up to now are essentially geared to find all frequent item sets. If one wants to confine the output to closed or maximal item sets, additional filtering procedures are needed. The most straightforward is perfect extension pruning: if an item set has a perfect extension, it cannot be closed (and thus also not maximal). However, not all perfect extensions are easy to detect. Only if the perfect extension item is in the conditional transaction database associated with the item set to check, it will be immediately detected in the recursion and thus can be used without effort to suppress the output of the item set in question.

However, if the path to the current subproblem contains calls for the second type of subproblem, there can be perfect extension items that will not be checked in the recursion, namely among those that were eliminated in these calls (eliminated items). To find such perfect extensions, essentially two approaches are employed: the first refers to the definition of a perfect extension, goes back to the original transaction database and checks whether there is an eliminated item that is contained in all originals of the projected transactions of the current conditional transaction database. This check is best carried out with a horizontal transaction representation by intersecting the set of eliminated items incrementally with the transactions. The advantage of such a scheme is that if the set becomes empty, there is no perfect extension and no further intersections are necessary. If, on the other hand, items remain after all relevant transactions have been intersected, there is a perfect extension, choices of which have even been identified in the intersection result. This approach is used in LCM,[10–12] where it is particularly fast, especially due to a bit representation of the k most frequent items.

An alternative to a direct test of the defining condition is to maintain a repository of already found closed item sets, which is queried for a superset with the same support. (Note that this requires that the first subproblem is processed before the second one.) If there is such a superset, there is a perfect extension among the eliminated items and the current item set cannot be closed.

It is important to note that a repository approach is competitive only if not only a single global repository is used, because the number of item sets accumulating in it in the course of the recursive search severely impedes the efficiency of looking up a found item set. The solution is to create, alongside the conditional transaction databases, conditional repositories, which are filtered in the same way by the prefix/split item. This approach works extremely well in the FP-Growth implementations of Refs 7, 9, where the repository is laid out as an FP-Tree, which makes the retrieval very efficient. However, a simple top-down prefix tree may be used as well.

Additional pruning techniques for closed item sets as well as special algorithms have been suggested, for example, in Refs 19, 20, 22, 23, 37, 46–48. Since maximal item sets are also closed, they allow for basically the same or at least analogous filtering techniques as closed item sets. Some additional techniques, as well as other specialized algorithms, can be found, for instance, in Refs 14–15, 49, 50.

INTERSECTING TRANSACTIONS

The general search schemes reviewed in the preceding sections all enumerate candidate item sets and then prune infrequent candidates. However, there exists an alternative, which intersects (sets of) transactions to find the closed (frequent) item sets. Different variants of this approach were proposed in Refs 24, 51, 52 and it was combined with an item set enumeration scheme in Ref 53.

The fundamental idea of methods based on intersecting transactions is that closed item sets cannot only be defined as in *Closed and Maximal Item Sets*

and Generators (no superset has the same support), but also as follows: an item set $I \subseteq B$ is (frequent and) closed if $s_T(I) = |K_T(I)| \geq s_{\min} \wedge I = \bigcap_{k \in K_T(I)} t_k$. In other words, an item set is closed if it is equal to the intersection of all transactions that contain it (its cover). This definition is obviously equivalent to the one given in *Closed and Maximal Item Sets and Generators*: if an item set is a proper subset of the intersection of the transactions it is contained in, there exists a superset (especially the intersection of the containing transactions itself) that has the same cover and thus the same support. If, however, an item set is equal to the intersection of the transactions containing it, adding any item will remove at least one transaction from its cover and will thus reduce the item set support.

Intersection approaches can nicely be justified in a formal way by analyzing the Galois connection between the set of all possible item sets 2^B and the set of all possible sets of transaction indices $2^{\{1,\ldots,m\}}$ (where m is the number of transactions),[54] as it was emphasized and explored in detail in Ref 55: the Galois connection gives rise to a bijective mapping between the closed item sets and closed sets of transaction indices.

The intersection approach is implemented in the Carpenter algorithm[24] by enumerating sets of transactions (or, equivalently, sets of transaction indices) and intersecting them. This is done with basically the same divide-and-conquer scheme as for the item set enumeration approaches, only that it is applied to transactions (i.e., items and transactions exchange their meaning, cf., Ref 55). Technically, the task to enumerate all transaction index sets is split into two subtasks: (1) enumerate all transaction index sets that contain the index 1 and (2) enumerate all transaction index sets that do not contain the index 1. These subtasks are then further divided w.r.t. the transaction index 2: enumerate all transaction index sets containing (1.1) both indices 1 and 2, (1.2) index 1, but not index 2, (2.1) index 2, but not index 1, (2.2) neither index 1 nor index 2, and so on.

Formally, all subproblems occurring in the recursion can be described by triples $S = (I, K, \ell)$. $K \subseteq \{1, \ldots, m\}$ is a set of transaction indices, $I = \bigcap_{k \in K} t_k$, that is, I is the item set that results from intersecting the transactions referred to by K, and ℓ is a transaction index, namely, the index of the next transaction to consider. The initial problem, with which the recursion is started, is $S = (B, \emptyset, 1)$, where B is the item base, no transactions have been intersected yet, and transaction 1 is the next to process.

A subproblem $S_0 = (I_0, K_0, \ell_0)$ is processed as follows: form the intersection $I_1 = I_0 \cap t_{\ell_0}$. If $I_1 = \emptyset$, do nothing (return from recursion). If $|K_0| + 1 \geq s_{\min}$, and there is no transaction t_j with $j \in \{1, \ldots, m\} - K_0$ such that $I_1 \subseteq t_j$, report I_1 with support $s_T(I_1) = |K_0| + 1$. If $\ell_0 < m$, form the subproblem $S_1 = (I_1, K_1, \ell_1)$ with $K_1 = K_0 \cup \{\ell_0\}$ and $\ell_1 = \ell_0 + 1$ and the subproblem $S_2 = (I_2, K_2, \ell_2)$ with $I_2 = I_0$, $K_2 = K_0$ and $\ell_2 = \ell_0 + 1$ and process them recursively. For the necessary optimizations to make this approach efficient, see Refs 24, 52, 56.

An alternative to transaction set enumeration is a scheme that maintains a repository of all closed item sets, which is updated by intersecting it with the next transaction (incremental approach).[51,56] To justify this approach formally, we consider the set of all closed frequent item sets for $s_{\min} = 1$, that is, the set $\mathcal{C}(T) = \left\{I \subseteq B \mid \exists S \subseteq T : S \neq \emptyset \wedge I = \bigcap_{t \in S} t\right\}$. This set satisfies the following simple recursive relation: (1) $\mathcal{C}(\emptyset) = \emptyset$, (2) $\mathcal{C}(T \cup \{t\}) = \mathcal{C}(T) \cup \{t\} \cup \{I \mid \exists s \in \mathcal{C}(T) : I = s \cap t\}$. As a consequence, we can start the procedure with an empty set of closed item sets and then process the transactions one by one, each time updating the set of closed item sets by adding the new transaction itself and the additional closed item sets that result from intersecting it with the already known closed item sets. In addition, the support of already known closed item sets may have to be updated. Details of an efficient data structure and updating scheme can be found in Ref 56.

It should be noted that the performance of the intersection approaches is usually not competitive for standard data sets (like supermarket data). However, they can be the method of choice for data sets with few transactions and (very) many items as they occur, for instance, in gene expression analysis[56,57] or text mining. On such data, transaction intersection approaches can outperform item set enumeration by orders of magnitude.

EXTENSIONS

Frequent item set mining approaches have been extended in various ways and considerable efforts have been made to reduce the output to the relevant patterns. This section is certainly far from complete and only strives to give a flavor of some of the many possibilities.

Association Rules

Although historically preceding frequent item set mining, association rules[58] have to be considered an extension. The reason is that association rule induction is a two step process: in the first step the frequent item sets are found, from which association rules are

generated in the second step. The idea is simply to split a frequent item set into two disjoint subsets, the union of which is the frequent item set (2-partition). One of the subsets is used as the antecedent of a rule, the other as its consequent. This rule is then evaluated by computing its so-called *confidence*, which for a rule $X \to Y$ and a given transaction database T is defined as $c_T(X \to Y) = s_T(X \cup Y)/s_T(X)$. Intuitively, the confidence estimates the conditional probability of the consequent given the antecedent of the rule. Rules are then filtered with a user-specified *minimum confidence* $c_{\min}$: only rules reaching or exceeding this threshold are reported.

Extensions of standard association rule induction include, among many others, the incorporation of taxonomies for the items,[59] quantitative association rules,[60] and fuzzy association rules,[61] which use fuzzy sets over continuous domains as items.

A popular way to rank association rules is the *lift*[62] $l_T(X \to Y) = \frac{c_T(X \to Y)}{c_T(\emptyset \to Y)} = \frac{c_T(X \to Y)}{s_T(Y)/m}$, which measures how much the relative frequency of Y is increased if the transactions are restricted to those that contain X. Alternatives include *leverage*[63] $\lambda_T(X \to Y) = s_T(X \cup Y) - s_T(X)s_T(Y)/m$, which states how much more often X and Y occur together than expected under independence, and *conviction*[62] $\gamma_T(X \to Y) = \frac{1-c_T(\emptyset \to Y)}{1-c_T(X \to Y)} = \frac{1-s_T(Y)/m}{1-c_T(X \to Y}$, which measures how much more often the rule would be incorrect if X and Y occurred independently. Overviews of ranking and selection measures can be found in Refs 64–66. In general, any measure for the dependence of two binary variables (X is contained in a transaction or not, Y is contained in a transaction or not) is applicable. Approaches based on statistical methods to select the best k patterns have been proposed, for example, in Ref 67, 68. Generally, the task to select relevant rules from the abundance that is produced by unfiltered mining has become a strong focus of current research.

Cover Similarity

Support-based frequent item set mining has the disadvantage that the support does not say much about the actual strength of association of the items in the set: a set of items may be frequent simply because its elements are frequent and thus their frequent co-occurrence can be expected by chance. As a consequence, the (usually few) interesting item sets drown in a sea of irrelevant ones.

One of several approaches to improve this situation is to replace the support by a different antimonotone measure. Such a measure can, for instance, be obtained by generalizing measures for the similarity of sets or binary vectors[69] to more than two arguments and applying them to the covers of the items in a set.[70]

As an example, consider the Jaccard index,[71] which for two sets A and B is defined as $J(A, B) = |A \cap B|/|A \cup B|$. Obviously, $J(A, B)$ is 1 if the sets coincide (i.e., $A = B$) and 0 if they are disjoint (i.e., $A \cap B = \emptyset$). To generalize this measure to more than two sets (here: item covers), one defines the *carrier* $L_T(I)$ of an item set I as $L_T(I) = \{k \in \{1, \ldots, m\} \mid I \cap t_k \neq \emptyset\} = \bigcup_{i \in I} K_T(\{i\})$. The *extent* $r_T(I)$ of an item set I w.r.t. a transaction database T is the size of its carrier, that is, $r_T(I) = |L_T(I)|$. Together with the notions of *cover* and *support* (see *Problem Definition*), the generalized Jaccard index of an item set I is then defined as its support divided by its extent, that is, as $J_T(I) = \frac{s_T(I)}{r_T(I)} = \frac{|\cap_{i \in I} K_T(\{i\})|}{|\cup_{i \in I} K_T(\{i\})|}$. It is easy to show that $J_T(I)$ is antimonotone.

Depending on the application, Jaccard item set mining can yield better and more informative results than a simple support-based mining, but should always be combined with an additional filter for the support.

Notable other modifications of the support criterion include a size-decreasing minimum support[72,72] (i.e., the larger an item set, the lower the support threshold) and using the area of the binary matrix tile corresponding to an item set as a selection criterion,[74] which is analogous to a size-dependent support.

Item Set Ranking and Selection

To reduce the number of reported item sets, additional measures to rank and filter them can be employed. A straightforward approach simply compares the support of an item set I to its expected support under independence, for example, as $e_T(I) = s_T(I)/\prod_{i \in I} s_T(i)$. However, this has the disadvantage that adding an independent item j to a well scoring set still yields a high value, as in this case $e_T(I \cup \{j\}) = s_T(j)s_T(I)/(s_T(j)\prod_{i \in I} s_T(i)) = e_T(I)$. Better approaches rely on measures for association rule evaluation and ranking (see *Association Rules*): form all possible association rules that can be created from a given item set I (or only those with a single item in the consequent) and aggregate (average, take minimum or maximum) their evaluations to obtain an evaluation for the item set I.

However, any such measure-based evaluation suffers from the *multiple testing problem* due to which one loses control of the significance level of statistical tests: in a large number of tests some positive results are to be expected simply by chance, which can

lead to many *false discoveries*. Common approaches to deal with this problem are to apply Bonferroni correction[75,76] or the Holm–Bonferroni method[77] in the search,[68,78] mining only part of the data and statistically validating the results on a hold-out subset[68] as well as randomization approaches,[79,80] which create surrogate data sets that implicitly encode the null hypothesis.

An extended problem is the selection of so-called *pattern sets*, for example, as sets of (binary) features for classification purposes. In this case, not individual item sets, but sets of such patterns are desired, for example, a (small) pattern set that covers the data well or exhibits little overlap between its member patterns (low redundancy). To find such pattern sets, various approaches have been devised, for example, finding pattern sets with which the data can be compressed well[81,82] or pattern sets in which all patterns contribute to partitioning the data.[83] A general framework for this task, which has become known as *constraint based pattern mining*, has been suggested in Ref 84. An alternative, statistics based reduction of the output in the spirit of closed item sets are *self-sufficient item sets*[85]: item sets the support of which is within expectation are removed.

Fault-Tolerant Item Sets

In standard frequent item set mining only transactions that contain *all* of the items in a given set are counted as supporting this set. In contrast to this, in fault-tolerant (or approximate) item set mining transactions that contain only a subset of the items can still support an item set, though possibly to a lesser degree than transactions containing all items. To cope with missing items in the transaction data to analyze, several fault-tolerant item set mining approaches have been proposed (for an overview see, e.g., Ref 86). They can be categorized roughly into three classes: (1) error-based, (2) density-based, and (3) cost-based approaches.

Error-based approaches: Examples of error-based approaches are Refs 87 and 88. In the former, the standard support measure is replaced by a fault-tolerant support, which allows for a maximum number of missing items in the supporting transactions, thus ensuring that the measure is still antimonotone. The search algorithm itself is derived from the Apriori algorithm (see *Breadth-First/Levelwise Search*). In Ref 88, constraints are placed on the number of missing items as well as on the number of (supporting) transactions that do not contain an item in the set. Hence, it is related to the tile-finding approach in Ref 89. However, it uses an enumeration search scheme that traverses sublattices of items and transactions, thus ensuring a complete search, whereas Ref 89 relies on a heuristic scheme.

Density-based approaches: Rather than fixing a maximum *number* of missing items, density-based approaches allow a certain *fraction* of the items in a set to be missing from the transactions, thus requiring the corresponding binary matrix tile to have a minimum density. This means that for larger item sets more items are allowed to be missing than for smaller item sets. As a consequence, the measure is no longer antimonotone if the density requirement is to be fulfilled by each individual transaction. To overcome this, Ref 90 requires only that the average density over all supporting transaction must exceed a user-specified threshold, whereas Ref 91 defines a recursive measure for the density of an item set. The approach in Ref 92 requires both items and transactions to satisfy a density constraint and defines a corresponding fault-tolerant support that allows for efficient mining.

Cost-based approaches: In error- or density-based approaches all transactions that satisfy the constraints contribute equally to the support of an item set, regardless of how many items of the set they contain. In contrast to this, cost-based approaches define the support contribution of transactions in proportion to the number of missing items. In Refs 40, 93, this is achieved by means of user-provided item-specific costs or penalties, with which missing items can be inserted. These costs are combined with each other and with the initial transaction weight of 1 with the help of a t-norm. In addition, a minimum weight for a transaction can be specified, by which the number of insertions can be limited.

Note that the cost-based approaches can be made to contain the error-based approaches as a limiting or extreme case, as one may set the cost/penalty of inserting an item in such a way that the transaction weight is not reduced. In this case, limiting the number of insertions obviously has the same effect as allowing for a maximum number of missing items.

Related to fault-tolerant item set mining—but nevertheless fundamentally different—is the case of uncertain transactional data. In such data, each item is endowed with a transaction-specific weight or probability, which is meant to indicate the degree or chance with which it is a member of the transaction. Approaches to this problem can be found, for example, in Refs 94–96. The problem of determining the (expected) support of an item set is best treated (in the case of independent occurrences of items in the transactions) by simple sampling and then applying standard frequent item set mining[97] or by using a normal distribution approximation.[98]

SUMMARY

Frequent item set mining has been a fruitful and intensely researched topic, which has produced remarkable results. The currently fastest frequent item set mining algorithms are the Eclat-variant LCM and FP-Growth, provided they are equipped with state-of-the-art optimizations, and there seems to be very little room left for speed improvements. Lasting challenges of frequent item set mining are to find better ways to filter the produced frequent item sets and association rules (or produce fewer in the first place), as even with the methods discussed above (such as closed and maximal item sets), the really interesting patterns still run the risk of drowning in a sea of irrelevant ones. Additional filtering with quality measures or statistical tests improves the situation, but still leaves a lot of room for improvements.

REFERENCES

1. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Databases* (*VLDB 1994*; Santiago de, Chile). San Mateo, CA: Morgan Kaufmann; 1994, 487–499.
2. Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo A. Fast discovery of association rules. In: *Advances in Knowledge Discovery and Data Mining*. Cambridge, CA: AAAI Press/MIT Press; 1996, 307–328.
3. Zaki MJ, Parthasarathy S, Ogihara M, Li W. New algorithms for fast discovery of association rules. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (*KDD 1997*; Newport Beach, CA). Menlo Park, CA: AAAI Press; 1997, 283–296.
4. Zaki MJ, Gouda K. Fast vertical mining using diffsets. In: *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2003*; Washington, DC). New York: ACM Press; 2003, 326–335.
5. Schmidt-Thieme L. Algorithmic features of Eclat. In: *Proceedings of the Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
6. Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: *Proceedings of the 19th ACM International Conference on Management of Data* (*SIGMOD 2000*; Dallas, TX). New York, NY: ACM Press; 2000, 1–12.
7. Grahne G, Zhu J. Efficiently using prefix-trees in mining frequent itemsets. In: *Proceedings of the Workshop Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
8. Rácz B. Nonordfp: an FP-growth variation without rebuilding the FP-tree. In: *Proceedings of the 2nd International Workshop on Frequent Itemset Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2003.
9. Grahne G, Zhu J. Reducing the main memory consumptions of Fpmax* and FPclose. In: *Proceedings of the Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
10. Uno T, Asai T, Uchida Y, Arimura H. LCM: an efficient algorithm for enumerating frequent closed item sets. In: *Proceedings of the Workshop on Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). TU Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
11. Uno T, Kiyomi M, Arimura H. LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: *Proceedings of the Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
12. Uno T, Kiyomi M, Arimura H. LCM ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In: *Proceedings of the 1st Open Source Data Mining on Frequent Pattern Mining Implementations* (*OSDM 2005*; Chicago, IL). New York, NY: ACM Press; 2005, 77–86.
13. Gerstein GL, Perkel DH, Subramanian KN. Identification of functionally related neural assemblies. Brain Res 1978, 140:43–62.
14. Bayardo RJ. Efficiently mining long patterns from databases. In: *Proceedings of the ACM International Conference Management of Data* (*SIGMOD 1998*; Seattle, WA). New York, NY: ACM Press; 1998, 85–93.
15. Lin D-I, Kedem ZM. Pincer-search: a new algorithm for discovering the maximum frequent set. In: *Proceedings of the 6th International Conference on Extending Database Technology* (*EDBT 1998*; Valencia, Spain). Heidelberg, Germany: Springer-Verlag; 1998, 103–119.
16. Agrawal RC, Aggarwal CC, Prasad VVV. Depth first generation of long patterns. In: *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2000*; Boston, MA). New York, NY: ACM Press; 2000, 108–118.

17. Aggarwal CC. Towards long pattern generation in dense databases. *SIGKDD Explor* 2001, 3:20–26.
18. Burdick D, Calimlim M, Gehrke J. MAFIA: a maximal frequent itemset algorithm for transactional databases. In: *Proceedings of the 17th International Conference on Data Engineering* (*ICDE 2001*; Heidelberg, Germany). Piscataway, NJ: IEEE Press; 2001, 443–452.
19. Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules. In: *Proceedings of the 7th International Conference on Database Theory* (*ICDT 1999*; Jerusalem, Israel). London, United Kingdom: Springer-Verlag; 1999, 398–416.
20. Bastide Y, Taouil R, Pasquier N, Stumme G, Lakhal L. Mining frequent patterns with counting inference. *SIGKDD Explor* 2002, 2:66–75.
21. Zaki MJ. Generating non-redundant association rules. In: *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2000*, Boston, MA). New York, NY: ACM Press; 2000, 34–43.
22. Cristofor D, Cristofor L, Simovici D. Galois connection and data mining. *J Univ Comput Sci* 2000, 6:60–73.
23. Pei J, Han J, Mao R. Closet: an efficient algorithm for mining frequent closed itemsets. In: *Proceedings of the SIGMOD International Workshop on Data Mining and Knowledge Discovery* (*DMKD 2000*; Dallas, TX). ACM Press, New York, NY; 2000, 21–30.
24. Pan F, Cong G, Tung AKH, Yang J, Zaki MJ. Carpenter: finding closed patterns in long biological datasets. In: *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2003*; Washington, DC). New York, NY: ACM Press; 2003, 637–642.
25. Bastide Y, Pasquier N, Taouil R, Stumme G, Lakhal L. Mining minimal non-redundant association rules using frequent closed itemsets. In: *Proceedings of the 1st International Conference on Computational Logic* (*CL 2000*; London, UK). London, United Kingdom: Springer-Verlag, 2000, 972–986.
26. Bykowski A, Rigotti C. A condensed representation to find frequent patterns. In: *Proceedings of the 20th ACM Symposium on Principles of Database Systems* (*PODS 2001*; Santa Barbara, CA). New York, NY: ACM Press; 2001, 267–273.
27. Kryszkiewicz M, Gajek M. Concise representation of frequent patterns based on generalized disjunction-free generators. In: *Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD 2002*; Paipei, Taiwan). New York, NY: Springer-Verlag; 2002, 159–171.
28. Liu G, Li J, Wong L, Hsu W. Positive borders or negative borders: how to make lossless generators based representations concise. In: *Proceedings of the 6th SIAM International Conference on Data Mining* (*SDM 2006*; Bethesda, MD). Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM); 2006, 469–473.
29. Liu G, Li J, Wong L. A new concise representation of frequent itemsets using generators and a positive border. *J Knowl Inf Syst* 2008, 17:35–56.
30. Kohavi R, Bradley CE, Frasca B, Mason L, Zheng Z. KDD-Cup 2000 organizers' report: peeling the onion. *SIGKDD Explor* 2000, 2:86–93.
31. Calders T, Goethals B. Mining all non-derivable frequent itemsets. In: *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery* (*PKDD 2002*; Helsinki, Finland). Berlin, Germany: Springer; 2002, 74–85.
32. Muhonen J, Toivonen H. Closed non-derivable itemsets. In: *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases* (*PKDD 2006*; Berlin, Germany). Berlin, Germany: Springer; 2006, 601–608.
33. Bodon F. A fast apriori implementation. In: *Proceedings of the Workshop on Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). TU Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
34. Borgelt C. Efficient implementations of apriori and Eclat. In: *Proceedings of the Workshop on Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). TU Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
35. Bodon F. Surprising results of trie-based fim algorithms. In: *Proceedings of the 2nd Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
36. Borgelt C. Recursion pruning for the apriori algorithm. In: *Proceedings of the 2nd Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
37. Bodon F, Schmidt-Thieme L. The relation of closed itemset mining, complete pruning strategies and item ordering in apriori-based FIM algorithms. In: *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases* (*PKDD 2005*; Porto, Portugal). Berlin Germany: Springer-Verlag; 2005.
38. Kosters WA, Pijls W. Apriori: a depth first implementation. In: *Proceedings of the Workshop on Frequent Item Set Mining Implementations* (FIMI 2003; Melbourne, FL). TU Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
39. Wang K, Tang L, Han J, Liu J. Top-down FP-growth for association rule mining. In: *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (*PAKDD 2002*; Taipei, Taiwan). London, United Kingdom: Springer-Verlag; 2002, 334–340.

40. Borgelt C, Wang X. SaM: a split and merge algorithm for fuzzy frequent item set mining. In: *Proceedings of the 13th International Fuzzy Systems Association World Congress and 6th Conference of European Society for Fuzzy Logic and Technology* (*IFSA/EUSFLAT'09*; Lisbon, Portugal). Lisbon, Portugal: IFSA/EUSFLAT Organization Committee; 2009, 968–973.
41. Rácz B, Bodon F, Schmidt-Thieme L. Benchmarking frequent itemset mining algorithms: from measurement to analysis. In: *Proceedings of the 1st Open Source Data Mining on Frequent Pattern Mining Implementations* (*OSDM 2005*, Chicago, IL). New York, NY: ACM Press; 2005, 36–45.
42. Bayardo R, Goethals B, Zaki MJ, eds. In: *Proceedings of the 2nd Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
43. Goethals B, Zaki MJ, eds. In: *Proceedings of the Workshop Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
44. Pietracaprina A, Zandolin D. Mining frequent itemsets using patricia tries. In: *Proceedings of the Workshop on Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). TU Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
45. Schlegel B, Gemulla R, Lehner W. Memory-efficient frequent-itemset mining. In: *Proceedings of the 14th International Conference on Extending Database Technology* (*EDBT 2011*; Uppsala, Sweden). New York, NY: ACM Press; 2011, 461–472.
46. Zaki MJ, Hsiao C-J. CHARM: an efficient algorithm for closed itemset mining. In: *Proceedings of the 2nd SIAM International Conference on Data Mining* (*SDM 2002*; Arlington, VA). Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM); 2002, 457–473.
47. Wang J, Han J, Pei J. Closet+: searching for the best strategies for mining frequent closed itemsets. In: *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2003*; Washington, DC). New York, NY: ACM Press; 2003.
48. Lucchese C, Orlando S, Perego R. DCI closed: a fast and memory efficient algorithm to mine frequent closed itemsets. In: *Proceedings of the 2nd Workshop Frequent Item Set Mining Implementations* (*FIMI 2004*; Brighton, UK). Aachen, Germany: CEUR Workshop Proceedings 126; 2004.
49. Gouda K, Zaki MJ. Efficiently mining maximal frequent itemsets. In: *Proceedings of the 1st IEEE International Conference on Data Mining* (*ICDM 2001*; San Jose, CA). Piscataway, NJ: IEEE Press; 2001, 163–170.
50. Burdick D, Calimlim M, Flannick J, Gehrke J, Yiu T. MAFIA: a performance study of mining maximal frequent itemsets. In: *Proceedings of the Workshop on Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). TU Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
51. Mielikäinen T. Intersecting data to closed sets with constraints. In: *Proceedings of the Workshop Frequent Item Set Mining Implementations* (*FIMI 2003*; Melbourne, FL). Aachen, Germany: CEUR Workshop Proceedings 90; 2003.
52. Cong G, Tan KI, Tung AKH, Pan F. Mining frequent closed patterns in microarray data. In: *Proceedings of the 4th IEEE International Conference on Data Mining* (*ICDM 2004*; Brighton, UK). Piscataway, NJ: IEEE Press; 2004, 363–366.
53. Pan F, Tung AKH, Cong G, Xu X. Cobbler: combining column and row enumeration for closed pattern discovery. In: *Proceedings of the 16th International Conference on Scientific and Statistical Database Management* (*SSDBM 2004*; Santori Island, Greece). Piscataway, NJ: IEEE Press; 2004, 21–30.
54. Ganter B, Wille R. *Formal Concept Analysis: Mathematical Foundations*. Berlin: Springer, 1999.
55. Rioult F, Boulicaut J-F, Crémilleux B, Besson J. Using transposition for pattern discovery from microarray data. In: *Proceedings of the 8th ACMSIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (*DMKD2003*; San Diego, CA). New York, NY: ACM Press; 2003, 73–79.
56. Borgelt C, Yang X, Nogales-Cadenas R, Carmona-Saez P, Pascual-Montano A. Finding closed frequent item sets by intersecting transactions. In: *Proceedings of the 14th International Conference on Extending Database Technology* (*EDBT 2011*; Uppsala, Sweden). New York, NY: ACM Press; 2011, 367–376.
57. Creighton C, Hanash S. Mining gene expression databases for association rules. Bioinformatics 2003, 19:79–86.
58. Agrawal R, Imielienski T, Swami A. Mining association rules between sets of items in large databases. In: *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD 1993*; Washington, DC). New York, NY: ACM Press; 1993, 207–216.
59. Srikant R, Agrawal R. Mining generalized association rules. In: *Proceedings of the 21st International Conference on Very Large Databases* (*VLDB 1995*; Zurich, Switzerland). San Mateo, CA: Morgan Kaufmann; 1995, 407–419.
60. Srikant R, Agrawal R. Mining quantitative association rules in large relational tables. In: *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD 1996*; Montreal, Canada). New York, NY: ACM Press; 1996, 1–12.
61. Kuok C, Fu A, Wong M. Mining fuzzy association rules in databases. SIGMOD Rec 1998, 27:41–46.
62. Brin S, Motwani R, Ullman JD, Tsur S. Dynamic itemset counting and implication rules for market basket data. In: *Proceedings of the ACM International*

Conference on Management of Data (*SIGMOD 1997*; Tucson, AZ). New York, NY: ACM Press; 1997, 265–276.

63. Piatetsky-Shapiro G. Discovery, analysis, and presentation of strong rules. In: Piatetsky-Shapiro G, Frawley WJ, eds. *Knowledge Discovery in Databases*. Palo Alto, CA: AAAI Press; 1991, 229–248.
64. Tan P-N, Kumar V, Srivastava J. Selecting the right interestingness measure for association patterns. In: *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2002*; Edmonton, Canada). New York, NY: ACM Press; 2002, 32–41.
65. Tan P-N, Kumar V, Srivastava J. Selecting the right objective measure for association analysis. *Inf Syst* 2004, 29:293–313.
66. Geng L, Hamilton HJ. Interestingness measures for data mining: a survey. *ACM Comput Surv (CSUR)* 2006, 38:Article 9.
67. Webb GI, Zhang S. *k*-Optimal-rule-discovery. *Data Min Knowl Discov* 2005, 10:39–79.
68. Webb GI. Discovering significant patterns. *Mach Learn* 2007, 68:1–33.
69. Choi S-S, Cha S-H, Tappert CC. A survey of binary similarity and distance measures. *J Syst Cybern Inf* 2010, 8:43–48.
70. Segond M, Borgelt C. Item set mining based on cover similarity. In: *Proceedings of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD 2011*; Shenzhen, China). Berlin, Germany: Springer-Verlag; 2011, LNCS 6635:493–505.
71. Jaccard P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin de la Société Vaudoise des Sciences Naturelles 1991; 37:547–579. France 1901.
72. Seno M, Karypis G. LPMiner: an algorithm for finding frequent itemsets using length decreasing support constraint. In: *Proceedings of the 1st IEEE International Conference on Data Mining* (*ICDM 2001*; San Jose, CA). Piscataway, NJ: IEEE Press; 2001, 505–512.
73. Wang J, Karypis G. BAMBOO: accelerating closed itemset mining by deeply pushing the length-decreasing support constraint. In: *Proceedings of the SIAM International Conference on Data Mining* (*SDM 2004;* Disneyworld, FL). Philadelphia, PA: Society for Industrial and Applied Mathematics; 2004, 432–436.
74. Geerts F, Goethals B, Mielikäinen T. Tiling databases. In: *Proceedings of the 7th International Conference on Discovery Science* (*DS 2004*; Padova, Italy). Berlin, Germany: Springer; 2004, 278–289.
75. Bonferroni CE. Il calcolo delle assicurazioni su gruppi di teste. Studi in Onore del Professore Salvatore Ortu Carboni 1935, 13–60.
76. Abdi H. Bonferroni and Sidák corrections for multiple comparisons. In: Salkind NJ, ed. *Encyclopedia of Measurement and Statistics*. Thousand Oaks, CA: Sage Publications; 2007.
77. Holm S. A simple sequentially rejective multiple test procedure. Scand J Stat 1979, 6:65–70.
78. Webb GI. Layered critical values: a powerful direct adjustment approach to discovering significant patterns. Mach Learn 2008, 71:307–323.
79. Megiddo N, Srikant R. Discovering predictive association rules. In: *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining* (*KDD 1998*; New York, NY). Menlo Park, CA: AAAI Press; 1998, 27–78.
80. Gionis A, Mannila H, Mielikäinen T, Tsaparas P. Assessing data mining results via swap randomization. In: *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2006*; Philadelphia, PA). New York, NY: ACM Press; 2006, 167–176.
81. Siebes A, Vreeken J, van Leeuwen M. Item sets that compress. In: *Proceedings of the SIAM International Conference on Data Mining* (*SDM 2006*; Bethesda, MD). Philadelphia, PA: Society for Industrial and Applied Mathematics; 2006, 393–404.
82. Vreeken J, van Leeuwen M, Siebes A. Krimp: mining itemsets that compress. Data Min Knowl Discov 2011, 23:169–214.
83. Bringmann B, Zimmermann A. The chosen few: on identifying valuable patterns. In: *Proceedings of the 7th IEEE International Conference on Data Mining* (*ICDM 2007*; Omaha, NE). Piscataway, NJ: IEEE Press; 2007, 63–72.
84. De Raedt L, Zimmermann A. Constraint-based pattern set mining. In: *Proceedings of the 7th IEEE International Conference on Data Mining* (*ICDM 2007*; Omaha, NE). Piscataway, NJ: IEEE Press; 2007, 237–248.
85. Webb GI. Self-sufficient itemsets: an approach to screening potentially interesting associations between items. *ACM Trans Knowl Discov Data (TKDD)* 2010, 4:Article 3.
86. Cheng H, Yu PS, Han J. Approximate frequent itemset mining in the presence of random noise. In: Maimon O, Rokach L, eds. *Soft Computing for Knowledge Discovery and Data Mining*. Vol. IV. Berlin: Springer; 2008, 363–389.
87. Pei J, Tung AKH, Han J. Fault-tolerant frequent pattern mining: problems and challenges. In: *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (*DMKD 2001*; Santa Babara, CA). New York, NY: ACM Press; 2001.
88. Besson J, Robardet C, Boulicaut J-F. Mining a new fault-tolerant pattern type as an alternative to formal concept discovery. In: *Proceedings of the International Conference on Computational Science* (*ICCS 2006*; Reading, United Kingdom). Berlin, Germany: Springer-Verlag; 2006, 144–157.

89. Gionis A, Mannila H, Seppänen JK. Geometric and combinatorial tiles in 0-1 data. In: *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases* (*PKDD 2004*; Pisa, Italy). Berlin, Germany: Springer-Verlag; 2004, LNAI 3202:173–184.
90. Yang C, Fayyad U, Bradley PS. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In: *Proceedings of the 7th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2001*; San Francisco, CA). New York, NY: ACM Press; 2001, 194–203.
91. Seppänen JK, Mannila H. Dense itemsets. In: *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2004*; Seattle, WA). New York, NY: ACM Press; 2004, 683–688.
92. Liu J, Paulsen S, Sun X, Wang W, Nobel A, Prins J. Mining approximate frequent itemsets in the presence of noise: algorithm and analysis. In: *Proceedings of the 6th SIAM Conference on Data Mining* (*SDM 2006*; Bethesda, MD). Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM); 2006, 405–416.
93. Wang X, Borgelt C, Kruse R. Mining fuzzy frequent item sets. In: *Proceedings of the 11th International Fuzzy Systems Association World Congress* (*IFSA 2005*; Beijing, China). Beijing, China; Heidelberg, Germany: Tsinghua University Press; Springer-Verlag; 2005, 528–533.
94. Chui C-K, Kao B, Hung E. Mining frequent itemsets from uncertain data. In: *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD 2007*; Nanjing, China). Berlin, Germany: Springer-Verlag; 2007, 47–58.
95. Leung CK-S, Carmichael CL, Hao B. Efficient mining of frequent patterns from uncertain data. In: *Proceedings of the 7th IEEE International Conference on Data Mining Workshops* (*ICDMW 2007*; Omaha, NE). Piscataway, NJ: IEEE Press; 2007, 489–494.
96. Aggarwal CC, Lin Y, Wang J, Wang J. Frequent pattern mining with uncertain data. In: *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining* (*KDD 2009*; Paris, France). New York, NY: ACM Press; 2009, 29–38.
97. Calders T, Garboni C, Goethals B. Efficient pattern mining of uncertain data with sampling. In: *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD 2010*; Hyderabad, India). Berlin, Germany: Springer-Verlag; 2010, I:480–487.
98. Calders T, Garboni C, Goethals B. Approximation of frequentness probability of itemsets in uncertain data. In: *Proceedings of the IEEE International Conference on Data Mining* (*ICDM 2010*; Sydney, Australia). Piscataway, NJ: IEEE Press; 2010, 749–754.

