EFFICIENT CONSTRAINT-BASED SEQUENTIAL PATTERN MINING USING DATASET FILTERING TECHNIQUES

Tadeusz Morzy, Marek Wojciechowski, and Maciej Zakrzewicz
Poznan University of Technology, Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland

Abstract Basic formulation of the sequential pattern discovery problem assumes that the only constraint to be satisfied by discovered patterns is the minimum support 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 discuss efficient constraint-based sequential pattern mining using dataset filtering techniques. We show how to transform a given data mining task into an equivalent one operating on a smaller dataset. We present an extension of the GSP algorithm using dataset filtering techniques and experimentally evaluate performance gains offered by the proposed method.

Keywords: data mining, sequential patterns

1. Introduction

Data mining aims at discovery of useful patterns from large databases or data warehouses. One of the data mining methods is sequential pattern discovery introduced in [2]. Informally, sequential patterns are the most frequently occurring subsequences in sequences of sets of items.

Among many proposed sequential pattern mining algorithms, most of them are designed to discover all sequential patterns exceeding a user-specified minimum support threshold. Some of them (e.g. *GSP* [8]) also allow users to specify time constraints to be taken into account when checking whether a given data-sequence contains a given subsequence.

However, very often users are interested in patterns that satisfy more sophisticated criteria, for example concerning size or contents of patterns. Data mining tasks involving various types of constraints can be regarded as data mining queries [5].

It is obvious that additional pattern structure constraints 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 [4].

Very little work concerning constraint-driven sequential pattern discovery has been done so far. In fact, only the algorithms from the *SPIRIT* family [3] 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 constrains 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 that pushing regular expression constraints deep into the mining process can reduce processing time by more than an order of magnitude. Nevertheless, it appears that further research on constraint-based sequential pattern mining is needed.

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

– post-processing (filtering out patterns that do not satisfy user-specified pattern constraints after the actual discovery process);
– candidate filtering (application of pattern constraints to reduce the number of processed candidates);
– dataset filtering (restricting the source dataset to objects that can possibly support patterns that satisfy user-specified pattern constraints).

In the context of sequential pattern mining, candidate filtering techniques are represented by the *SPIRIT* algorithm family, whereas dataset filtering techniques have not been considered before. Dataset filtering techniques have been first proposed for efficient constraint-based discovery of association rules. However, due to different pattern constraints and the presence of time constraints, adaptation of these techniques to sequential pattern discovery is not straightforward.

In this paper we present new dataset filtering techniques to be used in the context of sequential pattern discovery. We identify pattern constraints that can be pushed down to dataset selection queries, which leads to a transformed data mining task on a smaller dataset but equivalent in terms of resulting sequential patterns. The proposed techniques can be integrated with any sequential pattern discovery algorithm. We present an efficient way of integrating the dataset filtering techniques with *GSP*, and experimentally evaluate performance gains in comparison with the original *GSP* algorithm.

1.1 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 $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We call $<Y_{i1}\ Y_{i2}\ ...\ Y_{in}>$ 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 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.

1.2 Review of the GSP Algorithm

The *GSP* algorithm, introduced in [8], exploits the following property: all contiguous subsequences of a frequent sequence also have to be frequent. *GSP* makes multiple passes over the data. During the first pass the support of each item is counted. At the end of the first pass, the set of frequent items (equivalent to the set of 1-element frequent sequences) is known. In each subsequent iteration, new potentially frequent sequences, called *candidate* sequences, are generated from the frequent sequences found in the previous pass. Each candidate sequence has one more item than frequent sequences from the previous iteration. In each iteration, the source dataset is scanned to evaluate the support of the candidate sequences and determine which candidates are actually frequent. The algorithm terminates when there are no candidates generated or if none of the candidates turns out to be frequent.

1.3 Related Work

Constraint-driven mining was extensively studied in the context of association rules [6][9][10]. The key step in association rule mining is discovery of frequent itemsets. Techniques of constraint-driven discovery of association rules proposed so far apply two kinds of optimizations: generating only frequent itemsets that can lead to rules satisfying the constraints, and restricting the source collection of sets to those which can contain such itemsets. However, because of the presence of time dependencies and time constraints in case of sequential pattern mining, direct adaptation of techniques proposed for association rules is not possible, or at least not sufficient.

Most of the research on sequential patterns focused on introducing new algorithms, more efficient than *GSP* (e.g. *PrefixSpan* [7]). However, the novel methods do not handle time constraints. Thus, *GSP* still remains the most general sequential pattern discovery algorithm and the reference point for new methods and techniques.

2. Pushing Pattern Constraints into Dataset Selection Queries

In constraint-based sequential pattern mining, we identify the following classes of constraints: database constraints, pattern constraints, and time constraints. Database constraints are used to specify the source dataset. Pattern constraints specify which patterns are interesting and should be returned by the query. Finally, time constraints influence the process of checking whether a given data-sequence contains a given pattern. The basic formulation of the sequential pattern discovery problem introduces three time constraints: max-gap, min-gap, and time window, and assumes only one pattern constraint (the minimum support threshold). We model pattern constraints as complex Boolean predicates having the form of a conjunction of the following basic Boolean predicates on patterns and pattern elements:

- π(**SPG**, α, pattern) - true if pattern support is greater than α, false otherwise;
- π(**SL**, α, pattern) - true if pattern size is less than α, false otherwise;
- π(**SG**, α, pattern) - true if pattern size is greater than α, false otherwise;
- π(**LL**, α, pattern) - true if pattern length is less 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;
- π(**NC**, β, pattern) - true if β is not a subsequence of the pattern, false otherwise;
- π(**SL**, α, pattern_n) - true if the size of the *n*-th element of the pattern is less than α, 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;
- ρ(**NC**, γ, pattern_n) - true if γ is not a subset of the *n*-th element of the pattern, false otherwise.

We believe that the above list of predicates is sufficient to allow users to express their pattern selection criteria. For simplicity's sake, in length and size predicates we consider only sharp inequalities.

Dataset filtering techniques consist in discarding data-sequences that cannot support any pattern satisfying pattern constraints specified by a user. There are two questions that have to be answered. Firstly, basic Boolean predicates concerning patterns or pattern elements whose presence in pattern constraints of a sequential pattern query leads to the possibility of dataset filtering have to be identified. Secondly, for each of the applicable basic Boolean pattern predicates, the corresponding predicate concerning data-sequences has to be provided.

Before we present theorems describing relationships between predicates concerning patterns or pattern elements and properties of data-sequences that can possibly support patterns satisfying these predicates, we have to introduce basic Boolean predicates concerning data-sequences to be used for dataset filtering:

- σ(**SG**, α, sequence) – true if the size of the data-sequence is greater than α, false otherwise;
- σ(**LG**, α, sequence) – true if the length of the data-sequence is greater than α, false otherwise;

- σ(**C**, β, sequence, maxgap, mingap, window) – true if the data-sequence contains the sequence forming the pattern β using given time constraints, false otherwise;
- σ(**CS**, α, sequence, 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;
- σ(**CL**, α, sequence, maxgap, mingap, 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.

Theorem 1 Sequential patterns of size greater than k cannot be supported by 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 2 Sequential patterns of length greater than k, to be returned by a data mining query, can be supported only by 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 3 Sequential patterns, to be returned by a data mining query, containing a given sequence can be supported only by 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 4 Sequential patterns, to be returned by a data mining query, whose n-th element has the size greater than k can be supported only by 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 5 Sequential patterns, to be returned by a data mining query, whose n-th element contains a given set can be supported only by 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.

The above theorems concern the basic Boolean predicates on patterns or pattern elements that can be used for dataset filtering and provide corresponding data-sequence predicates to be used in the filtering process. These predicates and their corresponding data-sequence predicates are presented in Table 1.

Table 1. Basic Boolean predicates on patterns or pattern elements and corresponding data-sequence predicates

Basic Boolean predicate on a pattern or n-th element of a pattern	Basic Boolean predicate on a data-sequence
π(**SG**, α, pattern)	σ(**SG**, α, sequence)
π(**LG**, α, pattern)	σ(**CL**, α+1, sequence, *max*, *min*, *win*)
π(**C**, β, pattern)	σ(**C**, β, sequence, +∞, *min*, *win*)
ρ(**SG**, α, pattern_n)	σ(**CS**, α+1, sequence, win)
ρ(**C**, γ, pattern_n)	σ(**C**, <γ>, sequence, *max*, *min*, *win*)

In the above table, <γ> 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.

The presence of other basic Boolean predicates in pattern constraints of a sequential pattern query does not affect the filtering process since patterns having support greater than a certain value, size or length less than a certain value, size of the n-th element less than a certain value, as well as patterns not containing a certain sequence, or whose *n*-th element does not contain a certain itemset, can be supported by any data-sequence.

Example 1 Consider the following sequential pattern query (for the query we present time and pattern constraints, the specification of the source dataset is omitted for brevity): DMQ = {max-gap: 100, min-gap: 7, window-size: 0, π(**SPG**, 0.01, pattern) ∧ π(**SG**, 3, pattern) ∧ π(**LL**, 5, pattern) ∧ π(**C**, <(A)(B)>, pattern)}. The query returns sequential patterns having support greater than 1%, having more than three items but less than five elements, containing the sequence <(A)(B)>. The specified values of max-gap, min-gap, and window-size constraints are 100, 7, and 0 respectively.

Thus, according to the Table 1 (based on the Theorems 1 – 5), the following dataset filtering predicate has to be satisfied by a data-sequence containing a pattern satisfying pattern constraints: σ(**SG**, 3, sequence) ∧ σ(**C**, <(A)(B)>, sequence, +∞, 7, 0). The dataset filtering predicate says that only data-sequences having more than three items and containing the sequence <(A)(B)> with max-gap of +∞, min-gap of 7, and window-size of 0, have to be considered in the discovery process.

According to the Theorems 1 - 5, a sequential pattern query having one or more basic Boolean pattern predicates from the left column of Table 1 in its pattern constraints 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 with appropriate data-sequence predicates. Secondly, the minimum support constraint 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 data-sequences containing the pattern. The Theorems 1 – 5 guarantee that the number of data-sequences 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: $sup_F(p) = |D| * sup(p) / |D_F|$, where $sup_F(p)$ and $sup(p)$ denote the support of the pattern *p* in the filtered and original dataset respectively, and $|D_F|$ and $|D|$ denote the number of data-sequences 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 data-sequences in the original dataset according to the above formula.

Dataset filtering techniques can be combined with any sequential 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 data-sequences that cannot contain patterns of interest) and conversion of pattern constraints concerning pattern support can be performed before the actual discovery process. However, in reality such explicit transformation might be impossible due to space limitations. Some sequential pattern discovery algorithms perform certain projections of the database (e.g. *PrefixSpan*) by nature, while others (e.g. *GSP*) do not transform the database in any way, which is a serious advantage if the database is large and the storage space limited. We believe that if the original algorithm has some desirable properties, any extension applied to the algorithm should preserve them.

3. Integration of Dataset Filtering Techniques with GSP

In this section we present an extension of the *GSP* algorithm (denoted as *GSP-F*) exploiting dataset filtering techniques to support efficient constraint-based sequential pattern mining. We chose *GSP* as the basis for implementing the dataset filtering techniques for two reasons. Firstly, *GSP* (and its extensions) is still the only sequential pattern discovery algorithm supporting time constraints, which affect the dataset filtering process. Secondly, *GSP* does not create any temporal structures to store portions of the source database and preserving this property may be a challenging task.

GSP iteratively generates candidate sequences and evaluates their support by testing their occurrence in each data-sequence from the source dataset. Since we do not want to materialize the filtered dataset, filtering has to be performed on-line in each iteration of the algorithm. Data-sequences that do not satisfy dataset filtering constraints derived from pattern constraints are excluded from the candidate verification process (conceptually the discovery process takes place in the reduced dataset). It should be noted that since we do not explicitly transform the data mining task into an equivalent one, the support conversions discussed in the previous section are not necessary.

The detailed *GSP-F* algorithm extending *GSP* with dataset filtering techniques is presented below. The algorithm takes a collection *D* of data-sequences as input, and returns all sequential patterns in *D* supporting user-specified pattern and time constraints.

Algorithm *GSP-F*

```
DF = dataset filtering predicate derived from pattern constraints;
scan D in order to:
1) evaluate minimum number of supporting data-sequences for
a pattern to be called frequent (mincount)
```

```
2) find $L_1$ (set of 1-sequences contained in at least *mincount*
data-sequences satisfying *DF*);
for ($k$ = 2; $L_{k-1} \neq \emptyset$; $k$++) do
begin
    $C_k$ = apriori_gen($L_{k-1}$);    /* generate new candidate sequences */
    if $C_k = \emptyset$ then break;
    forall data-sequences $d \in D$ do
        if $d$ satisfies *DF* then
            forall candidates $c \in C_k$ do
                if $d$ contains $c$ using user-specified time constraints then
                    $c$.count ++;
                end if;
        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;
```

The algorithm starts with deriving dataset filtering constraints from pattern constraints provided by a user. These dataset filtering constraints are used in each scan of the source dataset and data-sequences 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. This phase is required since dataset filtering itself 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.

GSP-F does not reduce the amount of data read from the database in each iteration in comparison to the original *GSP* algorithm. However, we expect it to be more efficient since data-sequences that do not satisfy dataset filtering constraints are excluded from the costly candidate verification process.

4. Experimental Results

In order to evaluate performance gains offered by our dataset filtering techniques, we performed several experiments on a synthetic dataset generated by means of the *GEN* generator from the *Quest* project [1]. The dataset contained 1000 data-sequences built from 50 different items, the average number of transactions per data-sequence was 5.5, and the average number of items per transaction was 1.2. Since *GEN* does not generate transaction times, we treated transaction identifiers as transaction times, thus the time gap between each two consecutive elements of each data-sequence was always equal to one time unit. The generated data-sequences were stored in a database table (a local *Oracle8i* database server was used).

We started the experiments with varying the constraints regarding pattern size, length, and contents for the fixed minimum support threshold of 1%, infinite max-gap, and min-gap and window-size equal to 0. Apart from measuring the total execution times of *GSP* and *GSP-F*, we also registered the time spent by *GSP-F* on dataset filtering, and the selectivity of dataset filtering constraints derived from pattern constraints (expressed as the percentage of data-sequences in the database satisfying dataset filtering constraints). Figure 1 presents the ratio of the execution time of *GSP-F* to the execution time of *GSP* for different values of selectivity of the derived dataset filtering constraints.

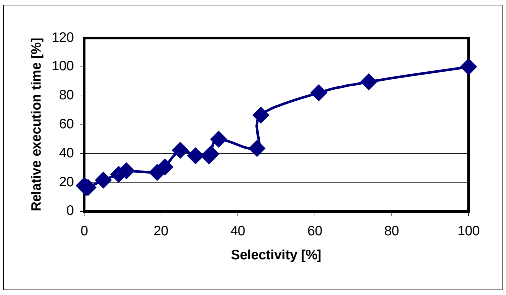

Figure 1. Performance improvements for different values of selectivity

As we expected, the experiments showed that the lower the selectivity of dataset filtering constraints, the better the performance of *GSP-F* is likely to be as compared to *GSP*. However, we observe that sometimes the performance improvement might be better for a query leading to a larger filtered dataset. This can be easily explained by the fact that performance of *GSP-F* depends not only on the number of data-sequences against which candidates generated in each iteration have to be verified but also on the number of the candidates, which depends on the data distribution within the filtered dataset. Nevertheless, dataset filtering reduces the processing time (usually several times), except for an unlikely situation when dataset filtering constraints derived from pattern constraints do not filter out any source data-sequences. However, even that unrealistic situation does not pose a problem since, according to our experiments, the time spent on extra dataset filtering operations constitutes less than 1% of the overall processing time.

The selectivity of dataset filtering constraints depends on the original pattern constraints but also on the actual contents of the database. In general, we observed that pattern constraints involving the presence of a certain subsequence or subset led to much better results (reducing the processing time 2 to 5 times) than constraints referring only to pattern size or length (typically reducing the processing time by less than 10%). This is due to the fact that sequential patterns are usually smaller in terms of size and length than source data-sequences, and therefore even restrictive constraints on pattern size/length result in weak constraints on data-sequences.

In another series of experiments, we observed the influence of varying the minimum support threshold and time constraints on performance gains offered by dataset filtering. *GSP* encounters problems when the minimum support threshold is low or time constraints are relaxed (large max-gap and window-size, small min-gap) because of the huge number of candidates to be verified. In our experiments, decreasing the minimum support threshold or relaxing time constraints worked in favor of our dataset filtering techniques, leading to bigger performance gains. (Figure 2 presents the influence of varying the minimum support from 0.5% to 2% on the relative processing time of *GSP-F* compared to *GSP* for an example query with min-gap = 0, max-gap = +∞, win-size = 0 and pattern constraints resulting in dataset filtering constraint having the selectivity of 25%.) 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.

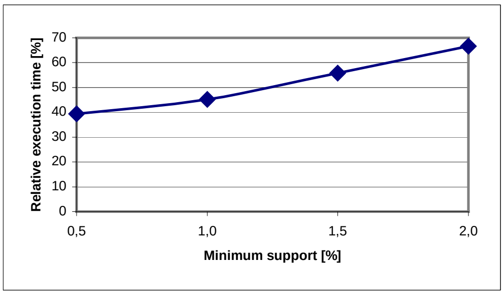

Figure 2. Performance improvements for different values of minimum support

5. Concluding Remarks

We have discussed the application of dataset filtering techniques to efficient sequential pattern mining in the presence of various pattern constraints. We identified the set of pattern selection predicates that support dataset filtering and presented the method of pushing these constraints down to dataset selection queries. Dataset filtering techniques can be applied to any sequential 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 with the *GSP* algorithm. Our experiments show that dataset filtering can result in significant performance improvements, especially in case of pattern constrains involving the presence of a certain subsequence or subset.

References

[1] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, T. Bollinger, The Quest Data Mining System, In E. Simoudis, J. Han, U. M. Fayyad (Eds.), Proceedings of the 2nd KDD Conference, Portland, Oregon, AAAI Press, 1996

[2] R. Agrawal, R. Srikant, Mining Sequential Patterns, In P. S. Yu, A. L. P. Chen (Eds.), Proceedings of the 11th ICDE Conf., Taipei, Taiwan, IEEE Computer Society, 1995

[3] M. Garofalakis, R. Rastogi, K. Shim, SPIRIT: Sequential Pattern Mining with Regular Expression Constraints, In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, M. L. Brodie (Eds.), Proceedings of 25th VLDB Conference, Edinburgh, Scotland, UK, Morgan Kaufmann, 1999

[4] J. Han, L. Lakshmanan, R. Ng, Constraint-Based Multidimensional Data Mining, IEEE Computer, Vol. 32, 1999; 8:46-50

[5] T. Imielinski, H. Mannila, A Database Perspective on Knowledge Discovery, Communications of the ACM, Vol. 39, 1996; 11:58-64

[6] R. Ng, L. Lakshmanan, J. Han, A. Pang, Exploratory Mining and Pruning Optimizations of Constrained Association Rules, In L. M. Haas, A. Tiwary (Eds.), Proceedings of the 1998 SIGMOD Conference, Seattle, Washington, ACM Press, 1998

[7] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M-C. Hsu, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, Proceedings of the 17th ICDE Conference, Heidelberg, Germany, IEEE Computer Society, 2001

[8] R. Srikant, R. Agrawal, Mining Sequential Patterns: Generalizations and Performance Improvements, In P. M. G. Apers, M. Bouzeghoub, G. Gardarin (Eds.), Proceedings of the 5th EDBT Conference, Avignon, France, Springer, 1996

[9] R. Srikant, Q. Vu, R. Agrawal, Mining Association Rules with Item Constraints, In D. Heckerman, H. Mannila, D. Pregibon (Eds.), Proceedings of the 3rd KDD Conference, Newport Beach, California, AAAI Press, 1997

[10] M. Zakrzewicz, Data Mining within DBMS Functionality, In A. Caplinskas (Ed.), Proceedings of the 4th IEEE International Baltic Workshop on Databases & Information Systems, Vilnius, Lithuania, 2000