Chapter 1

Knowledge Discovery in Databases: The Search for Frequent Patterns


1.1 Finding frequent sets

The simple data model we consider is the following.

Definition 1.1.1 Given a set \( R \), a 0/1 relation \( r \) over \( R \) is a collection (or multiset) of subsets of \( R \). The elements of \( R \) are called items, and the elements of \( r \) are called rows or transactions. The number of transactions in \( r \) is denoted by \( |r| \), and the size of \( r \) is \( ||r|| = \sum_{t \in r} |t| \). □

Definition 1.1.2 Let \( R \) be a set and \( r \) a 0/1 relation over \( R \), and let \( X \subseteq R \) be a set of items. The item set \( X \) matches a transaction \( t \in r \), if \( X \subseteq t \), we also say that \( t \) supports \( X \). The (multi) set of transactions in \( r \) matched by \( X \) is the support of \( X \), denoted by \( supp_r(X) \), i.e., \( supp_r(X) = \{ t \in r \mid X \subseteq t \} \). The frequency of \( X \) in \( r \), denoted by \( fr(X,r) \), is \( \frac{|supp_r(X)|}{|r|} \). We write simply \( supp(X) \) and \( fr(X) \) if the database is unambiguous in the context. Given a frequency threshold \( \text{min}_fr \in [0,1] \), the set \( X \) is frequent\(^1\) if \( fr(X,r) \geq \text{min}_fr \). □

Definition 1.1.3 Let \( R \) be a set, \( r \) a 0/1 relation over \( R \), and \( \text{min}_fr \) a frequency threshold. The collection of frequent sets in \( r \) with respect to \( \text{min}_fr \) is denoted by \( \mathcal{F}(r, \text{min}_fr) \),

\[
\mathcal{F}(r, \text{min}_fr) = \{ X \subseteq R \mid fr(X,r) \geq \text{min}_fr \},
\]

\(^1\)In the literature, also the terms large and covering have been used for “frequent”, and the term support for “frequency”.
or simply by \( \mathcal{F}(r) \) if the frequency threshold is clear in the context. The collection of frequent sets of size \( l \) is denoted by \( \mathcal{F}_l(r) = \{ X \in \mathcal{F}(r) \mid |X| = l \} \).

Exhaustive search of frequent sets is obviously infeasible for all but the smallest sets \( R \): the search space of potential frequent sets consists of the \( 2^{|R|} \) subsets of \( R \). A more efficient method for the discovery of frequent sets can be based on the following iterative approach. For each \( l = 1, 2, \ldots \), first determine a collection \( C_l \) of candidate sets of size \( l \) such that \( \mathcal{F}_l(r) \subseteq C_l \), and then obtain the collection \( \mathcal{F}_l(r) \) of frequent sets by computing the frequencies of the candidates from the database.

For large data collections, the computation of frequencies from the database is expensive. Therefore it is useful to minimize the number of candidates, even at the cost of the generation phase. To generate a small but sufficient collection of candidates, observe the following properties of item sets. Obviously a subset of items is at least as frequent as its superset, i.e., frequency is monotone increasing with respect to contraction of the set. This means that for any sets \( X \) and \( Y \) of items such that \( Y \subseteq X \), we have \( \text{supp}(Y) \supseteq \text{supp}(X) \) and \( fr(Y) \geq fr(X) \), and we have that if \( X \) is frequent then \( Y \) is also frequent. Proposition 1.1.4 takes advantage of this observation and gives useful information for candidate generation: given a set \( X \), if any of the subsets of \( X \) is not frequent then \( X \) can be safely discarded from the candidate collection \( C_{|X|} \). The proposition also states that it actually suffices to know if all subsets one smaller than \( X \) are frequent or not.

**Proposition 1.1.4** Let \( X \subseteq R \) be a set. If any of the proper subsets \( Y \subset X \) is not frequent then (1) \( X \) is not frequent and (2) there is a non-frequent subset \( Z \subset X \) of size \( |X| - 1 \).

**Proof** Claim (1) follows directly from the observation that if \( X \) is frequent then all subsets \( Y \subset X \) are frequent. The same argument applies for claim (2): for any \( Y \subset X \) there exists \( Z \) such that \( Y \subseteq Z \subset X \) and \( |Z| = |X| - 1 \). If \( Y \) is not frequent, then \( Z \) is not frequent.

**Example 1.1.5** If we know that

\[
\mathcal{F}_2(r) = \{ \{A, B\}, \{A, C\}, \{A, E\}, \{A, F\}, \{B, C\}, \{B, E\}, \{C, G\} \},
\]

then we can conclude that \( \{A, B, C\} \) and \( \{A, B, E\} \) are the only possible members of \( \mathcal{F}_3(r) \), since they are the only sets of size 3 whose all subsets of size 2 are included in \( \mathcal{F}_2(r) \). Further on, we know that \( \mathcal{F}_4(r) \) must be empty.

We now use Proposition 1.1.4 to define a candidate collection of sets of size \( l + 1 \) to consist of those sets that can possibly be frequent, given the frequent sets of size \( l \). This definition is the heart of the Apriori algorithm.
Algorithm 1.1.7
Input: A set $R$, a 0/1 relation $r$ over $R$, and a frequency threshold $\text{min}_fr$.
Output: The collection $\mathcal{F}(r, \text{min}_fr)$ of frequent sets and their frequencies.
Method:
1. $C_1 := \{\{A\} \mid A \in R\};$
2. $l := 1;$
3. while $C_l \neq \emptyset$ do
4. // Database pass
5. compute $F_l(r) := \{X \in C_l \mid fr(X, r) \geq \text{min}_fr\};$
6. $l := l + 1;$
7. // Apriori candidate generation (Algorithm 1.1.11):
8. compute $C_l := C(F_{l-1}(r));$
9. for all $l$ and for all $X \in F_l(r)$ do output $X$ and $fr(X, r);$

Definition 1.1.6 Given a collection $F_l(r)$ of frequent sets of size $l$, the (Apriori) candidate collection
generated from $F_l(r)$, denoted by $C(F_l(r))$, is
the collection of sets of size $l + 1$ that can possibly be frequent:

$$C(F_l(r)) = \{X \subseteq R \mid |X| = l + 1 \text{ and } Y \in F_l(r) \text{ for all } Y \subseteq X, |Y| = l\}.$$

We now finally give Algorithm 1.1.7 (Apriori) that finds all frequent sets. The subtasks of the algorithm, for which only specifications are given, are described in detail in following subsections.

Theorem 1.1.8 Algorithm 1.1.7 works correctly.

Proof We show by induction on $l$ that $F_l(r)$ is computed correctly for all $l$. For $l = 1$, the collection $C_1$ contains all sets of size one (line 1), and collection $F_1(r)$ contains then correctly exactly those that are frequent (line 5). For $l > 1$, assume $F_{l-1}(r)$ has been correctly computed. Then we have $F_l(r) \subseteq C_l = C(F_{l-1}(r))$ by Proposition 1.1.4 (line 8). Collection $F_l(r)$ is then correctly computed to contain the frequent sets (line 5).

Note also that the algorithm computes $F_{|X|}(r)$ for each frequent set $X$: since $X$ is frequent, there are frequent sets—at least the subsets of $X$—of sizes 1 to $|X|$, so the ending condition $C_l = \emptyset$ is not true for $l \leq |X|$.

From Proposition 1.1.4 it follows that Definition 1.1.6 gives a sufficiently large candidate collection. Theorem 1.1.9, below, shows that the definition gives the smallest possible candidate collection in general.

Theorem 1.1.9 For any collection $\mathcal{S}$ of subsets of $X$ of size $l$, there exists a
0/1 relation $r$ over $R$ and a frequency threshold $\text{min}_fr$ such that $F_l(r) = \mathcal{S}$ and $F_{l+1}(r) = C(\mathcal{S})$.

Proof We use a simple trick: set $r = \mathcal{S} \cup C(\mathcal{S})$ and $\text{min}_fr = 1/|r|$. Now all sets in $\mathcal{S}$ and $C(\mathcal{S})$ are frequent, i.e., $\mathcal{S} \subseteq F_l(r)$ and $C(\mathcal{S}) \subseteq F_{l+1}(r)$. 
Further on, $F_{l+1}(r) \subseteq C(S)$ since there are no other sets of size $l+1$ in $r$. To complete the proof we show by contradiction that $F_l(r) \subseteq S$. Assume that $Y \in F_l(r)$ is not in $S$. Then there must be $X \in C(S)$ such that $Y \subset X$. However, by Definition 1.1.6 all subsets of $X$ of size $l$ are in $S$.  

In candidate generation, more information can be used than just whether all subsets are frequent or not, and this way the number of candidates can be further reduced. Sometimes even the exact frequency of a set can be inferred.

Example 1.1.10 Assume sets $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{B, C\}$, and $\{B, D\}$ are frequent. Definition 1.1.6 gives $\{A, B, C\}$ and $\{A, B, D\}$ as candidates for $l = 3$, and Theorem 1.1.9 shows that such a 0/1 relation exists where $\{A, B, C\}$ and $\{A, B, D\}$ are indeed frequent.

If, however, we know that $fr(\{A, B, C\}) = fr(\{A, B\})$, i.e. $\{A, B\}$ is not closed, then we can infer that $fr(\{A, B, D\}) < \min_{fr}$. Intuitively, item $C$ partitions the database: all of $\{A, B\}$ occurs with $C$, but less than $\min_{fr}$ of $D$ occurs with $C$, since $fr(\{C, D\}) < \min_{fr}$, and therefore $\{A, B, D\}$ cannot be frequent. If the frequency of $\{A, B, C\}$ is computed first, it is not necessary to compute the frequency of $\{A, B, D\}$ from the database at all.

We have a slightly different situation if $fr(\{A, B\}) = fr(\{A\})$. Then we have $supp(\{A\}) \subseteq supp(\{B\})$, so $supp(\{A, C\}) = supp(\{A, B, C\})$ and $fr(\{A, C\}) = fr(\{A, B, C\})$. Thus the frequency $fr(\{A, B, C\})$ needs not to be computed from the database.

An algorithm such as 1.1.7 can be adapted to generate directly only the closed or maximal patterns I can also, in principle, take advantage of situations similar to the above examples. Such situations do not, however, occur frequently, and the effort saved can be less than the effort put into finding these cases. Furthermore, Algorithm 1.1.7 combines the computations of the frequencies of all candidate sets of size $l$ to one pass; the number of database passes would seldom be reduced.

1.1.1 Apriori candidate generation

We now consider the computation of candidate collections as defined in Definition 1.1.6. The trivial method to compute the candidate collection $C(F_l(r))$ is to check for each possible set of size $l+1$ whether the definition holds, i.e., if all its $l+1$ subsets of size $l$ are frequent. A more efficient way is to first compute potential candidates as unions $X \cup Y$ of size $l+1$ such that $X$ and $Y$ are frequent sets of size $l$, and then to check the rest of their subsets of size $l$. Algorithm 1.1.11 presents such a method. For efficiency reasons, it is assumed that both item sets and collections of item sets are stored as arrays, sorted in the lexicographical order. We write $X < Y$ to denote that $X$ precedes $Y$ in the lexicographical order.
Algorithm 1.1.11
Input: A lexicographically sorted array $\mathcal{F}_l(r)$ of frequent sets of size $l$.
Output: $\mathcal{C}(\mathcal{F}_l(r))$ in lexicographical order.
Method:
1. for all $X \in \mathcal{F}_l(r)$ do
2. for all $Y \in \mathcal{F}_l(r)$ such that $X < Y$ and $X$ and $Y$ share their $l-1$ lexicographically first items do
3. for all $Z \subset (X \cup Y)$ such that $|Z| = l$ do
4. if $Z$ is not in $\mathcal{F}_l(r)$ then continue with the next $Y$ at line 2;
5. output $X \cup Y$;

Theorem 1.1.12 Algorithm 1.1.11 works correctly.
Proof First we show that the collection of potential candidates $X \cup Y$ considered by the algorithm is a superset of $\mathcal{C}(\mathcal{F}_l(r))$. Given a set $W$ in $\mathcal{C}(\mathcal{F}_l(r))$, consider the subsets of $W$ of size $l$, and denote by $X'$ and $Y'$ the first and the second subset in the lexicographical order, respectively. Then $X'$ and $Y'$ share the $l-1$ lexicographically first items of $W$. Since $W$ is a valid candidate, $X'$ and $Y'$ are in $\mathcal{F}_l(r)$. In the algorithm, $X$ iterates over all sets in $\mathcal{F}_l(r)$, and at some phase we have $X = X'$. Now note that every set between $X'$ and $Y'$ in the lexicographical ordering of $\mathcal{F}_l(r)$ must share the same $l-1$ lexicographically first items. Thus we have $Y = Y'$ in some iteration while $X = X'$. Hence we find a superset of the collection of all candidates. Finally, a potential candidate is correctly output if and only if all of its subsets of size $l$ are frequent (line 4).

The time complexity of Algorithm 1.1.11 is polynomial in the size of the collection of frequent sets and it is independent of the database size.

Theorem 1.1.13 Algorithm 1.1.11 can be implemented to run in time $O(l^2 |\mathcal{F}_l(r)|^2 \log |\mathcal{F}_l(r)|)$.
Proof The outer loop (line 1) and the inner loop (line 2) are both iterated $O(|\mathcal{F}_l(r)|)$ times. Given $X$ and $Y$, the conditions on line 2 can be tested in time $O(l)^2$. On line 4, the remaining $l-1$ subsets need to be checked. With binary search, a set of size $l$ can be located from $\mathcal{F}_l(r)$ in time $O(l \log |\mathcal{F}_l(r)|)$. The output on line 5 takes time $O(l)$ for each potential candidate. The total time complexity is thus $O(|\mathcal{F}_l(r)|^2 (l + (l - 1)l \log |\mathcal{F}_l(r)| + l)) = O(l^2 |\mathcal{F}_l(r)|^2 \log |\mathcal{F}_l(r)|)$.

The upper bound of Theorem 1.1.13 is met when $l = 1$: all pairs of frequent sets of size 1 are created. After that the number of iterations of the inner loop on line 2 is typically only a fraction of $|\mathcal{F}_l(r)|$.

Actually, the values of $Y$ can be determined more efficiently with some extra bookkeeping information stored every time candidates are generated. A more detailed method using this idea is presented in Section 1.1.2.
CHAPTER 1. KNOWLEDGE DISCOVERY IN DATABASES

Instead of only computing $C(F_l(r))$, several successive families $C(F_l(r)), C(C(F_l(r))), C(C(C(F_l(r))))$, \ldots can be computed and then checked in a single database pass. This trades off a reduction in the number of database passes against an increase in the number of candidates, i.e., database processing against main memory processing. Candidates of size $l + 2$ are computed assuming that all candidates of size $l + 1$ are in fact frequent, and therefore $C(F_{l+1}(r)) \subseteq C(C(F_l(r)))$. Several candidate families can be computed by several calls to Algorithm 1.1.11.

Generating several candidate families is useful when the overhead of generating and testing the extra candidates $C(C(F_l(r))) \setminus C(F_{l+1}(r))$ is less than the effort of a database pass. Unfortunately, estimating the volume of extra candidates is in general difficult. The obviously useful situations are when $|C(C(F_l(r)))|$ is small.

Example 1.1.14 Assume

$$F_2(r) = \{\{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}, \{B, C\}, \{B, D\}, \{B, G\}, \{C, D\}, \{F, G\}\}.$$  

Then we have

$$C(F_2(r)) = \{\{A, B, C\}, \{A, B, D\}, \{A, C, D\}, \{B, C, D\}\},$$
$$C(C(F_2(r))) = \{\{A, B, C, D\}\},$$
$$C(C(C(F_2(r)))) = \emptyset.$$  

It would be practical to evaluate the frequency of all 5 candidates in a single pass.

1.1.2 Generation of candidate episodes

We present in detail a candidate generation method which is a generalization of the candidate generation for frequent sets. The method can be adapted to deal with sets or multisets of items, whether ordered (i.e. sequences) or not.

Algorithm 1.1.15 computes the candidates for unordered multisets of items, later called bags. In the algorithm, such a bag is represented as a lexicographically sorted array of items. The array is denoted by the name of the bag and the items in the array are referred to using square brackets. For example, a bag with items $A, C, C,$ and $F$ is represented as an array $\alpha$ with $\alpha[1] = A, \alpha[2] = C, \alpha[3] = C,$ and $\alpha[4] = F$. Collections of bags are also represented as lexicographically sorted arrays, i.e., the $i$th bag of a collection $F$ is denoted by $F[i]$.

Since the bags and bag collections are sorted, all bags that share the same first items are consecutive in the bag collection. In particular, if bags $F_l[i]$ and $F_l[j]$ of size $l$ share the first $l - 1$ items, then for all $k$ with $i \leq k \leq j$
Algorithm 1.1.15

Input: A sorted array $F_l$ of frequent bags of size $l$.
Output: A sorted array of candidate bags of size $l+1$.

Method:
1. $C_{l+1} := \emptyset$;
2. $k := 0$;
3. if $l = 1$ then All bags belong to the same block;
4. for $i := 1$ to $|F_l|$ do
5. for ($j := i; F_l[j]$ in the same block; $j := j + 1$) do
6. // $F_l[i]$ and $F_l[j]$ have $l - 1$ first event types in common,
7. // build a potential candidate $\alpha$ as their combination:
8. for $x := 1$ to $l$ do $\alpha[x] := F_l[i][x]$;
10. for $y := 1$ to $l - 1$ do
11. // Build and test subbags $\beta$ that do not contain $\alpha[y]$:
12. for $x := 1$ to $y - 1$ do $\beta[x] := \alpha[x]$;
13. for $x := y$ to $l$ do $\beta[x] := \alpha[x + 1]$;
14. if $\beta$ is not in $F_l$ then continue with the next $j$ at line 5;
15. // All subbags are in $F_l$, store $\alpha$ as candidate:
16. $k := k + 1$;
17. $C_{l+1}[k] := \alpha$;
18. output $C_{l+1}$;

we have that $F_l[k]$ shares also the same items. A maximal sequence of consecutive bags of size $l$ that share the first $l - 1$ items is called a block. Potential candidates can be identified by creating all combinations of two bags in the same block.

Theorem 1.1.16 Algorithm 1.1.15 works correctly.

Proof The crucial claim is that in the algorithm the pairs $F_l[i]$ and $F_l[j]$ of bags generate all candidates. —In the following we identify a bag with its index in the collection.

In the outer loop (line 4) variable $i$ iterates through all bags in $F_l$, and in the inner loop (line 5) variable $j$ iterates through those bags in $F_l$ that are in the same block with $i$ but are not before $i$. Consider now any block $b$ of bags in $F_l$. Variables $i$ and $j$ obviously iterate through all (unordered) pairs of bags in block $b$, including the case where $i = j$.

Since $i$ and $j$ are in the same block, they have the same $l - 1$ first items. Conceptually we construct a new potential candidate $\alpha$ as the union of bags $i$ and $j$. We build $\alpha$ by taking first the common $l - 1$ items and the $l$th item from bag $i$ (both done on line 8), and finally the item number $l + 1$ from bag $j$ (line 9). Then the items of a potential candidate are lexicographically sorted. Since the iteration of bags proceeds in lexicographical order (over the sorted collection $F_l$), the collection of candidates is also constructed in lexicographical order.

Next we show that the collection of potential candidates $\alpha$ contains all valid candidates $\gamma$ of size $l+1$. All subbags of $\gamma$ are frequent, and in particular
those two subbags $\delta_1$ and $\delta_2$ of size $l$ that contain all but the last and the second last items of $\gamma$, respectively. Since $\delta_1$ and $\delta_2$ are in $F_l$ and they have $l - 1$ items in common, they are in the same block. At some time in the algorithm we have $F_l[i] = \delta_1$ and $F_l[j] = \delta_2$, and $\gamma$ is considered as a potential candidate in the algorithm.

We need to show that no false candidates are output. A bag of size $l + 1$ has $l + 1$ subbags $\beta$ of size $l$, and for all of these we make sure that they are in $F_l$. We obtain all these subbags by leaving out one of the items in $\alpha$ at a time (line 10). Note that the two subbags that were used for constructing $\alpha$ do not need to be checked again. Only if all subbags of size $l - 1$ are in $F_l$, $\alpha$ correctly output as a candidate.

Algorithm 1.1.15 can be easily modified to generate candidate ordered multisets, i.e. sequences. Now the items in the array representing the sequence are in the order imposed by a total order $\leq$. For instance, a sequence $\beta$ with items $C, A, F,$ and $C$, in that order, is represented as an array $\beta$ with $\beta[1] = C$, $\beta[2] = A$, $\beta[3] = F$, and $\beta[4] = C$. Collections of sequences are still stored as lexicographically sorted arrays. The only change to the algorithm is to replace line 5.

Theorem 1.1.17 With the line

5. \textbf{for} $(j := \text{first index of the block}; F_l[j] \text{ in the same block}; j := j + 1)$ \textbf{do}

Algorithm 1.1.15 works correctly for sequences.

\textbf{Proof} The proof is similar to the proof for Theorem 1.1.16; now, however, $i$ and $j$ iterate over all ordered pairs of sequences in each block. The (potential) candidates are ordered sequences of items, not sorted arrays as before, but the candidate collection is still constructed in lexicographical order. The same arguments for the correctness of the candidate collection hold.

There are further options with the algorithm. If we are interested in sets rather than multisets, i.e., items should not appear more than once, simply add one line.

Theorem 1.1.18 With the line

5b. if $j = i$ \textbf{then} continue with the next $j$ at line 5;

inserted after line 5, Algorithm 1.1.15 works correctly for sets (or ordered sets with at most one occurrence of any item with the change of Theorem 1.1.17).

\textbf{Proof} Clearly, the effect of the inserted line is that some candidates are not generated. Consider now those excluded candidates. First note that only candidates $\alpha$ that contain some item at least twice are excluded. Either a
candidate is excluded explicitly because \( i = j \), or it is not generated because some of its subset is not in \( F_l \). If \( \alpha \) is excluded explicitly, then it contains the item \( \alpha[l] = \alpha[l + 1] \) twice. If, on the other hand, some tested subset \( \beta \) is not in the collection \( F_l \), then there must be a subset \( \gamma \preceq \beta \) that has been excluded explicitly. Then \( \alpha \) contains twice the item \( \gamma[|\gamma|] \).

Now note that no set \( \alpha \) with at least two occurrences of an item is generated. Let \( A \) be an item that occurs at least twice in \( \alpha \). Then for the item \( \gamma \) of size 2 such that \( \gamma[1] = A \) and \( \gamma[2] = A \) we have \( \gamma \preceq \alpha \), and thus \( \alpha \) cannot be a candidate unless \( \gamma \) is frequent. However, \( \gamma \) has been excluded explicitly by the inserted line in an earlier iteration, and thus \( \alpha \) is not a candidate.

The time complexity of Algorithm 1.1.15 is polynomial in the size of the collection of frequent sequences and it is independent of the length of the item sequence.

**Theorem 1.1.19** Algorithm 1.1.15 (with any of the above variations) has time complexity \( O(l^2 |F_l|^2 \log |F_l|) \).

**Proof** The initialization (line 3) takes time \( O(|F_l|) \). The outer loop (line 4) is iterated \( O(|F_l|) \) times and the inner loop (line 5) \( O(|F_l|) \) times. Within the loops, a potential candidate (lines 8 and 9) and \( l - 1 \) subcandidates (lines 10 to 13) are built in time \( O(l + 1 + (l - 1)l) = O(l^2) \). More importantly, the \( l - 1 \) subsets need to be searched for in the collection \( F_l \) (line 14). Since \( F_l \) is sorted, each subcandidate can be located by binary search in time \( O(l \log |F_l|) \). The total time complexity is thus \( O(|F_l| + |F_l||F_l|(l^2 + (l - 1)l \log |F_l|)) = O(l^2 |F_l|^2 \log |F_l|) \).

In practical situations the time complexity is likely to be close to \( O(l^2 |F_l| \log |F_l|) \), since the blocks are typically small.

### 1.2 The discovery task

We start by defining the knowledge discovery setting we consider in this chapter. Given a set of patterns, i.e., a class of expressions about databases, and a predicate to evaluate whether a database satisfies a pattern, the task is to determine which patterns are satisfied by a given database.

**Definition 1.2.1** Assume that \( \mathcal{P} \) is a set and \( q \) is a predicate \( q : \mathcal{P} \times \{r \mid r \text{ is a database} \} \rightarrow \{\text{true, false}\} \). Elements of \( \mathcal{P} \) are called patterns and \( q \) is a selection criterion over \( \mathcal{P} \). Given a pattern \( \varphi \) in \( \mathcal{P} \) and a database \( r \), we say that \( \varphi \) is selected if \( q(\varphi, r) \) is true. Since the selection criterion is often based on the frequency of the pattern, we use the term frequent as a synonym for “selected”. Given a database \( r \), the theory \( T(\mathcal{P}, r, q) \) of \( r \) with respect to \( \mathcal{P} \) and \( q \) is \( T(\mathcal{P}, r, q) = \{\varphi \in \mathcal{P} \mid q(\varphi, r) \text{ is true}\} \).
Example 1.2.2 The problem of finding all frequent item sets can be described as a task of discovering frequent patterns in a straightforward way. Given a set $R$, a binary database $r$ over $R$, and a frequency threshold $min\_fr$, the set $P$ of patterns consists of all item sets, i.e., $P = \{X \mid X \subseteq R\}$, and for the selection criterion we have $q(\varphi, r) = true$ if and only if $fr(\varphi, r) \geq min\_fr$.

Note that we do not specify any satisfaction relation for the patterns of $P$ in $r$: this task is taken care of by the selection criterion $q$. For some applications, “$q(\varphi, r)$ is true” could mean that $\varphi$ occurs often enough in $r$, that $\varphi$ is true or almost true in $r$, or that $\varphi$ defines, in some way, an interesting property or subgroup of $r$. Obviously, the task of determining the theory of $r$ is not tractable for arbitrary sets $P$ and predicates $q$. If, for instance, $P$ is infinite and $q(\varphi, r)$ is true for infinitely many patterns, an explicit representation of $T(P, r, q)$ cannot be computed.

In the discovery tasks considered here the aim is to find all patterns that are selected by a relatively simple criterion—such as exceeding a frequency threshold—in order to efficiently identify a space of potentially interesting patterns. Other criteria can then be used for further pruning and processing of the patterns. Consider as an example the discovery of association rules: first frequent sets are discovered, then all rules with sufficient frequency are generated, and a confidence threshold is used to further prune the rules.

The task of discovering frequent sets has two noteworthy properties. First, all frequent sets are needed for the generation of association rules. It is not sufficient to know just the largest frequent sets, although they determine the collection of all frequent sets. The second important property is that the selection criterion, i.e., frequency, is monotone decreasing with respect to expansion of the set. We consider only the situation where the predicate $q$ is monotone with respect to a given partial order on the patterns.

Definition 1.2.3 Let $P$ be a set of patterns, $q$ a selection criterion over $P$, and $\leq$ a partial order on the patterns in $P$. If for all databases $r$ and patterns $\varphi, \theta \in P$ we have that $q(\varphi, r)$ and $\theta \leq \varphi$ imply $q(\theta, r)$, then $\leq$ is a specialization relation on $P$ with respect to $q$. If we have $\theta \leq \varphi$, then $\varphi$ is said to be more special than $\theta$ and $\theta$ to be more general than $\varphi$. If $\theta \leq \varphi$ and not $\varphi \leq \theta$ we write $\theta \prec \varphi$.

Example 1.2.4 The set inclusion relation $\subseteq$ is a specialization relation for frequent sets. For instance, if the set $\{A, B, C\}$ is frequent, then its subset $\{A, C\}$ must also be frequent.

In more general, given two item sets $X, Y \subseteq R$, the set $X$ is more general, $X \preceq Y$, if and only if $X \subseteq Y$. That is, $X \preceq Y$ implies that if the more specific set $Y$ is frequent then the more general set $X$ is frequent, too.
For practical purposes the specialization relation has to be computable, i.e., given patterns \( \varphi \) and \( \theta \) in \( \mathcal{P} \), it must be possible to determine whether \( \varphi \preceq \theta \). Typically, the specialization relation \( \preceq \) is a restriction of the converse of the semantic implication relation: if \( \theta \preceq \varphi \), then \( \varphi \) implies \( \theta \). If the predicate \( q \) is defined in terms of, e.g., statistical significance, then the semantic implication relation is not a specialization relation with respect to \( q \): a pattern can be statistically significant even when a more general pattern is not. Recall that the predicate \( q \) is not meant to be the only way of identifying the interesting patterns; a threshold for the statistical significance can be used to further prune patterns found using \( q \).

1.3 The generic levelwise algorithm

In this section we present an algorithm for the task of discovering all frequent patterns in the case where there exists a computable specialization relation between patterns. We use the following notation for the relative speciality of patterns.

**Definition 1.3.1** Given a specialization relation \( \preceq \) on patterns in \( \mathcal{P} \), the **level** of a pattern \( \varphi \) in \( \mathcal{P} \), denoted \( \text{level}(\varphi) \), is 1 if there is no \( \theta \) in \( \mathcal{P} \) for which \( \theta \prec \varphi \). Otherwise \( \text{level}(\varphi) \) is \( 1 + L \), where \( L \) is the maximum level of patterns \( \theta \) in \( \mathcal{P} \) for which \( \theta \prec \varphi \). The collection of frequent patterns of level \( l \) is denoted by \( \mathcal{T}_l(\mathcal{P}, r, q) = \{ \varphi \in \mathcal{T}(\mathcal{P}, r, q) \mid \text{level}(\varphi) = l \} \).

Algorithm 1.3.2, analogical to Algorithm 1.1.7, finds all frequent patterns. It works in a levelwise or breadth-first manner, starting with the set \( \mathcal{C}_1 \) of the most general patterns, and then generating and evaluating more and more special candidate patterns. The algorithm prunes those patterns that cannot be frequent given all the frequent patterns obtained in earlier iterations.

The algorithm is generic: details depending on the specific types of patterns and data are left open, and instances of the algorithm must specify these. The levelwise algorithm aims at minimizing the number of evaluations of \( q \) on line 5. As with the frequent set discovery algorithm, the computation to determine the candidate collection does not involve the database at all.

**Theorem 1.3.3** Algorithm 1.3.2 works correctly.

**Proof** We show by induction on \( l \) that \( \mathcal{T}_l(\mathcal{P}, r, q) \) is computed correctly for all \( l \). For \( l = 1 \), the collection \( \mathcal{C}_1 \) contains all patterns of level one (line 1), and collection \( \mathcal{T}_l(\mathcal{P}, r, q) \) is then correctly computed (line 5).

For \( l > 1 \), assume the collections \( \mathcal{T}_i(\mathcal{P}, r, q) \) have been computed correctly for all \( i < l \). Note first that \( \mathcal{T}_i(\mathcal{P}, r, q) \subseteq \mathcal{C}_i \). Namely, consider any pattern \( \varphi \) in \( \mathcal{T}_i(\mathcal{P}, r, q) \): we have \( \text{level}(\varphi) = l \) and thus for all patterns \( \theta \prec \varphi \) we have \( \text{level}(\theta) < l \). Since \( \mathcal{T}_i(\mathcal{P}, r, q) \) has been computed for each \( i < l \), each \( \theta \prec \varphi \)
Algorithm 1.3.2

**Input:** A database schema $R$, a database $r$ over $R$, a finite set $P$ of patterns, a computable selection criterion $q$ over $P$, and a computable specialization relation $\preceq$ on $P$.

**Output:** The set $T(P,r,q)$ of all frequent patterns.

**Method:**
1. compute $C_1 := \{ \varphi \in P \mid \text{level}(\varphi) = 1 \}$;
2. $l := 1$;
3. while $C_l \neq \emptyset$ do
   4. // Database pass:
      5. compute $T_l(P,r,q) := \{ \varphi \in C_l \mid q(\varphi, r) \}$;
      6. $l := l + 1$;
   7. // Candidate generation:
      8. compute $C_l := \{ \varphi \in P \mid \text{level}(\varphi) = l \text{ and } \theta \in T_{\text{level}(\theta)}(P,r,q) \text{ for all } \theta \in P \text{ such that } \theta \prec \varphi \}$;
9. for all $l$ do output $T_l(P,r,q)$;

is correctly in $T_{\text{level}(\varphi)}(P,r,q)$, and so $\varphi$ is put into $C_l$ (line 8). The collection $T_l(P,r,q)$ is then computed correctly on line 5.

Finally note that for every $\varphi \in T(P,r,q)$ there is an iteration where the variable $l$ has value $\text{level}(\varphi)$. By the definition of level, there are more general patterns $\theta \prec \varphi$ on every level less than $\text{level}(\varphi)$, and since $\preceq$ is a specialization relation they are all frequent, so the ending condition $C_l = \emptyset$ is not true with $l \leq \text{level}(\varphi)$.

The input specification of the algorithm states that the set $P$ is finite. Actually, it does not always need to be finite: the algorithm works correctly as long as the number of candidate patterns is finite. There are some desirable properties for the specialization relation $\preceq$. An efficient method for accessing the more specific and more general patterns on neighboring levels is useful, or otherwise finding the collection of valid candidates may be expensive.