Mining Patterns Using Relaxations of User Defined Constraints

Cláudia Antunes and Arlindo L. Oliveira

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

Abstract. The main drawbacks of sequential pattern mining have been its lack of focus on user expectations and the high number of discovered patterns. However, the solution commonly accepted – the use of constraints – approximates the mining process to a hypothesis-testing task. In this paper, we propose a new methodology to mine sequential patterns, keeping the focus on user expectations, without compromising the discovery of unknown patterns. Our methodology is based on the use of constraint relaxations, and it consists on using them to filter accepted patterns during the mining process. We propose a hierarchy of relaxations, applied to constraints expressed as context-free languages.

1 Introduction

Sequential Pattern Mining addresses the discovery of existing maximal frequent sequences in a given database. This type of structure appears 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.

In general, we can see sequential pattern mining as an approach to perform inter-transactional analysis, being able to deal with sequences of sets of items. Indeed, it was motivated by the need to perform that kind of analysis, mostly in the retailing industry, but with applications in other areas like the medical domain. The problem was first introduced by Agrawal and Srikant, and in the last years, several sequential pattern mining algorithms were proposed [11], [13], [9]. Despite the reasonable efficiency of those algorithms, the lack of focus and user control has been prohibitive for the generalized use of sequential pattern mining. Indeed, the large number of discovered patterns makes the analysis of discovered information a difficult task.

In order to solve this problem, several authors have promoted the use of constraints to represent background knowledge and to filter the patterns of interest to the final user. This approach has been widely accepted by the data mining community, since it allows the user to control the mining process and reduces the search space, which contributes significantly to achieve better performance and scalability levels.

The simplest constraint over the sequence content is to impose that only some items are of interest – *item constraints*. An example of such constraint is the use of Boolean expressions over the presence or absence of items [12]. When applied to sequential pattern mining, constraints over the content can be just a constraint over the items to consider, or a constraint over the sequence of items. More recently, regular languages have been proposed [3] and used to constrain the mining process, by accepting patterns that may be accepted by a regular language. The constrained algorithms use a deterministic finite automaton (DFA) to define the regular language. Generally, a finite automaton consists of a set of states and a set of transitions from state to state that occur on symbols chosen from an alphabet. When applied to sequential pattern mining, strings (sequences of symbols) are replaced by sequences of itemsets.

Although this approach has contributed to reduce the number of discovered patterns and to match them to the user expectations, restricting the search approximates the mining process to a simple hypothesis-testing task [4]. Indeed, using constraints to filter the discovered sequential patterns prevents the discovery of unknown ones, failing to accomplish one of the main goals of data mining – the discovery of novel information This claim is even stronger when applied to sequential data, where more restrictive constraints (like regular languages) have been used. By novel information, we mean the information that is not trivially inferred from the constraint.

In this paper, we propose a new mining methodology to solve the trade-off between satisfying user expectations (by using background knowledge) and mining novel information. Our methodology is based on the use of constraint relaxations, and it assumes that the user is responsible for choosing the strength of the restriction used to constrain the mining process. We propose a hierarchy of constraint relaxations (for constraints expressed as formal languages – either regular or context-free), from conservative to non-conservative relaxations.

After a small description of the use of context-free languages to deal with sequences of itemsets (section 2), the new methodology is defined (section 3), presenting each of the relaxations (including the extension of the ones proposed in [3] – Naïve, Legal and Valid). In section 4, we evaluate the use of constraint relaxations, comparing the number of discovered patterns and the processing times for each relaxation, when mining a synthetic and a real-life dataset. Section 5 concludes the paper with some discussion and ideas for future work.

2 Context-free Languages for Sequences of Itemsets

Recent work [1] has shown that regular expressions can be substituted by context-free languages, without compromising the performance of algorithms, when dealing with strings of items. This is useful because context-free languages are more expressive than regular languages, being able to represent constraints that are more interesting. In particular, the structure of constrained sequential pattern mining algorithms does not need any change to use context-free languages as constraints. The only adaptation is the substitution of the finite automaton by a pushdown automaton

(PDA), to represent the context-free language.

> A *pushdown automaton* is a tuple $\mathcal{M}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, where: Q is a finite set of states; Σ is an alphabet called the input alphabet; Γ is an alphabet called the stack alphabet; δ is a mapping from $Q\times\Sigma\cup\{\varepsilon\}\times\Gamma$ to finite subsets of $Q\times\Gamma^*$; $q_0\in Q$ is the initial state; $Z_0\in\Gamma$ is a particular stack symbol called the start symbol, and $F\subseteq Q$ is the set of final states [6].

The language accepted by a pushdown automaton is the set of all inputs for which some sequence of moves causes the pushdown automaton to empty its stack and reach a final state.

When applied to the process of mining sequential patterns from sequences of itemsets instead of strings (sequences of symbols), pushdown automata have to be redefined. The problem is related with the fact that existent algorithms manipulate one item per iteration, instead of an entire itemset. In this manner, we need to perform partial transitions, corresponding to the item involved at the specific step iteration. To illustrate this situation consider the pushdown automaton defined over itemsets represented in Fig. 1 (left). This PDA generates sequences with the same number of baskets (*a*,*b*) on the left and right side of *c*, which means that it generates

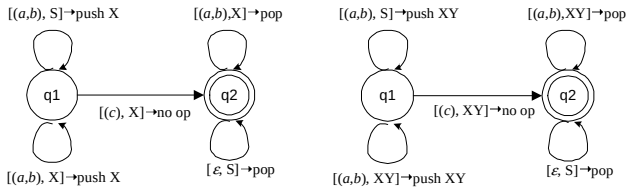

Fig. 1 Pushdown (left) and Extended Pushdown (right) automata

sequences like (*a*,*b*)*c*(*a*,*b*) or (*a*,*b*)(*a*,*b*)*c*(*a*,*b*)(*a*,*b*). Formally, it can be defined as the tuple $\mathcal{M}=(Q, \Sigma, \Gamma, \delta, q_1, S, \mathcal{F})$, with $Q=\{q_1, q_2\}$ the set of states, $\Sigma=\{a, b, c\}$ its alphabet, $\Gamma=\{S, X\}$ the stack alphabet, q_1 the initial state, *S* the initial stack symbol and $\mathcal{F}=\{q_2\}$ the set of final or accepting states. Finally, δ corresponds to the five transitions illustrated in Fig. 1-left (for example "[(*a*,*b*),*S*]→push*X*" represents the transition from state q_1 to state q_2, when the stack has the symbol *S* in the top and we are in the presence of (*a*,*b*)).

Consider for example that algorithm *PrefixGrowth* [10] is applied and it finds *a*, *b* and *c* as frequent. Then it will have to proceed to discover which items are frequent after *a*. At this point, there is already one problem: given that it has found *a*, which operation should it perform over the stack? If it pushes *X*, then *c* will be accepted after *a*, but if it only applies the push operation after finding *b*, then it will accept, as "potentially accepted", sequences like *aaa*, *aaaaa* and so on, since *S* remains on the top of the stack.

In order to deal with itemsets, we extend the notion of PDA.

> An *extended pushdown automaton* (ePDA) is a pushdown automaton $\mathcal{E}=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \mathcal{F})$, where δ is a mapping function from $Q\times\mathcal{P}(\Sigma)\cup\{\varepsilon\}\times\Gamma^*$ to finite subsets of $Q\times\Lambda^*$, with Λ equal to Γ^* and $\mathcal{P}(\Sigma)$ representing the powerset of Σ

The difference to standard pushdown automata is the transition function, which manipulates itemsets and strings of stack elements instead of items and stack elements, respectively. With this extension, it is possible to explore sequences of itemsets with existing algorithms. Fig. 1-right illustrates an extension to the PDA illustrated before. Clearly, on one hand, by using extended pushdown automata, algorithms such as *SPIRIT* or *PrefixGrowth* do not need any alteration on their structure. On the other hand, their performances remain tightly connected to the number of discovered patterns and almost nothing related to the complexity of the constraint. Because the lack of space, those results are not presented in this paper.

3 Constraint Relaxations

While the problems of representing background knowledge in sequential pattern mining and the reduction of the number of discovered patterns can be solved using formal languages, the challenge of discovering unknown information, keeping the process centered on user expectations, remains open. At this point, it is important to clarify the meaning of some terms. By **novel information**, we mean both the information that cannot be inferred in the reference frame of the information system or of the user himself. **Centering the process in the user** has essentially two aspects: the management of user expectations and the use of user background knowledge in the mining process. By expectation management, we mean that the results from the process have to be in accordance with user expectations, with similarity measured by comparing them to the user's background knowledge.

Considering these notions, we propose a new methodology to mine unknown patterns, while keeping the process centered on the user. This methodology is based on the use of constraint relaxations, instead of constraints themselves, to filter the discovered patterns during the mining process. The notion of constraint relaxation has been widely used when real-life problems are addressed, and in sequential pattern mining, they were first used to improve the performance of the algorithm [3].

A *constraint relaxation* can be seen as an approximation to the constraint. While the constraint represents the known information about the domain in analysis, the relaxation encapsulates a method to extend that knowledge. Since they do not represent existing knowledge, they cannot be simple constraints.

When used instead of the constraint, relaxations can give rise to the discovery of unknown information that will approximately match user expectations. If these relaxations are used to mine new patterns, instead of simply used to filter the patterns that satisfy the imposed constraint, the discovery of unknown information is possible. Given that 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 novel and unknown information. In this manner, the goal of data mining will be achieved. Additionally, some of the unresolved challenges of pattern mining will be addressed, namely: how to use constraints to specify background knowledge and user expectations; how to reduce the number of discovered patterns by constraining the search space, and how to reduce the amount of time in processing the discovery.

In order to achieve those results, we propose four classes of relaxations over constraints expressed as context-free languages, as illustrated in Fig. 2. Additionally, each of these relaxations can be combined, creating compositions of relaxations, imposing particular filters. Examples of such compositions are approximate-legal or non-approximate.

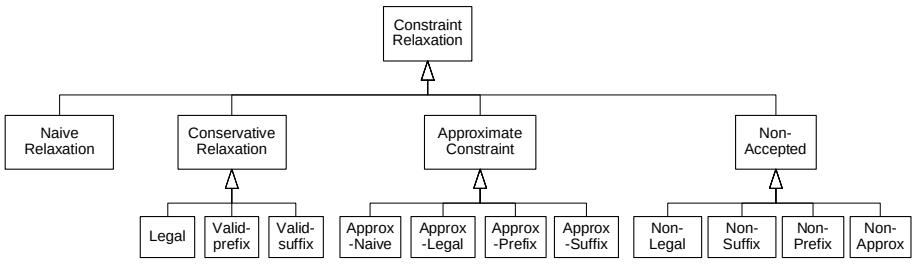

Fig. 2 Hierarchy of constraint relaxations

The first class of relaxations is the Naïve relaxation used in SPIRIT. Conservative relaxations group the other two already known relaxations (Legal and Valid-Suffix) and a third one – Valid-Prefix. Approximate and Non-Accepted Relaxations are two novel non-conservative relaxations, which introduce the ability to deal with unknown models.

3.1 Conservative Relaxations

Conservative relaxations group the *Legal* and *Valid* relaxations, used in SPIRIT, and a third one – *Valid-Prefix*, complementary to the *Valid* relaxation (called Valid-Suffix in the rest of the paper). These relaxations impose a weaker condition than the original constraint, accepting patterns that are subsequences of accepted sequences. When used in conjunction with context-free languages, those relaxations remain identical, but we have to redefine the related notions.

First of all consider the partial relation ψ, which maps from $Q\times\Sigma\times\Gamma^*$ to $Q\times\Lambda^*$ representing the achieved state $q\in Q$ and top of the stack $\lambda\in\Lambda^*$ (with Λ equal to Γ^*), when in the presence of a particular sequence $s\in\Sigma$ in a particular state $q\in Q$ and a string of stack symbols $w\in\Gamma^*$. Also, consider that λ.top is the operation that returns the first element on λ.

$\psi(q_i, s=<s_1\ldots s_n>, w)$ is defined as follows:

i. (q_i, λ), *if* $|s|=0 \wedge \exists\, \lambda\in\Lambda^*: \lambda=w$

ii. (q_j, λ), *if* $|s|=1 \wedge \exists\, q_j\in Q; \lambda\in\Lambda^*: \delta(q_i,s_1,w) \supset (q_j, \lambda)$

iii. $\psi(q_j,<s_2\ldots s_n>,\lambda.top)$, *if* $|s|>1 \wedge \exists\, q_j\in Q; \lambda\in\Lambda^*: \delta(q_i,s_1,w) \supset (q_j,\lambda)$

Additionally, consider that the elements on each itemset are ordered lexicographically (as assumed by sequential pattern mining algorithms). In this manner, it is possible to define two new predicates:

Given two itemsets $a=(a_1\ldots a_n)$ and $b=(b_1\ldots b_m)$, with n<m: a is a *prefix* of b if for all $1\le i\le n$ a_i is equal to b_i and a is a *suffix* of b if for all $1\le i\le n$ a_i is equal to $b_{i+(m-n)}$

Legal. The *Legal* relaxation requires that every sequence is legal with respect to some state of the automaton, which specifies the constraint language. The extension of legal relaxation to context-free languages is non-trivial, since the presence of a stack (on the automaton) makes the identification of legal sequences more difficult. However, it is possible to extend the notion of legality of a sequence with respect to any state of a pushdown automaton.

A sequence $s=\langle s_1 \ldots s_n \rangle$ is *legal with respect to state* q_i with the top of the stack w, iff

i. $|s|=1 \wedge \exists\, s_k \in \Sigma^*; q_j \in Q; \lambda \in \Lambda^*: \delta(q_i, s_k, w) \supset (q_j, \lambda) \wedge s_1 \subseteq s_k$

ii. $|s|=2 \wedge \exists\, s_k, s_k' \in \Sigma^*; \lambda, \lambda' \in \Lambda^*; q_j, q_j' \in Q: \delta(q_i, s_k, w) \supset (q_j', \lambda) \wedge s_1$ suffixOf s_k $\wedge\, \delta(q_j', s_k', \lambda.top) \supset (q_j, \lambda') \wedge s_2$ prefixOf s_k'

iii. $|s|>2 \wedge \exists\, s_k, s_k' \in \Sigma^*; \lambda, \lambda', \lambda'' \in \Lambda^*; q_j, q_j', q_j'' \in Q: \delta(q_i, s_k, w) \supset (q_j', \lambda) \wedge s_1$ suffixOf s_k $\wedge\, \psi(q_j', s_2 \ldots s_{n-1}, \lambda.top) = (q_j'', \lambda') \wedge \delta(q_j'', s_k', \lambda'.top) \supset (q_j, \lambda'') \wedge s_n$ prefixOf s_k'

This means that any sequence with one itemset is legal with respect to an extended pushdown automaton state, if there is a transition from it, defined over a superset of the itemset (i). When the sequence is composed of two itemsets, it is legal with respect to a state, if the first itemset is a suffix of a legal transition from the current state, and the second itemset is a prefix of a legal transition from the achieved state (ii). Otherwise, the sequence is legal if the first itemset is a suffix of a legal transition from the state, and the last one is a prefix of a legal transition from the state reached with $s_2 \ldots s_{n-1}$. Using the PDA in Fig. 3, (*ab*)*ca* or *bca* are examples of legal sequences. Note that ψ is only defined for non-empty stacks. Indeed, in order to verify the legality of some sequence *s*, it is necessary to find a sequence of itemsets *t* that can be concatenated to *s*, creating a sequence *ts* accepted by the automata.

Valid-Suffix. The *Valid-Suffix* relaxation only accepts sequences that are valid suffixes with respect to any state of the automaton. Like for legal relaxation, some adaptations are needed when dealing with context-free languages.

A sequence $s=\langle s_1 \ldots s_n \rangle$ is a *valid-suffix with respect to state* q_i with top of the stack w, iff

i. $|s|=1 \wedge \exists\, s_k \in \Sigma^*; \lambda \in \Lambda^*; q_j \in Q: \delta(q_i, s_k, w) \supset (q_j, \lambda) \wedge s_1$ *suffixOf* $s_k \wedge \lambda.top = \varepsilon$

ii. $|s|>1 \wedge \exists\, s_k', s_k'' \in \Sigma^*; \lambda', \lambda'' \in \Lambda^*; q_j, q_j', q_j'' \in Q: \delta(q_i, s_k, w) \supset (q_j', \lambda) \wedge s_1$ *suffixOf* s_k $\wedge\, \psi(q_j', s_2 \ldots s_n, \lambda.top) = (q_j, \lambda') \wedge \lambda.top = \varepsilon$

This means that a sequence is a valid-suffix with respect to a state if it is legal with respect to that state, achieves a final state and the resulting stack is empty. In particular, if the sequence only has one itemset, it has to be a suffix of a legal transition to an accepting state. For the same example, *b* and *bc*(*ab*) are examples of valid-suffixes.

Note that, in order to generate valid-suffix sequences with respect to any state, it is easier to begin from the final states. In order to avoid this difficulty, using prefix instead of suffix validity could represent a more useful relaxation, when dealing with context-free languages. Note that valid-suffixes are not prefix-monotone, and could not be easily used by pattern-growth methods [9].

Valid-Prefix. The valid-prefix relaxation is the counterpart of valid-suffix, and requires that every sequence is legal with respect to the initial state.

A sequence $s=<s_1 \ldots s_n>$ is said to be *prefix-valid* iff:

i. $|s|=1 \wedge \exists s_k \in \Sigma^*; \lambda \in \Lambda^*: \delta(q_0,s_k,Z_0) \supset (q_j,\lambda) \wedge s_1$ *prefixOf* s_k

ii. $|s|>1 \wedge \exists s_k \in \Sigma^*; \lambda,\lambda' \in \Lambda^*; q_j,q_j' \in Q: \psi(q_0,s_1 \ldots s_{n-1},Z_0)=(q_j',\lambda') \wedge \delta(q_j',s_k,\lambda'.top) \supset (q_j,\lambda) \wedge s_n$ *prefixOf* s_k

This means that a sequence is prefix-valid if it is legal with respect to the initial state and the first itemset is a prefix of a transition from the initial state. Sequences with valid prefixes are not difficult to generate, since the simulation of the stack begins with the initial stack: the stack containing only the stack start symbol. The benefits from using the suffix-validity and prefix-validity are similar. When using the prefix-validity to generate the prefix-valid sequences with *k* elements, the frequent *k*-1-sequences are extended with the frequent 1-sequences, in accordance with the constraint. Examples of valid-prefixes are (*ab*) and (*ab*)*ca*.

Note that the legal relaxation accepts all the patterns accepted by valid-suffix and valid-prefix relaxations. In this manner, it is a less restrictive relaxation than the other two. Although these relaxations have considerable restrictive power, which improves significantly the focus on user expectations, they do not allow for the existence of errors. This represents a strong limitation in real datasets, since little deviations may exclude many instances from the discovered patterns.

3.2 Non-Conservative Relaxations

Non-conservative relaxations permit to discover patterns that are not subsequences of accepted patterns, by accepting any sequences with a specific alphabet (*Naïve*), by allowing some errors (*Approx*) or just by considering only the sequences that are not accepted by the constraint (*Non-Accepted*).

Naïve Relaxation. The first class of non-conservative relaxations is the Naïve relaxation, which corresponds to a simple item constraint. For example, in the context of the SPIRIT algorithm, it only accepts patterns containing the items that belong to the language alphabet. However, this relaxation prunes a small number of candidate sequences, which implies a limited focus on the desired patterns. Any sequence composed of items *a*, *b* and *c* would be accepted by naïve relaxation.

Approximate Constraints. Approximate matching at a lexical level has been considered an extremely important tool to assist in the discovery of new facts, but ignored in most of the approaches on pattern mining. It mainly consists of considering two sequences similar if they are at an edit distance below a given threshold. An exception to this generalized frame is *AproxMAP* [6], which uses this distance to count the support for each potential pattern. However, to our knowledge, edit distance has not been applied to constrain the pattern mining process.

To address the need to identify approximate matching we propose a new class of relaxations – the *Approx Constraints*, 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. In other words, *approx-constraints* only accept sequences that are at a given edit distance for an accepted sequence. This edit distance reflects

the cost of operations that have to be applied to a given sequence, so that 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 sequences, and the operations to consider can be the *Insertion*, *Deletion* and *Replacement* [8].

> Given a constraint C, expressed as a context-free language, and a real number ε which represents the maximum error allowed, a sequence *s* is said to be *approximate-accepted* by C, if its generation cost ξ(s, C) is less than or equal to ε. The *generation cost* ξ (s, C) is defined as the sum of costs of the cheapest sequence of edit operations transforming the sequence s into a sequence r accepted by the language C.

Examples of approximate accepted sequences with one error would be *ac*(*ab*) by the deletion of a *b* in the first itemset and (*abc*)*c*(*ab*) by the insertion of a *c* in the first itemset.

The other four classes of approximate constraints are defined by replacing the acceptance by legality and validity notions. In this manner, an *Approx-Legal* relaxation accepts sequences that are approximately legal with respect to some state. *Approx-Suffix* and *Approx-Prefix* relaxations are defined in a similar way. Finally, *Approx-Naïve* accepts sequences that have ε items (with ε the maximum error allowed) that do not belong to the language's alphabet.

Recent work has proposed a new algorithm ε–accepts [2] to verify if a sequence was approximately generated by a given deterministic finite automata (DFA). Fortunately, the extension to deal with context-free languages is simply achieved by replacing the use of a DFA by the use of an ePDA.

Non-accepted Relaxation. Another important issue is related with the discovery of low frequency behaviors that are still very significant to the domain. Fraud detection is the paradigm of such task. Note that the difficulties in fraud detection are related with the explosion of discovered information when the minimum support decreases.

Suppose that there is a model (expressed as a context-free language) able to describe the frequent patterns existent on a huge database (say for example that the minimum support allowed is 10%). If there are 3% of clients with a fraudulent behavior, it is possible that they are not discovered neither by using the unconstrained mining process, neither by using any of the proposed relaxations. However, the model of non-fraudulent clients may be used to discover the fraudulent ones: the fraudulent clients are known to not satisfy the model of non-fraudulent clients.

To address the problem of low frequency behaviors discovery, we propose an additional class of relaxations – the *Non-accepted relaxation*. If L is the language used to constrain the mining process, Non-accepted relaxations will only accept sequences that belong to the complementary language of L.

> A sequence $s=<s_1 \ldots s_n>$ is said to be *non-accepted* by the language if it is not generated by that language.

In fact, this is not really a relaxation, but another constraint (in particular the constraint that only accepts sequences that belong to the language that is the complement of the initial constraint). However, since they are defined based on the initial constraint, we choose to designate them as relaxations.

The benefits from using the non-accepted relaxation are mostly related to the possibility of not rediscovering already known information, which may contribute significantly to improve the performance of sequential pattern mining algorithms. Moreover, since context-free languages are not closed under complementation [6] (which means that the complement of a context-free language is not necessarily a context-free language), the use of the complement instead of the non-accepted relaxation could be prohibitive.

Note that using this new approach, it is possible to reduce the search space, and consequently to reduce the minimum support allowed. The non-accepted relaxation will find all the patterns discovered by the rest of the introduced relaxations, representing a small improvement in the focus on user expectations. In fact, it finds all the patterns discovered by unconstrained patterns minus the ones that are accepted by the constraint. Like for approx relaxations, an interesting improvement is to associate a subset of the alphabet in conjunction with non-accepted relaxation. This conjunction focus the mining process over a smaller part of the data, reducing the number of discovered sequences, and contributing to achieve our goal.

As before, the sub-classes of Non-Accepted relaxations result by combining the non-acceptance philosophy with each one of the others relaxations. While non-accepted relaxation filters only a few patterns, when the constraint is very restrictive, the non-legal relaxation filters all the patterns that are non-legal with respect to the constraint. With this relaxation is possible to discover the behaviors that completely deviate from the accepted ones, helping to discover the fraudulent behaviors.

3.3 Discussion: novelty and expectedness

The discussion about the concept of novel information is one of the most difficult in pattern mining. While the concept is clear in the reference frame of a knowledge acquisition system, the same is not true in the reference frame of the final user. Indeed, several interestingness measures have been proposed for the evaluation of the discovered patterns [4].

Moreover, this issue is more critical with the introduction of constraints in the mining process. In fact, in the presence of constraints the concept of novel patterns becomes unclear even in the reference frame of information systems, since they are then able to deal with that knowledge, represented as the constraint.

In order to bring some light into the discussion, consider that, given a model C as constraint, a pattern A is *more novel than* a pattern B, if the generation cost of A in order to C is larger than the generation cost of B in order to C (with the generation cost defined in the previous section). With this concept, it is now possible to understand the reason why non-accepted patterns can be more novel than the patterns discovered by conservative relaxations. It is now clear that, despite the differences between relaxations, all of them allow for the discovery of novel information. Indeed, the conservative relaxations are able to discover failure situations, this is, situations when for, some reason, the given model is not completely satisfied (Valid- Prefix and Valid-Suffix identify failures in the beginning and ending of the model, respectively, and Legal identifies problems in the middle of the model).

However, the great challenge of pattern mining is to discover novel information in accordance to user expectations. It is clear from the definition of the novel relation, that an unexpected pattern is more novel than an expected one. In fact, the challenge resides in the balance between the discovery of novel but expected patterns. The proposed relaxations cover a wide range of this balance, giving the user the option of which is the most relevant issue for the problem in hands: to discover novel information or to satisfy user expectations.

4 Experimental Results

In this section, our goal is to validate the claim that the use of relaxations enables the discovery of unknown information, keeping the mining process centered on the user and to demonstrate that the use of context-free languages does not impair the performance of the process of sequential pattern mining. In order to achieve this goal we have applied this new approach to discover the common sequences of subjects under a graduate program in computer science. In particular, we have applied the different relaxations to identify the common curricula in three different tasks: discovery of frequent patterns inside each scientific area, determination of which are the optional courses chosen by different students and identification of the sequences of subjects of students in specialty areas with few enrollments. While the two last tasks were accomplished with the use of a regular language that specify the curricula established in the graduate program, the first one was accomplished by using a context-free language that filters the sequences of subjects in each scientific area, where students have failed at most once (Fig. 3).

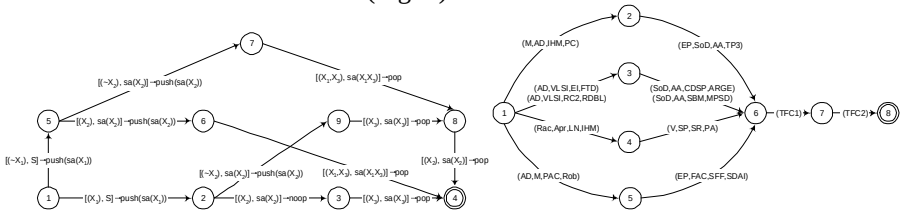

Fig. 3 PDA[1] used in the first task (left) and DFA used in the second task (right)

As expected, by using the constraints directly, we only discover a few patterns, which constitute already known information, since they satisfy the imposed languages. Therefore, these results are not enough to invalidate Hipp's arguments [4].

However, the situation is different in the presence of relaxations. First, the use of the *approx* relaxation allowed the discovery of which students choose each optional

[1] In this graph, each transition represents four transitions, one per scientific area. For example, [(~X2),sa(X2)]→push(sa(X2)) represents [(~F1),F]→push(F), [(~AM2),AM]→push(AM) [(~AED),MTP]→push(MTP), [(~AC),ASO]→push(ASO).. X_1, X_2 and X_3 represent the first, second and third subjects on each scientific area; sa is a function from the set of subjects to their scientific area, for example sa(F1)=F and sa(AM1)=AM.

subjects, with students only identified by their own curricula. Indeed, it was possible to discover that students in different specialty areas chose different subjects, and that students at a specific specialty area when have failed in at least one subject in the fourth curricular year, choose two subjects on Management as their optional subjects. This last discovery is very hard to achieve with the use of queries or constraints because all non-common subjects can be chosen as optional by some student. A simple count of each subject support does not give the expected answer, since most of the subjects are required to some percentage of students. The approach of querying the dataset to count the support of each subject, knowing that students have followed some given curriculum, is also unable to answer the question, since a considerable number of students (more than 50%) have failed one or more subjects, following a slightly different curriculum.

The third task is also difficult to solve, and indeed, is similar to fraud detection, where just a small percentage of total entities have a specific behavior. The use of constraints does not help, since we do not know the model behind the behaviors, but the use of unconstrained pattern mining is not possible because of the extremely large number of discovered patterns. The identification of the common curricula of specialty area with few students (IIN and IAR) is one of these cases, since have 13% of students, and like the others, less than 50% of those have not failed in any subject. The use of *Non-Accepted* relaxations discovers those curricula if we use a constraint that specifies the proposed curriculum for the other specialty areas and an item constraint corresponding to the subjects of the specialty area in analysis.

Finally, it is important to note that although the use of context-free languages increases the time spent per discovered pattern, in general, it does not increase the overall performance, as seen in Fig. 4. Additionally, it is clear that the use of non-conservative relaxations enables the discovery of more patterns than conservative ones, but that the number of patterns is still acceptable when compared with the

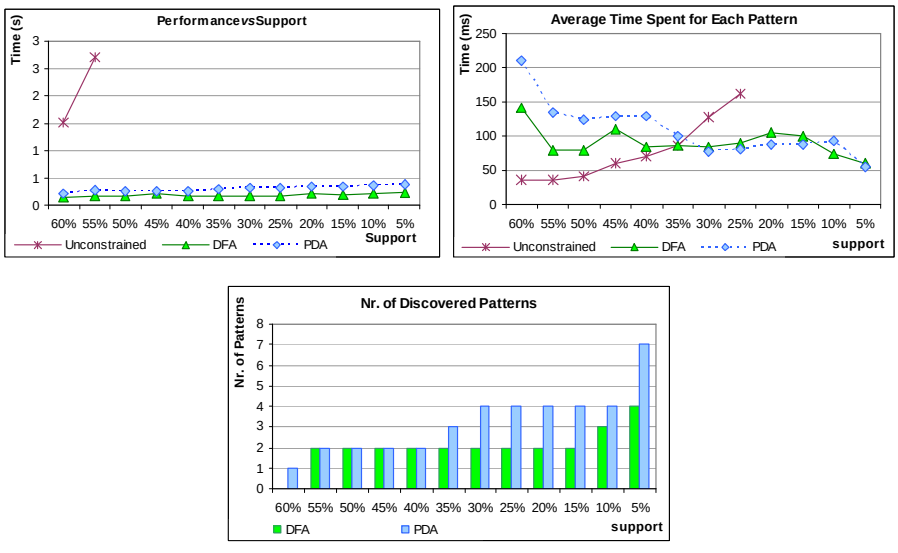

Fig. 4. Comparison of the impact of the use of DFAs and PDAs

number of discovered patterns by unconstrained processes.

5 Conclusions

In this paper, we show that the use of constraint relaxations allows for the discovery of novel information, centered on user expectations, when mining sequential patterns. A constraint relaxation imposes a weaker restriction, relaxing the constraint chosen to represent user background knowledge. Experimental results show that the use of relaxations reduces the number of discovered patterns when compared to unconstrained processes, but enables the discovery of unexpected patterns, when compared to constrained processes. Additionally, they show that the processing times spent in the mining process can be reduced if constraint relaxations are used. The experiments also show that the use of relaxations is of great help when there is not a precise knowledge about the behaviors shown.

One important challenge is to apply this methodology to the extraction of intra-transactional patterns, where there are no constraints to specify the structure of the transactions.

References

1. Antunes, C. and Oliveira, A.L., "Inference of Sequential Association Rules Guided by Context-Free Grammars", in *Int. Conf. Grammatical Inference*, Springer (2002) 1-13
2. Antunes, C. and Oliveira, A.L., "Sequential Pattern Mining with Approximated Constraints", *Int. Conf Applied Computing*, IADIS (2004) 131-138
3. Garofalakis, M., Rastogi, R. and Shim, K., "SPIRIT: Sequential Pattern Mining with Regular Expression Constraint", in *Int. Conf. Very Large Databases*, Morgan Kaufmann (1999) 223-234,
4. Hilderman, R and Hamilton, H., "Knowledge discovery and interestingness measures: a survey", *Technical Report* CS 99-04, Dep. Computer Science, University of Regina, 1999.
5. Hipp, J. and Güntzer, U., "Is pushing constraints deeply into the mining algorithms really what we want?". *SIGKDD Explorations*, vol. 4, no. 1, ACM (2002) 50-55
6. Hopcroft, J. and Ullman, J., *Introduction to Automata Theory, Languages and Computation*. Addison Wesley. 1979.
7. Kum, H.-C., Pei, J., Wang, W. and Duncan, D., "ApproxMAP: Approximate Mining of Consensus Sequential Patterns", in *Int. Conf on Data Mining*, IEEE (2003).
8. Levenshtein, V., "Binary Codes capable of correcting spurious insertions and deletions of ones", in *Problems of Information Transmission*, 1, Kluwer (1965) 8-17
9. Pei J, Han J et al: "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", in *Int. Conf Data Engineering*, IEEE (2001), 215-226
10. Pei, J., Han, J. and Wang, W., "Mining Sequential Patterns with Constraints in Large Databases", in *Conf Information and Knowledge Management*, ACM (2002) 18-25
11. Srikant R, Agrawal R.: "Mining Sequential Patterns: Generalizations and Performance Improvements", in *Int. Conf Extending Database Technology*, Springer (1996) 3-17
12. Srikant R, Agrawal R, "Mining association rules with item constraints" in *Int. Conf. Knowledge Discovery and Data Mining*, ACM (1997) 67-73
13. Zaki, M."Efficient Enumeration of Frequent Sequences", in *Int. Conf. Information and Knowledge Management*, ACM (1998) 68-75