Dataset Filtering Techniques in Constraint-Based Frequent Pattern Mining

Marek Wojciechowski and Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Marek.Wojciechowski@cs.put.poznan.pl
Maciej.Zakrzewicz@cs.put.poznan.pl

Abstract. Many data mining techniques consist in discovering patterns frequently occurring in the source dataset. Typically, the goal is to discover all the patterns whose frequency in the dataset exceeds a user-specified threshold. However, very often users want to restrict the set of patterns to be discovered by adding extra constraints on the structure of patterns. Data mining systems should be able to exploit such constraints to speed-up the mining process. In this paper, we focus on improving the efficiency of constraint-based frequent pattern mining by using dataset filtering techniques. Dataset filtering conceptually transforms a given data mining task into an equivalent one operating on a smaller dataset. We present transformation rules for various classes of patterns: itemsets, association rules, and sequential patterns, and discuss implementation issues regarding integration of dataset filtering with well-known pattern discovery algorithms.

1 Introduction

Many data mining techniques consist in discovering patterns frequently occurring in the source dataset. The two most prominent classes of patterns are frequent itemsets [1] and sequential patterns [3]. Informally, frequent itemsets are subsets frequently occurring in a collection of sets of items, and sequential patterns are the most frequently occurring subsequences in sequences of sets of items. Frequent itemsets themselves provide useful information on the correlations between items in the database. Nevertheless, discovered frequent itemsets are very often treated only as the basis for association rule generation [1]. Frequent itemsets, association rules, and sequential patterns were introduced in the context of market basket analysis but their applications also include fraud detection, analysis of telecommunication systems, medical records, web server logs, etc.

Typically, in frequent pattern mining the goal is to discover all patterns whose frequency (called support) in the source dataset exceeds a user-specified threshold. If frequent patterns discovered are to be used to generate rules, the minimum accepted confidence of a rule also has to be specified. Additionally, in sequential pattern discovery several time constraints have been proposed to be

used when deciding if a given pattern is contained in a given sequence from the source dataset [10]. Since it was shown that derivation of rules from patterns is a straightforward task, the research focused mainly on improving the efficiency of algorithms discovering all patterns whose support in the source dataset exceeds a user-specified threshold.

However, it has been observed that users are very often interested in patterns that satisfy more sophisticated criteria, for example concerning size, length, or contents of patterns. Data mining tasks involving the specification of various types of constraints can be regarded as data mining queries [7]. It is obvious that additional constraints regarding the structure of patterns can be verified in a post-processing step, after all patterns exceeding a given minimum support threshold have been discovered. Nevertheless, such a solution cannot be considered satisfactory since users providing advanced pattern selection criteria may expect that the data mining system will exploit them in the mining process to improve performance. In other words, the system should concentrate on patterns that are interesting from the user's point of view, rather than waste time on discovering patterns the user has not asked for [5].

We claim that techniques applicable to constraint-driven pattern discovery can be classified into the following groups:

1. post-processing (filtering out patterns that do not satisfy user-specified pattern constraints after the actual discovery process);
2. pattern filtering (integration of pattern constraints into the actual mining process in order to generate only patterns satisfying the constraints);
3. dataset filtering (restricting the source dataset to objects that can possibly contain patterns that satisfy pattern constraints).

As the post-processing solution was considered unsatisfactory, the researchers focused on incorporating pattern constraints into classic pattern discovery algorithms. This led to the introduction of numerous constraint-based pattern discovery methods (e.g. [4][11]), all of which fall into the second group according to our classification. It should be noted that some of the methods from this class generate a superset of the collection of patterns requested by user, which means that a post-processing phase might still be required. Nevertheless, all these methods use pattern constraints to reduce the number of generated patterns, leading to a smaller set of patterns to be verified in the post-processing step than in case of classic algorithms.

In this paper we discuss an alternative approach to constraint-based pattern discovery, called *dataset filtering*. Dataset filtering is based on the observation that for some classes of pattern constraints, patterns satisfying them can only be contained in objects satisfying the same or similar constraints. The key issue in dataset filtering is derivation of filtering predicates to be applied to the source dataset from pattern constraints specified by a user. Dataset filtering is a general technique applicable to various types of patterns but for a particular class of patterns and a given constraint model distinct derivation rules have to provided. We focus on two types of patterns: frequent itemsets and sequential patterns. We also discuss extensions required to handle association rules. We

assume a relatively simple constraint model with pattern constraints referring to the size or length of patterns or to the presence of a certain subset or subsequence. Nevertheless, we believe that types of constraints we consider are the most intuitive and useful in practice.

Conceptually, dataset filtering transforms a given data mining task into an equivalent one operating on a smaller dataset. Thus, it can be integrated with any pattern discovery algorithm, possibly exploiting other constraint-based pattern discovery techniques. In this paper we focus on the integration of dataset filtering techniques within the *Apriori* framework. We discuss possible implementations of dataset filtering within *Apriori*-like algorithms, evaluating their strengths and weaknesses.

2 Background and Related Work

2.1 Frequent Itemsets and Association Rules

Let $L = l_1, l_2, ..., l_m$ be a set of literals, called *items*. An *itemset* X is a non-empty set of items ($X \subseteq L$). The *size* of an itemset X is the number of items in X. Let D be a set of variable size itemsets, where each itemset T in D has a unique identifier and is called a *transaction*. We say that a transaction T *contains* an item $x \in L$ if x is in T. We say that a transaction T *contains* an itemset $X \subseteq L$ if T contains every item in the set X. The *support* of the itemset X is the percentage of transactions in D that contain X. The problem of mining frequent itemsets in D consists in discovering all itemsets whose support is above a user-defined support threshold.

An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subseteq L$, $Y \subseteq L$, $X \cap Y = \emptyset$. We call X the *body* of a rule and Y the *head* of a rule. The *support* of the rule $X \rightarrow Y$ in D is the support of the itemset $X \cup Y$. The *confidence* of the rule $X \rightarrow Y$ is the percentage of transactions in D containing X that also contain Y. The problem of mining association rules in D consists in discovering all association rules whose support and confidence are above user-defined minimum support and minimum confidence thresholds.

2.2 Sequential Patterns

Let $L = l_1, l_2, ..., l_m$ be a set of literals called *items*. An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets and is denoted as $< X_1 X_2 ... X_n >$, where X_i is an itemset ($X_i \subseteq L$). X_i is called an *element* of the sequence. The *size* of a sequence is the number of items in the sequence. The *length* of a sequence is the number of elements in the sequence.

We say that a sequence $X =< X_1 X_2 ... X_n >$ is a *subsequence* of a sequence $Y =< Y_1 Y_2 ... Y_m >$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, ..., X_n \subseteq Y_{i_n}$. We call $< Y_{i_1} Y_{i_2} ... Y_{i_n} >$ an *occurrence* of X in Y.

Given a sequence $Y =< Y_1 Y_2 ... Y_m >$ and a subsequence X, X is a *contiguous* subsequence of Y if any of the following conditions hold: 1) X is derived from Y

by dropping an item from either Y_1 or Y_m. 2) X is derived from Y by dropping an item from an element Y_i which has at least 2 items. 3) X is a contiguous subsequence of X', and X' is a contiguous subsequence of Y.

Let D be a set of variable length sequences (called *data-sequences*), where for each sequence $S =< S_1 S_2 ... S_n >$, a timestamp is associated with each S_i. With no time constraints we say that a sequence X is *contained* in a data-sequence S if X is a subsequence of S. We consider the following user-specified time constraints while looking for occurrences of a given sequence in a given data-sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), and time window that allows a group of consecutive elements of a data-sequence to be merged and treated as a single element as long as their timestamps are within the user-specified *window-size*.

The *support* of a sequence $< X_1 X_2 ... X_n >$ in D is the fraction of data-sequences in D that contain the sequence. A *sequential pattern* (also called a *frequent sequence*) is a sequence whose support in D is above the user-specified minimum support threshold.

2.3 Review of Classic Pattern Mining Algorithms

The majority of frequent itemset and sequential pattern discovery algorithms fall into two classes: *Apriori*-like methods and *pattern-growth* methods. The first group of methods is based on the *Apriori* algorithm for frequent itemset mining [2]. *Apriori* relies on the property (called *Apriori* property) that an itemset can only be frequent if all of its subsets are frequent. It leads to a level-wise procedure. First, all possible 1-itemsets (itemsets containing 1 item) are counted in the database to determine frequent 1-itemsets. Then, frequent 1-itemsets are combined to form potentially frequent 2-itemsets, called candidate 2-itemsets. Candidate 2-itemsets are counted in the database to determine frequent 2-itemsets. The procedure is continued until in a certain iteration none of the candidates turns out to be frequent or the set of generated candidates is empty. Several extensions were added to improve the performance of *Apriori* (e.g. by reducing the number of database passes). The algorithm also served as a basis for algorithms discovering other types of patterns including sequential patterns.

The most prominent sequential pattern discovery algorithm from the Apriori family is *GSP*, introduced in [10]. *GSP* exploits a variation of the Apriori property: all contiguous subsequences of a frequent sequence also have to be frequent. In each iteration, candidate sequences, are generated from the frequent sequences found in the previous pass, and then verified in a database scan. It should be noted that *GSP* (and its variants) is the only sequential pattern discovery algorithm capable of handling time constraints (max-gap, min-gap, and window-size).

Recently, a new family of pattern discovery algorithms, called pattern-growth methods (see [6] for a review), has been developed for discovery of frequent patterns. The methods project databases based on the currently discovered frequent patterns and grow such patterns to longer ones in corresponding pro-

jected databases. Pattern-growth methods are supposed to perform better than *Apriori*-like algorithms in case of low minimum support thresholds. Nevertheless, practical studies [12] show that for real datasets *Apriori* (or its variants) might still be a more efficient solution. Moreover, in the context of sequential patterns pattern-growth methods still do not offer full functionality of *GSP*, as they do not handle time constraints.

2.4 Previous Work on Constraint-Based Pattern Mining

As we mentioned earlier the research on constraint-based pattern mining focused on incorporating pattern constraints into classic pattern discovery algorithms, especially in the context of frequent itemsets and association rules. Pattern constraints in frequent itemset and association rule mining were first discussed in [11]. Constraints considered there had a form of a Boolean expression in the disjunctive normal form built from elementary predicates requiring that a certain item is or is not present. The algorithms presented were *Apriori* variants using sophisticated candidate generation techniques. Rule constraints were handled by transforming them into itemset constraints. It was observed that after discovering all itemsets that can be used to generate the rules of interest, one extra scan of the dataset is required to count the supports of some subsets, required to evaluate confidences of some rules, and not known since the subsets did not satisfy the derived itemset constraints.

In [8], two interesting classes of itemset constraints were introduced: anti-monotonicity and succinctness, and methods of handling constraints belonging to these classes within the *Apriori* framework were presented. The methods for succinct constraints again consisted in modifying the candidate generation procedure. For anti-monotone constraints it was observed that in fact almost no changes to *Apriori* are required to handle them. A constraint is anti-monotone if the fact that an itemset satisfies it, implies that all of its subsets have to satisfy the constraint too. The minimum support threshold is an example of an anti-monotone constraint, and any extra constraints of that class can be used together with it in candidate pruning.

In [9], constraint-based discovery of frequent itemsets was analyzed in the context of pattern-growth methodology. In the paper, further classes of constraints were introduced, some of which could not be incorporated into the *Apriori* framework.

On the other hand, very little work concerning constraint-driven sequential pattern discovery has been done so far. In fact, only the algorithms from the *SPIRIT* family [4] exploit pattern structure constraints in order to improve performance. These algorithms can be seen as extensions of *GSP* using advanced candidate generation and pruning techniques. In the *SPIRIT* framework, pattern constraints are specified as regular expressions, which is an especially convenient method if a user wants to significantly restrict the structure of patterns to be discovered. It has been shown experimentally that pushing regular expression constraints deep into the mining process can reduce processing time by more than an order of magnitude.

3 Dataset Filtering in Constraint-Based Pattern Mining

In constraint-based pattern mining, we identify the following classes of constraints: database constraints, statistical constraints, pattern constraints, and time constraints. Database constraints are used to specify the source dataset. Statistical constraints are used to specify thresholds for the support and confidence measures. Pattern constraints specify which of the frequent patterns are interesting and should be returned by the query. Finally, time constraints used in sequential pattern mining influence the process of checking whether a given data-sequence contains a given pattern.

Basic formulations of pattern discovery problems do not consider pattern constraints. We model pattern constraints as a conjunction of basic Boolean predicates referring to pattern size or length (*size constraints*) or regarding the presence of a certain subset or subsequence (*item constraints*).

It should be noted that not all pattern predicates support the dataset filtering paradigm. For example, if a user is looking for frequent itemsets whose size exceeds a given threshold, it is rather obvious that such itemsets can be contained only in transactions whose size exceeds the same threshold. Thus, smaller transactions can be excluded from the mining process, which should lead to performance gains. On the other hand, if a user is interested in itemsets having the size not exceeding a given threshold, dataset filtering is not applicable as such itemsets can be contained in any transaction.

For each of the pattern predicates present in pattern constraints supporting dataset filtering, the corresponding predicate on transactions or data-sequences has to be derived. The resulting dataset filtering predicate is formed as a conjunction of those derived predicates (recall that we consider pattern constraints having the form of a conjunction of pattern predicates). The filtering predicate is then used to discard objects in the source dataset that cannot contain the patterns of interest. Below we identify pattern predicates in case of which dataset filtering is applicable in the context of frequent itemsets, association rules and sequential patterns. For each of the identified pattern predicates we provide a corresponding predicate on source objects, to be used in dataset filtering. As we will show, dataset filtering is rather straightforward in frequent itemset discovery but becomes more complicated in sequential pattern mining thanks to time constraints.

3.1 Dataset Filtering in Frequent Itemset Discovery

Let us consider the following predicate types that can appear in pattern constraints of a frequent itemset query:

- $\rho(\mathbf{SG}, \alpha, itemset)$ - true if itemset size is greater than α, false otherwise;
- $\rho(\mathbf{C}, \gamma, itemset)$ - true if γ is a subset of the itemset, false otherwise;

Theorem 1. *Itemsets of size greater than k cannot be contained in a transaction whose size is not greater than k.*

Proof. The proof is obvious since an itemset is contained in a transaction if it is a subset of the set of items present in the transaction.

Theorem 2. *Frequent itemsets, to be returned by a data mining query, containing a given subset can be supported only by transactions containing that subset.*

Proof. An itemset is contained in a transaction if it is a subset of the set of items present in the transaction. The transitivity of set inclusion relationship implies that if an itemset having a given subset is contained in a given transaction, the transaction also has to contain the subset.

According to the above theorems the following dataset filtering predicates are applicable in frequent itemset mining:

- $\tau(\mathbf{SG}, \alpha, transaction)$ - true if the size of the transaction is greater than α, false otherwise;
- $\tau(\mathbf{C}, \gamma, transaction)$ - true if the transaction contains the set γ, false otherwise;

For a frequent itemset query with pattern constraints, an appropriate dataset filtering predicate is derived in the following way: For each of the pattern predicates from the left column of Table 1 present in the query, the corresponding transaction predicate from the right table column is added to the dataset filtering predicate.

Table 1. Derivation rules for frequent itemset mining

Pattern predicate	Transaction predicate
$\rho(\mathbf{SG}, \alpha, itemset)$	$\tau(\mathbf{SG}, \alpha, transaction)$
$\rho(\mathbf{C}, \gamma, itemset)$	$\tau(\mathbf{C}, \gamma, transaction)$

3.2 Dataset Filtering in Association Rule Discovery

Let us consider the following predicate types that can appear in pattern constraints of an association rule query:

- $\rho(\mathbf{SG}, \alpha, rule)$ - true if the number of items in the rule is greater than α, false otherwise;
- $\rho(\mathbf{C}, \gamma, rule)$ - true if all items from γ are present in the rule, false otherwise;
- $\rho(\mathbf{SG}, \alpha, body(rule))$ - true if the size of the rule's body is greater than α, false otherwise;
- $\rho(\mathbf{C}, \gamma, body(rule))$ - true if γ is a subset of the rule's body, false otherwise;
- $\rho(\mathbf{SE}, \alpha, body(rule))$ - true if the size of the rule's body is α, false otherwise;

– $\rho(\mathbf{E}, \gamma, body(rule))$ - true if the rule's body is equal to γ, false otherwise;
– $\rho(\mathbf{SG}, \alpha, head(rule))$ - true if the size of the rule's head is greater than α, false otherwise;
– $\rho(\mathbf{C}, \gamma, head(rule))$ - true if γ is a subset of the rule's head, false otherwise;
– $\rho(\mathbf{SE}, \alpha, head(rule))$ - true if the size of the rule's head is α, false otherwise;
– $\rho(\mathbf{E}, \gamma, head(rule))$ - true if the rule's head is equal to γ, false otherwise;

We observe that the rule predicates can be directly transformed into predicates on itemsets that can be used to generate the rules having the desired properties. All the items required in the rule, rule's body, or rule's head have to appear in the frequent itemsets from which the rules are to be generated. The size threshold on rules implies the same threshold on itemsets. However, if a given size of the rule's body or head is required, then the itemset must have at least one more item (neither the head nor the body can be empty). Table 2 presents a corresponding itemset predicate for each of the rule predicates. (In fact, if the predicates for both head and body are present, and at least one of them is a size predicate, a more restrictive itemset size predicate can be derived. We omit the details for the sake of simplicity.)

Table 2. Rule predicates and their corresponding itemset predicates

Rule predicate	Itemset predicate
$\rho(\mathbf{SG}, \alpha, rule)$	$\rho(\mathbf{SG}, \alpha, itemset)$
$\rho(\mathbf{C}, \gamma, rule)$	$\rho(\mathbf{C}, \gamma, itemset)$
$\rho(\mathbf{SG}, \alpha, body(rule))$	$\rho(\mathbf{SG}, \alpha + 1, itemset)$
$\rho(\mathbf{C}, \gamma, body(rule))$	$\rho(\mathbf{C}, \gamma, itemset)$
$\rho(\mathbf{SE}, \alpha, body(rule))$	$\rho(\mathbf{SG}, \alpha, itemset)$
$\rho(\mathbf{E}, \gamma, body(rule))$	$\rho(\mathbf{C}, \gamma, itemset)$
$\rho(\mathbf{SG}, \alpha, head(rule))$	$\rho(\mathbf{SG}, \alpha + 1, itemset)$
$\rho(\mathbf{C}, \gamma, head(rule))$	$\rho(\mathbf{C}, \gamma, itemset)$
$\rho(\mathbf{SE}, \alpha, head(rule))$	$\rho(\mathbf{SG}, \alpha, itemset)$
$\rho(\mathbf{E}, \gamma, head(rule))$	$\rho(\mathbf{C}, \gamma, itemset)$

As the rule predicates implicate itemset predicates, there is no need to provide separate derivation rules for dataset filtering predicates to be used in association rule mining. It should be noted that while the collection of discovered frequent itemsets supporting the constraints derived from the rule constraints is sufficient to generate all the required rules, it may not contain all the itemsets needed to evaluate confidences of the rules as certain subsets of those itemsets may not satisfy pattern constraints. This problem is not specific to our dataset filtering techniques, and has to be solved by an extra scan of the dataset. In that extra scan, supports of itemsets whose support is needed but not known have to be counted.

3.3 Dataset Filtering in Sequential Pattern Discovery

Let us consider the following predicate types that can appear in pattern constraints of a sequential pattern query:

- π(**SG**, α, *pattern*) - true if pattern size is greater than α, false otherwise;
- π(**LG**, α, *pattern*) - true if pattern length is greater than α, false otherwise;
- π(**C**, β, *pattern*) - true if β is a subsequence of the pattern, false otherwise;
- ρ(**SG**, α, $pattern_n$) - true if the size of the n-th element of the pattern is greater than α, false otherwise;
- ρ(**C**, γ, $pattern_n$) - true if γ is a subset of the n-th element of the pattern, false otherwise;

Theorem 3. *Sequential patterns of size greater than k cannot be contained in a data-sequence whose size is not greater than k.*

Proof. The proof is obvious since an occurrence of a pattern in a sequence must consist of the same number of items as the pattern.

Theorem 4. *Sequential patterns of length greater than k, to be returned by a data mining query, can be contained only in data-sequences which contain some sequence of length $k+1$ using max-gap, min-gap, and window-size specified in the query.*

Proof. Each sequential pattern of length greater than k has at least one contiguous subsequence of length $k+1$. If a data-sequence contains some sequence, it contains every contiguous subsequence of that sequence. Thus, if a data-sequence contains some sequence of length greater than k, it contains at least one sequence of length $k+1$.

Theorem 5. *Sequential patterns, to be returned by a data mining query, containing a given sequence can be contained only in data-sequences containing that sequence using min-gap and window-size specified in the query, and max-gap of $+\infty$.*

Proof. If a data-sequence contains some sequence using certain values of max-gap, min-gap, and window-size, it also contains every contiguous subsequence of the sequence, using the same time constraints. If max-gap is set to $+\infty$, a data-sequence containing some sequence contains all its subsequences.

Theorem 6. *Sequential patterns, to be returned by a data mining query, whose n-th element has the size greater than k can be contained only in data-sequences which contain some 1-element sequence of size $k+1$ using window-size specified in the query.*

Proof. Each 1-element subsequence of any sequence is its contiguous subsequence (from the definition of a contiguous subsequence). If any element of a sequence has the size greater than k, the sequence has at least one 1-element contiguous subsequence of size $k+1$. If a data-sequence contains some sequence, it contains every contiguous subsequence of that sequence. Thus, if a data-sequence contains some sequence whose n-th element has the size greater than k, it has to contain some 1-element sequence of size $k+1$.

Theorem 7. *Sequential patterns, to be returned by a data mining query, whose n-th element contains a given set can be contained only in data-sequences which contain a 1-element sequence having the set as the only element, using time constraints specified in the query.*

Proof. Each 1-element subsequence of any sequence is its contiguous subsequence (from the definition of a contiguous subsequence). If any element of a sequence contains a given set, a 1-element sequence formed by the set is a contiguous subsequence of the sequence. If a data-sequence contains some sequence, it contains every contiguous subsequence of that sequence. Thus, if a data-sequence contains some sequence whose n-th element contains a given set, it has to contain a 1-element sequence having the set as the only element.

According to the above theorems the following dataset filtering predicates are applicable in sequential pattern mining:

- $\sigma(\mathbf{SG}, \alpha, \mathit{sequence})$ - true if the size of the data-sequence is greater than α, false otherwise;
- $\sigma(\mathbf{C}, \beta, \mathit{sequence}, \mathit{maxgap}, \mathit{mingap}, \mathit{window})$ - true if the data-sequence contains the sequence forming the pattern β using given time constraints, false otherwise;
- $\sigma(\mathbf{CS}, \alpha, \mathit{sequence}, \mathit{window})$ - true if there exists a 1-element sequence of size α that is contained in the sequence with respect to the window-size constraint, false otherwise;
- $\sigma(\mathbf{CL}, \alpha, \mathit{sequence}, \mathit{maxgap}, \mathit{mingap}, \mathit{window})$ - true if there exists a sequence of length α that is contained in the sequence with respect to the max-gap, min-gap, and window-size constraints, false otherwise.

For a sequential pattern query with pattern constraints, an appropriate dataset filtering predicate is derived in the following way: For each of the pattern predicates from the left column of Table 3 present in the query, the corresponding data-sequence predicate from the right table column is added to the dataset filtering predicate.

Table 3. Derivation rules for sequential pattern mining

Pattern predicate	Data-sequence predicate
$\pi(\mathbf{SG}, \alpha, \mathit{pattern})$	$\sigma(\mathbf{SG}, \alpha, \mathit{sequence})$
$\pi(\mathbf{LG}, \alpha, \mathit{pattern})$	$\sigma(\mathbf{CL}, \alpha+1, \mathit{sequence}, \mathit{max}, \mathit{min}, \mathit{win})$
$\pi(\mathbf{C}, \beta, \mathit{pattern})$	$\sigma(\mathbf{C}, \beta, \mathit{sequence}, +\infty, \mathit{min}, \mathit{win})$
$\rho(\mathbf{SG}, \alpha, \mathit{pattern}_n)$	$\sigma(\mathbf{CS}, \alpha+1, \mathit{sequence}, \mathit{win})$
$\rho(\mathbf{C}, \gamma, \mathit{pattern}_n)$	$\sigma(\mathbf{C}, <\gamma>, \mathit{sequence}, \mathit{max}, \mathit{min}, \mathit{win})$

In the above table, $<\gamma>$ denotes a 1-element sequence having the set γ as its only element, while *max*, *min*, and *win* represent values of max-gap, min-gap, and window-size time constraints respectively.

4 Implementation Issues Regarding Dataset Filtering

If any pattern predicates supporting dataset filtering are present in the pattern query specifying a certain pattern discovery task, the query can be transformed into a query representing a discovery task on a potentially smaller dataset in the following way. Firstly, database constraints of the query have to be extended by adding the appropriate dataset filtering predicate to them (the filtering predicate is derived according to the rules presented in the previous section). Secondly, the minimum support threshold has to be adjusted to the size of the filtered database. This step is necessary because the support of a pattern is expressed as the percentage of objects (transactions or data-sequences) containing the pattern. The theorems proved in the previous section guarantee that the number of objects containing a given pattern in the original and filtered dataset will be the same as long as the pattern satisfies pattern constraints. Thus, we have the following relationship between the support of a pattern p (satisfying pattern constraints) in the original and filtered datasets: $sup_F(p) =\mid D \mid *sup(p)/ \mid D_F \mid$, where $sup_F(p)$ and $sup(p)$ denote the support of the pattern p in the filtered and original dataset respectively, and $\mid D_F \mid$ and $\mid D \mid$ denote the number of objects in the filtered and original dataset respectively. After the patterns frequent in the filtered dataset have been discovered, their support has to be normalized with respect to the number of objects in the original dataset according to the above formula (the user specifies the support threshold as the percentage of objects in the original dataset, and expects that the supports of discovered patterns will be expressed in the same way).

Dataset filtering techniques can be combined with any frequent pattern discovery algorithm since they conceptually lead to a transformed discovery task guaranteed to return the same set of patterns as the original task. The transformation of the source dataset (by filtering out objects that cannot contain patterns of interest) and adjustment of the minimum support threshold can be performed before the actual discovery process. However, in reality such explicit transformation might be impossible due to space limitations. Moreover, it may not lead to the optimal solution because of one extra scan of the dataset performed during the transformation. A natural solution to this problem is integration of dataset filtering techniques within pattern mining algorithms. Thus, dataset filtering can be performed together with other operations in the first scan of the dataset required by a given algorithm. Please note that if dataset filtering is integrated within a pattern mining algorithm, the support conversions discussed above are not necessary because the support can always refer to the number of objects in the original dataset.

Regarding integration of dataset filtering within the *Apriori* framework, there are two general implementation strategies possible. The filtered dataset can either be physically materialized on disk during the first iteration or filtering can be performed on-line in each iteration. The second option might be the only solution if materialization of the filtered dataset is not possible due to space limitations.

Below we present two algorithm frameworks following the two strategies. We do not present separate solutions for frequent itemset and sequential pattern discovery. Instead, we consider a general pattern discovery task in a collection of objects. These objects are transactions or data-sequences depending on the actual discovery task. Pattern constraints and dataset filtering predicates are also different for frequent itemsets and sequential patterns. Additionally, in the context of sequential patterns the containment relationship takes into account time constraints specified by a user. The general algorithms presented below take a collection D of objects, the minimum support threshold (and optionally time constraints), and pattern constraints as input, and return all frequent patterns in D satisfying all the provided constraints.

Algorithm 1 Apriori on materialized filtered dataset

begin
  DF = dataset filtering predicate derived from pattern constraints;
  scan D in order to:
    1) evaluate minimum number of supporting
    objects for a pattern to be called frequent (*mincount*)
    2) find L_1 (set of items contained in at
    least *mincount* objects satisfying DF);
    3) materialize the collection D' of objects from D
    satisfying DF;
  for ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do**
  begin
    C_k = apriori_gen(L_{k-1}); /* generate new candidates */
    if $C_k = \emptyset$ **then break**;
    forall objects $d \in D'$ **do**
      forall candidates $c \in C_k$ **do**
        if d contains c **then**
          c.count ++;
        end if;
    $L_k = \{ c \in C_k \mid c.\text{count} \geq mincount\}$;
  end;
  output patterns from $\cup_k L_k$ satisfying pattern constraints;
end.

Algorithm 2 Apriori with on-line dataset filtering

begin
  DF = dataset filtering predicate derived from pattern constraints;
  scan D in order to:
    1) evaluate minimum number of supporting
    objects for a pattern to be called frequent (*mincount*)
    2) find L_1 (set of items contained in at
    least *mincount* objects satisfying DF);
  for ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do**

```
    begin
        C_k = apriori_gen(L_{k-1}); /* generate new candidates */
        if C_k = ∅ then break;
        forall objects d ∈ D do
            if d satisfies DF then
                forall candidates c ∈ C_k do
                    if d contains c then
                        c.count ++;
                    end if;
            end if;
        L_k = { c ∈ C_k | c.count ≥ mincount};
    end;
    output patterns from ∪_k L_k satisfying pattern constraints;
end.
```

Both algorithms start with deriving dataset filtering predicates from pattern constraints provided by a user. In the first approach these dataset filtering predicates are used in the first scan of the source dataset to select and materialize the collection of objects on which subsequent scans will be performed. All the objects from the materialized filtered collection are then used in the candidate verification phases. In the second approach dataset filtering predicates are used in each scan of the source dataset and objects that do not satisfy them are excluded from the candidate verification process. When the discovery of sequential patterns in the filtered dataset is finished, a post-processing step filtering out patterns that do not satisfy user-specified pattern constraints is applied in both approaches. This phase is required since dataset filtering itself, regardless of the implementation details, does not guarantee that only patterns supporting pattern constraints are to be discovered. It should be noted that the support of patterns not satisfying user-specified pattern constraints, counted in the filtered dataset, can be smaller than their actual support in the original dataset, but it is not a problem since these patterns will not be returned to the user. Moreover, this is in fact a positive feature as it can reduce the number of generated candidates not leading to patterns of user's interest.

5 Performance Analysis

In order to evaluate performance gains offered by our dataset filtering techniques, we performed several experiments on synthetic datasets. We measured performance improvements thanks to dataset filtering applied to the *Apriori* algorithm for frequent itemset mining, and *GSP* for sequential patterns. As we might have expected, the performance gains depend on the selectivity of dataset filtering predicates derived from pattern constraints (expressed as the percentage of objects in the dataset satisfying dataset filtering constraints). In general the lower the selectivity factor the better, but the actual performance depends not only on the selectivity but also on data distribution in the filtered dataset. We observed

that item constraints led to much better results (reducing the processing time 2 to 5 times) than size constraints (typically reducing the processing time by less than 10%). This is due to the fact that the patterns are usually smaller in terms of size or length than source objects, and therefore even restrictive constraints on pattern size/length result in weak constraints on source objects. As a consequence, if the actual task is discovery of association rules, and the only rule constraints present are size constraints, the gains due to dataset filtering sometimes do not compensate the cost of an extra pass needed to evaluate confidences of the rules.

Regarding the implementation strategies, for item constraints implementations involving materialization of the filtered dataset were more efficient than their on-line counterparts (the filtered dataset was relatively small and the materialization cost was dominated by gains due to the smaller costs of dataset scans in candidate verification phases). However, in case of size constraints rejecting a very small number of source objects, materialization of the filtered dataset sometimes lead to longer execution times than in case of the original algorithms. The on-line dataset filtering implementations were in general more efficient than the original algorithms even for size constraints (except for a situation, unlikely in practice, when the size constraint did not reject any source objects).

In the experiments, we also observed that decreasing the minimum support threshold or relaxing time constraints worked in favor of our dataset filtering techniques, leading to bigger performance gains. This behavior can be explained by the fact that since dataset filtering reduces the cost of candidate verification phase, the more this phase contributes to the overall processing time, the more significant relative performance gains are going to be. Decreasing the minimum support threshold also led to slight performance improvement of implementations involving materialization of the filtered dataset in comparison to their on-line counterparts. As the support threshold decreases, the maximal length of a frequent patterns (and the number of iterations required by the algorithms) increases. Materialization is performed in the first iteration and reduces the cost of the second and subsequent iterations. Thus, the more iterations are required, the better the cost of materialization is compensated.

6 Concluding Remarks

We have discussed application of dataset filtering techniques to efficient frequent pattern mining in the presence of various pattern constraints. We identified the types of pattern constraints in case of which dataset filtering is applicable in the context of frequent itemsets, association rules, and sequential patterns. For each of the pattern constraint types we provided an appropriate dataset filtering predicate. Dataset filtering techniques can be applied to any frequent pattern discovery algorithm since they conceptually lead to an equivalent data mining task on a possibly smaller dataset. We focused on the implementation details concerning integration of dataset filtering techniques within the *Apriori* framework. Our experiments show that dataset filtering can result in significant performance

improvements, especially in case of pattern constrains involving the presence of a certain subset or subsequence, which we believe are the most usuful ones.

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 SIGMOD Conference (1993)
2. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th VLDB Conference (1994)
3. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th ICDE Conf. (1995)
4. Garofalakis M., Rastogi R., Shim K.: SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. Proceedings of 25th VLDB Conference (1999)
5. Han J., Lakshmanan L., Ng R.: Constraint-Based Multidimensional Data Mining. IEEE Computer, Vol. 32, No. 8 (1999)
6. Han J., Pei J.: Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. SIGKDD Explorations, December 2000 (2000)
7. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
8. Ng R., Lakshmanan L., Han J., Pang A.: Exploratory Mining and Pruning Optimizations of Constrained Association Rules. Proc. of the 1998 SIGMOD Conference (1998)
9. Pei J., Han J., Lakshmanan L.: Mining Frequent Itemsets with Convertible Constraints. Proceedings of the 17th ICDE Conference (2001)
10. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference (1996)
11. Srikant R., Vu Q., Agrawal R.: Mining Association Rules with Item Constraints. Proceedings of the 3rd KDD Conference (1997)
12. Zheng Z., Kohavi R., Mason L.: Real World Performance of Association Rule Algorithms. Proc. of the 7th KDD Conference (2001)