SEQUENTIAL PATTERN MINING WITH APPROXIMATED CONSTRAINTS

Cláudia Antunes
Instituto Superior Técnico / INESC-ID,
Department of Information Systems and Computer Science,
Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
claudia.antunes@dei.ist.utl.pt

Arlindo L. Oliveira
Instituto Superior Técnico / INESC-ID,
Department of Information Systems and Computer Science,
Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
aml@inesc-id.pt

ABSTRACT

The lack of focus that is a characteristic of unsupervised pattern mining in sequential data represents one of the major limitations of this approach. This lack of focus is due to the inherently large number of rules that is likely to be discovered in any but the more trivial sets of sequences. Several authors have promoted the use of constraints to reduce that number, but those constraints approximate the mining task to a hypothesis test task.
In this paper, we propose the use of constraint approximations to guide the mining process, reducing the number of discovered patterns without compromising the prime goal of data mining: to discover unknown information. We show that existent algorithms, that use regular languages as constraints, can be used with minor adaptations. We propose a simple algorithm (ε-*accepts*) that verifies if a sequence is approximately accepted by a given regular language.

KEYWORDS

Data Mining, Pattern Mining, Sequential Pattern Mining, Constraints, Constraint Relaxations, Deterministic Finite Automata.

1. INTRODUCTION

The rapid increase of stored data in digital form has enlarged the interest in the automatic discovery of hidden information, and *data mining* techniques have gained acceptance as a viable mean to perform that task.

Usually, the term data mining is used to mean "the nontrivial extraction of implicit, previously unknown and potential useful information from data" [Frawley 1992]. In general, data mining techniques have been successfully applied from commercial domains, like customer relationship management, market basket analysis or credit card fraud detection, to scientific and engineering applications.

However, data mining algorithms are usually unable to produce optimal results with respect to all the trade-offs that they account for: precision versus recall, support versus confidence, concise and understandable models versus highly accurate black boxes, sample size versus error rate, or simply model expressiveness versus compute time, to name a few [Bayardo 2002]. In particular, algorithms for discovering frequent patterns (usually known as *pattern mining* algorithms) discover large amounts of patterns, most of the times, uninteresting and useless to the final user. The inability to focus the discovery process on user expectations and background knowledge, has lead to a process that is, in many cases, prohibitively expensive and very difficult to deal. The treatment of sequential data (the analysis of frequent behaviors, for example) is a particular case of pattern mining, usually known as sequential pattern mining, and suffers from the same drawbacks.

In order to minimize this problem, in either pattern or sequential pattern mining, recent approaches use constraints to restrict the number and scope of discovered patterns. Using constraints makes it possible to focus the mining process into areas or sub-spaces where useful information is likely to be gained. The ability to express or exploit constraints effectively allows the data miner to use knowledge to guide the data mining process [Bayardo 2002]. However, as pointed by some authors, restricting the search too much may approximate the mining process to a simple testing hypothesis task, since the process will find only already known patterns [Hipp 2002].

In this paper, we present a new pattern mining approach, which keeps the focus on user expectations and allows for the use of existing background knowledge, while keeping in sight the main goal of data mining: the discovery of new information. This new approach consists on the use of a constraint relaxation (expressed as a formal language [Garofalakis 1999] [Antunes 2002]) to guide the mining process, allowing the user to choose the strength of the restriction he wants to impose in the pattern mining process. This relaxation is just an approximation to the original constraint and makes the discovery of unknown information possible.

The rest of this paper is organized as follows: section 2 describes the sequential pattern mining problem and its main application areas. A special emphasis is given to the paradigm of constrained-based mining, pointing out its advantages and disadvantages. Section 3 presents a new pattern mining approach based on the use of approximated constraints, and presents an algorithm (ε-*accepts*) to find if a sequence could be generated by a regular language, when an error is allowed. Section 4 presents some performance results, and finally, section 5 presents the conclusions and points some guidelines for future research.

2. SEQUENTIAL PATTERN MINING

The problem of sequential pattern mining is one of the several that have deserved particular attention from the data mining community. The sequential nature of the problem appears when the data to be mined is naturally embedded in a one-dimensional space, i.e., when one of the relevant pieces of information can be viewed as one ordered set of elements. This variable can be time or some other dimension, as is common in other areas, like bioinformatics. Sequential pattern mining is defined as the process of discovering all sub-sequences that appear frequently on a given sequence database. The challenge resides in figuring out what sequences to try and then efficiently finding out which of those are frequent [Srikant 1996].

One of its obvious applications is on modeling the behavior of some entity along time. For instance, by using a database with transactions performed by customers at any instant, it is desirable to predict what would be the customer's next transaction, based on his past transactions. Examples of these tasks are easily found on the prediction of financial time series or on tasks related to patients' health monitoring.

2.1 Problem Definition

Sequential Pattern Mining algorithms address the problem of discovering the existent maximal frequent sequences in a given database. Algorithms for this problem are relevant when the data to be mined has some sequential nature, i.e., when each piece of data is an ordered set of elements, like events in the case of temporal information, or nucleotides and amino-acid sequences for problems in bioinformatics.

The problem was first introduced by Agrawal and Srikant, where the basic concepts involved in pattern detection were established [Agrawal 1995]. In the last years, several sequential pattern mining algorithms were proposed, but not all assume the same conditions. Despite the fact that the use of constraints is somehow orthogonal to the general philosophy of pattern mining algorithms, the use of some constraints (for example the gap constraint) has a great impact on their design and efficiency, as shown in [Antunes 2003].

Some basic definitions are needed, in order to formally introduce the problem:

Definition 1 - *An itemset is a non-empty subset of elements from a set C, the item collection, called items.*

An itemset, also called a basket, represents the set of items that occur together. If the data is time dependent, an itemset corresponds to the set of items transacted in a particular instant by a particular entity. The itemset composed of items *a* and *b* is denoted by (*ab*).

Definition 2 - *A sequence is an ordered list of itemsets. A sequence is maximal if it is not contained in any other sequence.*

A sequence with *k* items is called a *k*-sequence. The number of elements (itemsets) in a sequence *s* is the length of the sequence and is denoted by |s|. The i^{th} itemset in the sequence is represented by s_i. and <> denotes the empty sequence. The result of the concatenation of two sequences *x* and *y* is a new sequence denoted by *xy*.

The set of considered sequences is usually designated by database (DB), and the number of sequences by database size (|DB|).

Definition 3 - *A sequence $a=<a_1a_2... a_n>$ is contained in another sequence $b=<b_1b_2... b_m>$, or a is a subsequence of b, if there exist integers $1\leq i_1<i_2<\ldots<i_n\leq m$ such that $a_1\subseteq b_{i1}$, $a_2\subseteq b_{i2}$, …, $a_n\subseteq b_{in}$.*

A subsequence s' of s is denoted by $s'\subseteq s$, and by $s'\subset s$ if s' is a proper subsequence of s, i.e. if s' is a subsequence of s but is not equal to s.

It is usual to assume that the items in an itemset of a sequence are in lexicographic order. This assumption facilitates the design of sequential pattern mining algorithms, avoiding the repetition of some operations (such as the generation of repeated sequences). In this manner, prefixes and suffixes have specific meanings. They are special cases of subsequences: the sequence without the first element in the first itemset of the sequence and without the last element of the last itemset of the sequence, respectively.

Finally, the sequential pattern-mining problem may be stated in its entirety.

Definition 4 - *Given a database D of sequences, and some user-specified minimum support threshold σ and constraint c, a sequence is frequent if it is contained in at least σ sequences in the database, satisfying the constraint c. A sequential pattern is a maximal sequence that is frequent.*

2.2 Algorithms

There are two main approaches to the sequential pattern-mining problem: apriori-based and pattern-growth methods, with *GSP* [Srikant 1996] and *PrefixSpan* [Pei 2001] their best-known implementations, respectively. Despite there are several implementations of apriori-based methods, most of them assume some specific situations (for example that the entire dataset fits in memory, see for example [Ayres 2002]), allowing for significant improvements. Although *GSP* considers time constraints and uses taxonomies, if no taxonomy is provided, and time constraints are not set both algorithms have similar goals: to discover sequential patterns, without considering any constraints and without any database size restrictions.

GSP [Srikant 1996] follows the candidate generation and test philosophy. It begins with the discovery of frequent 1-sequences, and then generates the set of potentially frequent (k+1)-sequences from the set of frequent k-sequences (usually called candidates).

The generation of potentially frequent k-sequences (k-candidates) uses the frequent (k-1)-sequences discovered in the previous step, which may reduce significantly the number of sequences to consider at each moment. Note that to decide if one sequence s is frequent or not, it is necessary to scan the entire database, verifying if s is contained in each sequence in the database.

In order to reduce its processing time, *GSP* adopts three optimizations. First, it maintains all candidates in a hash-tree to scan the database once per iteration. Second, it only creates a new k-candidate when there are two frequent (k-1)-sequences with the prefix of one equal to the suffix of the other. Third, it eliminates all candidates that have some non-frequent maximal subsequence. By using these strategies, *GSP* reduces the time spent in scanning the database, increasing its general performance.

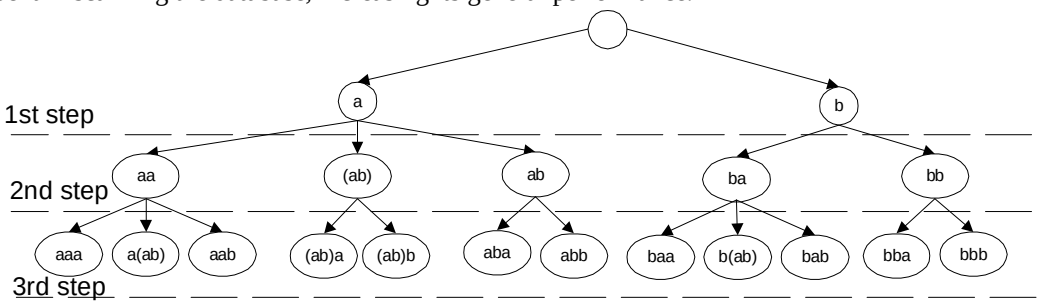

Figure 1 Illustration of apriori-based algorithms execution

In general, apriori-based methods can be seen as breadth-first traversal algorithms, since they construct all k-patterns simultaneously. Consider a database that is composed of sequences with items belonging to the set Σ={a, b}. If all possible arrangements of these two elements are frequent, *GSP* would work as illustrated in Figure 1. At the end of the first iteration, *GSP* would have found *a* and *b* as 1-sequences. At the end of the second step it would have found {*aa*, (*ab*), *ab*, *ba*, *bb*}. Finally, at the end of the third iteration, *GSP* would have found all arrangements of 2 items taken 3 at a time, including the basket (*ab*) {*aaa*, *a*(*ab*), *aab*, (*ab*)*a*, (*ab*)*b*, *aba*, *abb*, *baa*, *b*(*ab*), *bab*, *bba*, *bbb*}.

Pattern-growth methods are a more recent approach to deal with sequential pattern mining problems. The key idea is to avoid the candidate generation step altogether, and to focus the search on a restricted portion of the initial database.

PrefixSpan [Pei 2001] is the most promising of the pattern-growth methods and is based on recursively constructing the patterns, and simultaneously, restricting the search to projected databases. An α-projected database is the set of subsequences in the database, which are suffixes of the sequences that have prefix α. In each step, the algorithm looks for the frequent sequences with prefix α, in the correspondent projected database. In this way, the search space is reduced at each step, allowing for better performances in the presence of small support thresholds. In the presence of gap constraints, the algorithm has to be adapted, and this claim is no longer valid. However, as been shown, the new version of *PrefixSpan* remains faster than *GSP* [Antunes 2003].

In general, pattern-growth methods can be seen as depth-first traversal algorithms, since they construct each pattern separately, in a recursive way. If we consider the same database as previously, *PrefixSpan* would work as illustrated in Figure 2.

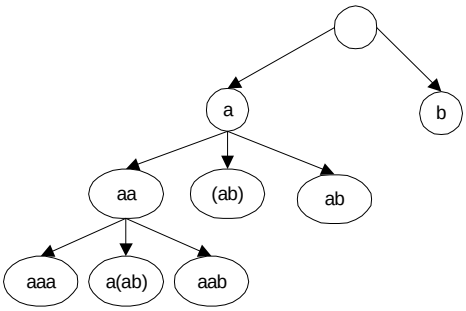

Figure 2 Illustration of pattern-growth methods execution

The main drawbacks of pattern mining in general and of sequential pattern mining in particular, are: the large amount of discovered patterns; its inability to use background knowledge; and the lack of focus on user expectations. In order to resolve these problems, the use of constraints was proposed.

A *constraint* is a predicate on the powerset of the set of items I, i.e. $C:2^I \rightarrow \{true, false\}$. Given a transaction database, a support threshold and a constraint *C*, the problem of constrained frequent pattern mining is the selection of the complete set of frequent itemsets satisfying *C* [Pei 2002a].

This approach has been widely accepted by the data mining community, since it allows the user to control the mining process, either by introducing his background knowledge deeply into the process or by narrowing the scope of the discovered patterns. The use of constraints also reduces the search space, which contributes significantly to achieve better performance and scalability levels ([Srikant 1996], [Pei 2002a] and [Garofalakis 1999]).

2.2.1 Algorithms with Constraints

SPIRIT [Garofalakis 1999] is a family of apriori-based algorithms that uses a regular language to constrain the mining process. The core of the algorithm is similar to *GSP*, and its main difference resides on the candidate generation step, which creates candidates that potentially satisfy the constraint. However, most of the interesting regular languages are not anti-monotone, and the candidate generation has to use constraint relaxations in order to be able to construct sequences that, at the end, are accepted by the constraint. The similarity of this procedure to the original in *GSP* depends on wether the relaxation is anti-monotone or not. If it is, *SPIRIT* can adopt the same philosophy (joining two (k-1)-sequences to generate a k-sequence), but if it is not *SPIRIT* has to extend a frequent (k-1)-pattern with a frequent element (1-pattern), generating more

non-frequent sequences than strictly necessary. Subsequent work [Antunes 2002] has shown that regular languages can be substituted by context-free languages, without compromising the algorithm's performance. This is useful because context-free languages are more expressive than regular languages, being able to represent knowledge that is more interesting.

PrefixGrowth [Pei 2002b] is a pattern-growth method, similar to *PrefixSpan* that only generate sequences that potentially satisfies the constraint *C*, this is, sequences that are prefixes of valid sequences according to the constraint. It has been shown that *PrefixGrowth* is efficient since the constraint is prefix-monotone, which means that either, "for every sequence that satisfies *C*, also do all of its prefixes, or for each sequence *s* satisfying *C* implies that all sequences that have *s* as a prefix also satisfy *C*". However, like *PrefixSpan*, *PrefixGrowth* does not deal with gap constraints. Fortunately, the extensions applied to the first algorithm can also be applied to *PrefixGrowth*.

3. APPROXIMATE CONSTRAINTS

As pointed by some authors [Hipp 2002], when used incautiously, constrained pattern mining may reduce to a hypothesis-testing task. Note that, if blindly applied, it avoids the discovery of unknown and unexpected patterns, which is the first and foremost data mining's goal.

Indeed, the formal languages may be seen as a parametric model for a set of sequences, which means that the mining process will only re-discover the information already known. However, if no constraint is used, existing background knowledge is completely discarded, which turns the mining process less efficient.

The key question is: how to combine the use of existing knowledge and the discovery of unknown information? We argue that the answer to this challenge is the use of constraint relaxations.

The notion of *constraint relaxation* was introduced in [Garofalakis 1999], to improve the performance of SPIRIT algorithm, and consists of a weaker constraint. If those relaxations are used to mine new patterns, instead of simply being used for filtering the patterns that satisfy the original constraint, the discovery of unknown information is possible, as shown in this paper. Since the user may choose the level of relaxation allowed, it is possible to keep the focus and the interactivity of the process, while still allowing for the discovery of new and unknown information.

However, proposed relaxations are not powerful enough to discover several interesting possibilities. Consider for example the case of biological sequences: they are the result of successive mutations along millions of years. These mutations are the basis for molecular evolution, and consist of insertions, deletions, transpositions or simply of point mutations. The discovery of some of such mutations is the key for the study of some of human inherited diseases. As pointed in earlier works [Searls 1992] *approximate matching* at a lexical level is an extremely important tool to assist in the discovery of new facts, and it mainly consists in considering two sequences similar if they are at an edit distance below a given threshold.

In order to solve this problem we propose a relaxation that accepts sequences that have a limited number of errors. If it is possible to correct those errors with a limited cost, then the sequence will be accepted. A new relaxation, called *approximately-accepted*, tries to accomplish this goal: it only accepts sequences that are at a given edit distance for a string accepted. This edit distance reflects the cost of operations that have to be applied to a given sequence, so it would be accepted as a positive example of a given formal language, and it will be called the *generation cost*. This cost is similar to the edit distance between two strings, and similarly, the operations to consider can be the *Insertion*, *Deletion* and *Replacement* [Levenstein 1965].

Definition 5 - *Given a constraint C, expressed as a regular language, and a real number ε which represents the maximum error allowed, a sequence $s=<s_1 \ldots s_n>$ is said to be approximately-accepted by C, if its generation cost $\xi(s, C)$ is less than or equal to ε.*

With the generation cost defined as follows

Definition 6 - *Given a constraint C, expressed as a regular language and a sequence $s=<s_1 \ldots s_n>$, the generation cost $\xi(s, C)$ is defined as the sum of costs of the cheapest sequence of edit operations $O=<o_1 \ldots o_k>$ transforming the sequence s to be accepted by the regular language C. The edit operations considered are the traditional in approximate string matching::*

- *Insertion – Ins(x,i) – adds the symbol x to the sequence at position i;*
- *Deletion – Del(x,i) – deletes the symbol x from the sequence at position i;*
- *Replacement – Repl(x,y,i) – substitutes the symbol x, that occurs at position i, by symbol y.*

Note that regular languages can range from very restrictive (like the language defined by the regular expression *ab*(*a*|*b*), which only accepts strings with three elements: a string with an *a* followed by a *b*, *followed by* another *a* or *b*), to unrestrictive (like the language defined by (a|b)*, which accepts any string built from {*a*, *b*}).

3.1.1 Algorithm ε-*accepts*

Unfortunately, the better-known algorithms for edit distance are performed on two sequences, and cannot be applied to one sequence and a formal language. *Agrep* [Wu 1992] verifies if a sequence was generated by a given formal language (expressed as an automaton) unless an error. However, and as pointed by its authors, *agrep* does not deal well with very large alphabets, and sequential pattern mining algorithms usually deal with alphabets with thousands of elements. In order to deal with this situation, and considering that useful automata are small in general, we propose *ε–accepts*, a new algorithm to verify if a sequence was approximately generated by a given deterministic finite automata (DFA), which is illustrated in Figure 3.

```
boolean ε-accepts(Sequence s=<s1..sn>, int ε,
                  DFA A=(Q,Σ,δ,q0,F)){
    return aproxAcc(s, 1, A.q0, 0, ε, A);
}
```

```
boolean aproxAcc(Sequence s, int i, Q qj,
                 int acε, int ε, DFA A=(Q,Σ,δ,q0,F)){
    // Exceeds maximal error
    if (acε>ε)
        return False;
    // Procede with next element
    if (i≤|s|)
        return accTrans(s,si,i,qj,acε,ε,A);
    // Test the achievement of final state
    else if (qj ∈ A.F)
        return True;
    // Try to achieve a final state
    else
        return accTrans(s,(),i,qj,acε,ε,A);
}
```

```
boolean accTrans(Sequence s, ItemSet si,int i,
          Q qj, int acε, int ε, DFA A=(Q,Σ,δ,q0,F)){
    for each δ(q,a)∈Aδ(qj){
        int dif = max(|si|,|a|)-|a ∩ si|;
        // Found the transition
        if (|si|=|a ∩ si|)
            ok=aproxAcc(s,i+1,q,acε,ε,A);
        //Try edit operations/replacement
        else
            ok=aproxAcc(s,i+1,q,acε+dif,ε,A);
        if (ok) return True;
        // If element is not valid,delete
        ok = aproxAcc(s,i+1,qj,acε+|si|,ε,A);
        if (ok) return True;
        // If deletion fails, try insertion
        ok = aproxAcc(s,i,q,acε+|a|,ε,A);
        if (ok) return True;
    }
    return False;
}
```

Figure 3 ε–*accepts* algorithm

ε–*accepts* is a recursive algorithm, which main idea is to avoid the generation of potential sequences, given that this operation consumes a considerable amount of time in sequential pattern mining algorithms. In this manner, the algorithm simulates the flow of transitions between automaton states, for each itemset in the input sequence. Whenever the maximum error is achieved, the algorithm backtracks to try one or another of the possible edit operations. Instead of generating the possible sequences, considering possible errors, we verify if each sequence element performs a valid transition in the automaton, beginning on the initial state. Whenever an element does not correspond to a valid transition, the algorithm tries to replace it (which corresponds to apply a *Replacement*), and if this fail, then tries to ignore (which corresponds to a *Deletion*) and finally by trying to introduce a valid transition (which corresponds to an *Insertion*). Given that *insertions* and *replacements* only try valid transitions, instead of trying every possible element, the algorithm's performance does not depend on the size of the number of different elements in the database, but only on the DFA number of states and alphabet.

Since we consider that DFAs are usually small, the simulation of transitions is not an expensive operation. (As some authors have noted, experts usually make use of a few simple and small rules to express their background knowledge, so the combination of those rules usual do not result on a complex automata).

4. RESULTS

In this section, we will illustrate the use of a simple constraint (expressed as a regular grammar) with a synthetic example. Our goal is to validate our claim that constraint approximations allow the discovery of unknown information, keeping the mining process centered on the user.

In order to do it, we have used the standard synthetic data set generator from IBM Almaden [Srikant 1996]. This data generator has been used in most studies on sequential pattern mining, and it generates datasets that mimic real-world transactions, where customers tend to make a sequence of transactions involving a set of items. Sequence and transaction sizes are clustered around a mean and some of them may have larger sizes. Note that a sequence may have repeated transactions, but a transaction cannot have repeated items.

The dataset contains 10.000 sequences (Parameter D of the generator set to 10), with 10 transactions each on the average (C=10). Each transaction has on the average 2 items (T=2). The average length of maximal patterns is set to 4 (S=4) and maximal frequent transactions set to 2 (I=2). These values were chosen in order to follow closely the parameters usually chosen in other studies. The values for different sequential patterns (Ns) and transactional patterns (Ni) were also chosen similarly, set to 5.000 and 10.000, respectively. However, the number of different items was set to 10 in order to increase the number of discovered patterns.

In order to evaluate our claim we proceeded as proposed in [Antunes 2002]: first, the unconstrained patterns are discovered, then a regular grammar able to represent part of the patterns is defined, and finally this constraint is applied to find the accepted and approximately accepted patterns.

Table 1 Comparison of the results achieved with and without constraints

Unconstrained Patterns			
2	*262*	*26*	*5*
(2,6)	*22*	*25*	*52*
(2,6)2	*2(2,6)*	*20*	*522*
(2,5)	*2(2,5)*	*29*	*0*
(2,5)2	*222*	*6*	*02*
(2,0)	*2222*	*62*	*9*
(2,9)	*226*	*622*	*7*

Table 1 presents the unconstrained patterns discovered using 50% as the minimum support threshold, and Figure 4 shows the automata defined over those patterns, which accepts about 20% of the unconstrained patterns and accepts about 30% as valid prefixes.

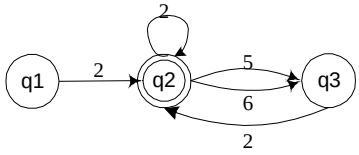

Figure 4 DFA defined over unconstrained patterns

As expected, by using the constraint itself we only discover already known information, which satisfy the imposed language. Therefore, these results are not able to invalidate Hipp's arguments about constraints.

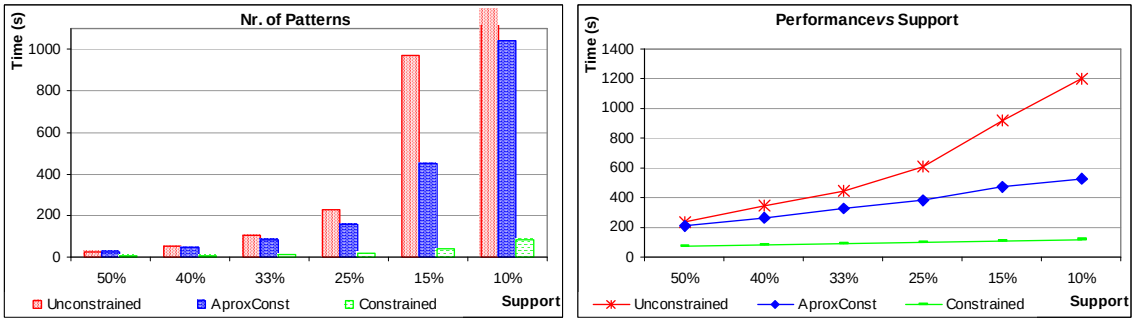

Figure 5 a) Number of discovered patterns; b) Performance using unconstrained and constrained algorithms

However, the use of the constraint approximation makes possible the discovery of unknown information. For example for a support threshold equal to 50%, all unconstrained patterns are discovered if we accept an error of 1. Figure 5.a) shows that the number of discovered patterns for approximately-accepted patterns is considerably larger than the number of discovered constrained patterns (about five times).

Figure 5.b) shows that the fully constrained algorithm spends much less time then the approximately constrained one, the difference to the time spent by unconstrained algorithms is significant.

The computer used to run the experiments was a Pentium IV 2,8GHz with 1GB of RAM. To make the time measurements more reliable, no other application was running on the machine while the experiments were running. The operating system used was Windows XP and the algorithms were implemented using the Java programming language (Java Virtual Machine version 1.4.2_01). The datasets were maintained in main memory during the algorithms processing, avoiding hard disk accesses. And, a gap constraint equal to 0 was used in the patterns discovery.

5. CONCLUSION

Sequential pattern mining has been one of the mining techniques applied to predict customer behaviors in areas such as basket analysis or log monitoring. However, existent techniques have been unable to use effectively existent knowledge, since constrained algorithms only discover already known information.

In this paper, we propose the use of constraint approximations, in order to guide the mining process, without loosing the ability to discover unknown information. Constraints are represented as regular languages (expressed as deterministic finite automata), and the verification of the acceptance of sequences is performed by a new algorithm – ε-*accepts*.

We show that approximately acceptance is able to guide the mining process, discovering unknown information and not compromising algorithms performance.

Since regular languages are not able to represent some interesting phenomena, the next step is to use context-free languages instead. However, those languages introduce some interesting challenges when dealing with sequences of itemsets instead of elements.

REFERENCES

Agrawal, R. and Srikant, R., 1995. Mining Sequential Patterns. *Proc. 11th Int. Conf. Data Engineering* (ICDE 95), Taipei, Taiwan, pp. 3-14.

Antunes, C. and Oliveira, A.L., 2002. Inference of Sequential Association Rules Guided by Context-Free Grammars. *Proc. Int'l Conf. on Grammatical Inference* (ICGI 2002). Amsterdam, Netherlands, pp. 1-13.

Antunes, C. and Oliveira, A.L., 2003. Generalization of Pattern-growth Methods for Sequential Pattern Mining with Gap Constraints. *Proc. Int'l Conf. Machine Learning and Data Mining* (MLDM'03), pp. 239 - 251.

Ayres, J., Gehrke, J., Yu, T. and Flannick, J., 2002. Sequential PAttern Mining using a Bitmap Representation. *Proc Int'l Conf Knowledge Discovery and Data Mining*, pp. 429-435.

Bayardo, R.J., 2002. The Many Roles of Constraints in Data Mining. *SIGKDD Explorations*, vol. 4, no. 1, pp. i-ii

Frawley, W. et al, 1992. Knowledge discovery in databases: an overview. *AI Magazine*, vol. 13, no. 3, pp. 57-70.

Garofalakis, M. et al, 1999. SPIRIT: Sequential Pattern Mining with Regular Expression Constraint. *Proc. Int'l Conf. Very Large Databases* (VLDB 1999), pp. 223-234.

Hipp, J. and Güntzer, U., 2002. Is pushing constraints deeply into the mining algorithms really what we want?. *SIGKDD Explorations*, vol. 4, no. 1, pp. 50-55

Levenshtein, V., 1965. Binary Codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1:8-17.

Pei, J. et al., 2001. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *Proc. 17th Int'l Conf. Data Engineering* (ICDE 2001), Heidelberg, Germany.

Pei, J. and Han, J., 2002a. Constrained frequent pattern mining: a pattern-growth view. *SIGKDD Explorations*, vol. 4, nr. 1 pp. 31-39.

Pei, J et al, 2002b. Mining Sequential Patterns with Constraints in Large Databases. *Proc. ACM Conf on Information and Knowledge Management* (CIKM) pp. 18-25.

Searls, D.B., 1992. The Linguistics of DNA, *American Scientist*, 80, pp. 579-591.

Srikant, R. and Agrawal, R., 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. *Proc. 5th Int'l. Conf. Extending Database Technology* (EDBT 96), pp. 3-17.

Wu, S. and Manber, U, 1992. Fast Text Searching Allowing Errors. *Communications of ACM*, 35:83—91.