Constraint Relaxations for Discovering Unknown Sequential Patterns

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 verification of what are the frequent patterns among the specified ones, instead of the discovery of unknown and unexpected patterns.
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, classifying the existing relaxations (*legal*, *valid* and *naïve*, proposed in SPIRIT [3]), and proposing several new classes of relaxations, ranging from the *approx* and *non-accepted*, to the composition of different types of relaxations, like the *approx-legal* or the *non-prefix-valid* relaxations. At last, we present a case study that show the results achieved with the application of this methodology on the analysis of the curricular sequences of computer science students.

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. Sequential pattern mining was motivated by the need to perform this 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 hampered the generalized use of sequential pattern mining. In general, 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 only patterns that are accepted by a regular language. In this case, constrained algorithms use a deterministic finite automaton (DFA) to define the regular language. Generally, these automata consist 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, the use of regular languages transforms the pattern mining process into the verification of which of the sequences of the language are frequent, completely avoiding the discovery of novel patterns. It is important to note that by specifying the language, we are explicitly specifying which are the sequences that can be considered, and the mining process is reduced to the identification of which of these are frequent. By novel information, we mean the information that is not trivially inferred from the constraint.

1.1 Motivation

Consider, for instance, the problem of identifying typical behaviors of some company customers. Suppose that the company considers that a well-behaved customer is a customer who has made all its payments at the time of the analysis. This knowledge can be represented by a context-free language, but cannot be by a regular language, since it uses a count of the number of invoices and payments that has to be respected.

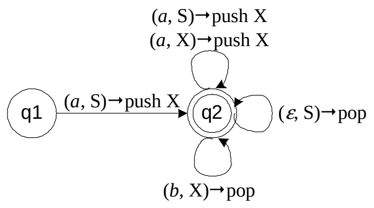


Fig. 1 Pushdown automaton for representing well-behaved customers

The pushdown automaton presented in Fig. 1 (pushdown automata are described in the next section) is able to represent that knowledge, with *a* corresponding to an invoice and *b* to a payment, and in conjunction with sequential pattern mining algorithms allows for the discovery of each sequence of invoices and payments are frequent (sequences like *abab*, *aaabbb* and *aabbab*).

However, if among the recorded data there are events related to second copies of invoices, cancellations, information requests and answers to requests, for example, and there is no knowledge about their relations with the other two kinds of events, it is not possible to discover those relations with constrained sequential pattern mining. On the other hand, with an unconstrained process the existing background knowledge will be ignored. In this manner, it is clear that although constraints (formal languages in particular) can be used to represent existing domain knowledge, they are not enough to address the main data mining challenge: to discover novel information.

In this work, 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), that range from conservative to non-conservative relaxations, proposing two new types of constraints – the *approx* and the *non-accepted* relaxations, and new relaxations resulting from the composition of the different classes of relaxations.

After a concise 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 present a case study with the analysis of curriculum sequences, and, based on that data, we evaluate the use of constraint relaxations, by comparing the number of discovered patterns and the processing times using each relaxation. Section 5 concludes the paper with a 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 existing 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. 2 (left). This PDA generates sequences with the same number of baskets (*a*,*b*) on the left and right side of item *c*, which means that it generates

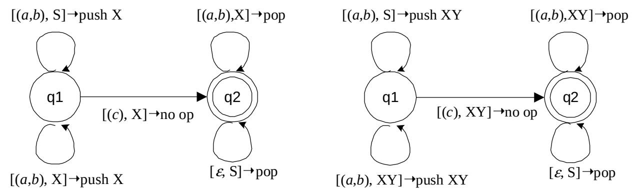


Fig. 2 Pushdown (left) and Extended Pushdown (right) automata

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 $\mathcal{M}$=($\mathcal{Q}$, Σ, Γ, δ, q_1, *S*, $\mathcal{F}$), with $\mathcal{Q}$={q_1, q_2} the set of states, Σ={ *a*, *b*, *c*} its alphabet, Γ={*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. 2-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 tuple $\mathcal{E}$=($\mathcal{Q}$, Σ, Γ, δ, q_0, Z_0, $\mathcal{F}$),), with $\mathcal{Q}$, Σ, Γ, q_0, Z_0 and $\mathcal{F}$ defined as for pushdown automata, but δ defined as a mapping function from $\mathcal{Q}\times\mathcal{P}(\Sigma)\cup\{\varepsilon\}\times\Gamma^*$ to finite subsets of $\mathcal{Q}\times\Lambda^*$, with Λ equal to Γ* and $\mathcal{P}$ (Σ) 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. 2-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.

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, and **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 to user expectations, with similarity measured by comparing them to the user's background knowledge.

In the case of using sequential pattern mining with constraints expressed as context-free languages, it is clear that:

- the existing background knowledge is represented and integrated in the mining process as a context-free language;
- the process is completely centered on the user, since the process will only discover patterns that are in accordance to his background knowledge, this is, sequences that belong to the context-free language;
- it is not possible to discover novel information, since all the discovered patterns can be directly inferred from the user’s reference frame – expressed by the language.

Considering these limitation, 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. In other words, while constraints explicitly specify which sequences can be discovered, constraint relaxations determine the way that the knowledge (expressed by the constraint) can be used to guide the mining process. In this manner, when used instead of constraints, relaxations can give rise to the discovery of novel information, in the sense that the patterns discovered are no longer directly inferred from the background knowledge, i.e. from the reference frame of the system. 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 novel information is possible. Given that the user may choose the level of relaxation allowed (the type of relaxation), 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 main classes of relaxations over constraints expressed as formal languages, as illustrated in Fig. 3. The differences between them result from the redefinition of the acceptability notion for the formal language that defines the constraint.

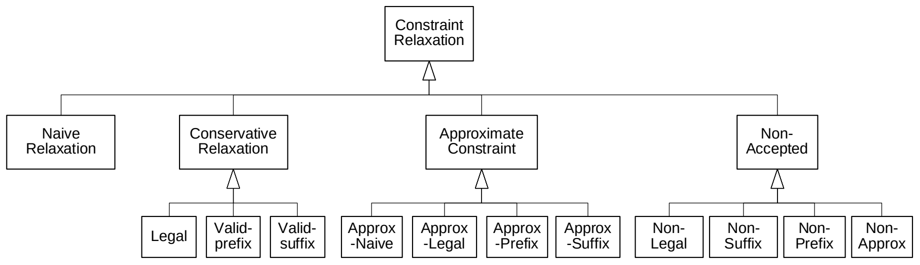


Fig. 3 Hierarchy of constraint relaxations

The first class of relaxations is the Naïve relaxation, which corresponds to a simple item constraint. However, in the context of constraints expressed as formal languages, it can be seen as a relaxation that only accepts patterns containing the items that belong to the language alphabet.

Conservative relaxations group the other already known relaxations, used by SPIRIT [3], and a third one – Valid-Prefix, similar to Valid-suffix.

It is important to note that conservative relaxations are not able to discover unknown patterns, just sub-patterns of expected ones. 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 to pattern mining. It considers two sequences similar if they are at an edit distance below a given threshold. An exception to this generalized frame is the *AproxMAP* [7], 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 relaxation*, which accept the patterns that are at an acceptable edit distance from some sequence accepted by the constraint.

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 threshold decreases.

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

Additionally, each of these relaxations can be combined, creating compositions of relaxations, imposing particular filters. Examples of such compositions are *approx-legal* or *non-approx*.

Next, we will present each class of constraint relaxations. To illustrate the concepts, consider the extended pushdown automaton in Fig. 2-right.

3.1 Naïve Relaxation

As stated, the *Naïve* relaxation only prunes candidate sequences containing elements that do not belong to the alphabet of the language, For example, if we consider the specified automaton, only sequences with *a*'s, *b*'s and *c*'s are accepted by the Naïve relaxation.

In this manner, a sequence is accepted by the naive criterion in exactly the same conditions than for regular languages. However, this relaxation prunes a small number of candidate sequences, which implies a limited focus on the desired patterns.

Since Naïve relaxation is anti-monotonic, no change in sequential pattern mining algorithms is needed.

3.2 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 $\mathcal{Q}\times\mathcal{S}\times\Gamma^*$ to $\mathcal{Q}\times\Lambda^*$ representing the achieved state $q\in\mathcal{Q}$ and top of the stack $\lambda\in\Lambda^*$ (with Λ equal to Γ^*), when in the presence of a particular sequence $s\in\mathcal{S}$ in a particular state $q\in\mathcal{Q}$ and a string of stack symbols $w\in\Gamma^*$. Also, consider that λ.top is the operation that returns the first string 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 $1\leq i\leq n$ a_i is equal to b_i and *a* is a *suffix* of *b* if for all $1\leq i\leq n$ 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 an extended pushdown automaton.

A sequence $s=<s_1 \dots s_n>$ 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
$\wedge\, \psi(q_j',s_2 \dots s_{n-1},\lambda.top)=(q_j'',\lambda') \wedge \delta(q_j'',s_k',\lambda'.top) \supset (q_j,\lambda'') \wedge s_n$ 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, $s_2 \dots s_{n-1}$ corresponds to a valid path from the state and stack reached, and the last itemset is a prefix of a legal transition from the state and stack reached with $s_2 \dots s_{n-1}$.

Examples of legal sequences with respect to the initial state of the specified automaton are: *a*, *b* and *c* (by rule i), *bc* and (*a*,*b*)*c* (by rule ii) and *bca* and (*a*,*b*)*ca* (by rule iii). Examples of non-legal sequences are *ac* (by ignoring rule ii) or *acb* (by ignoring rule iii).

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 $s=<s_1 \dots s_n>$ 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
$\wedge\, \psi(q_j',s_2 \dots s_n,\lambda.top)=(q_j,\lambda') \wedge \lambda.top=\varepsilon$

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.

Considering the extended pushdown automaton as before, examples of such sequences are *b*, (*a*,*b*), *c*(*a*,*b*) and *bc*(*a*,*b*). Negative examples are, for instance, *bca* or *bcb*. Note that, in order to generate valid-suffix sequences with respect to any state, it is easier to begin from the final states. However, this kind of generating process is one of the more difficult when dealing with pushdown automata, since it requires a reverse simulation of their stacks.

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;* (*a*)*c* or *bc* are examples of non-valid prefixes.

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.

3.3 Approximate Constraints

In order to solve this problem we propose a class of relaxations 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 class of relaxations, called *approx relaxations*, tries to accomplish this goal: they only accept sequences that are at a given edit distance for an accepted sequence. This edit distance is a similarity measure that 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. This cost of operations will be called the *generation cost*, and 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 $\xi(s, C)$ 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.

For example, considering the extended pushdown automaton defined above, *ac*(*a*,*b*) and (*a*,*b*)(*a*,*b*) are approx-accepted sequences with one error, which result from inserting a *b* on the first itemset, on the first example, and a *c* on the second position, on the second example, respectively. $c(a,b)$ and $b(a,b)$ are non-approximate accepted with one error, since two edit operations are needed to accept them.

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. The results shown in this paper were achieved by using such an algorithm.

3.4 Non-accepted Relaxation

Another important issue is related with the discovery of low frequency behaviors that are still very significant to the domain.

Suppose that there is a model (expressed as a context-free language) able to describe the frequent patterns existing 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, nor 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*, which accept sequences that are not accepted by the constraint.

A sequence is *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 by 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 the 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.

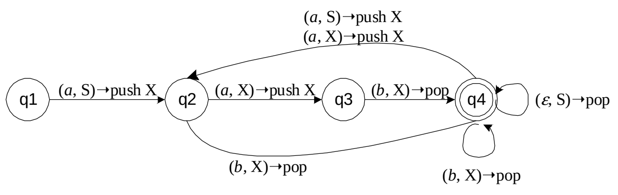
As before, the sub-classes of *Non-Accepted* relaxations result by combining the non-acceptance philosophy with each one of the others relaxations. While non-accepted relaxation filters only a few patterns, when the constraint is very restrictive, the non-legal relaxation filters all the patterns that 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.

3.5 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 above). 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.



Dataset	
eababraabb	*aababbaerb*
aabbaberab	*ababaabbab*
aebraaaccd	*aebaraacbe*
abaaaccder	*abaeraacbb*
aebaraaacb	*aaacbabab*

Fig. 4 Example of a pushdown automaton and a small dataset

Consider the PDA and the data shown in Fig. 4: the PDA represents a more restrictive notion of well-behaved costumer – a costumer who as at most two invoices to pay. If we apply the proposed methodology to analyze that data, we will be able to discover several different patterns with the different relaxations, as shown in Table 1.

Table 1 – Comparison of the results achieved with and without constraints

Frequent	Accepted	Legal	Prefix	Approx.(ε=1)	Non-Acc (w/ Σ={a,c,d,e,r}
be *baa* *era* *braa* *baba* *baer* *araa* *abab* *abaa* *raacb* *raaac* *aaacb* *aabbab* *aaaccd* *aebaraa*	*abab* *aabbab*	*baba* *abab* *abaa* *aabbab*	*abab* *abaa* *aabbab*	*abab* *aabbab* *ar*→R(*r*,*b*,2) *aeb*→D(*e*,2) *abaa*→R(*a*,*b*,4) *aacb*→R(*c*,*b*,3)	*aer* *era* *raac* *araa* *raaac* *aaaccd*

In order to permit an easy comparison, only maximal patterns are shown. In this manner, some patterns appear only in some columns. In the column relative to the *approx* relaxation, are shown the edit operations needed to transform each pattern to a pattern belonging to the context-free language.

As expected, by using the constraint itself we only discover two patterns, which satisfy the context-free language. Therefore, these results are not enough to invalidate Hipp's arguments [5] about constraints. Nevertheless, with Legal and Valid-prefixes, it is possible to discover some other intermediate patterns, which are potentially accepted by the complete constraint.

Finally, with *approx* and *non-accepted* relaxations, it is possible to discover unexpected patterns. Indeed, the approx relaxation shows the most interesting behavior, since it is possible to discover that it is usual that after sending the second invoice, customers pay their old bills (*aacb*). With non-accepted relaxation, it is also possible to discover interesting patterns. In this case, it is common that after three invoices without any payment, and the emission of two second invoices, the customer account is canceled (*aaaccd*).

4 Discovering Frequent Curricula: a case study

In this paper, we claim that it is possible to discover unknown information, using sequential pattern mining with constraint relaxations, without loosing the focus on user expectations. In order to validate these claims, we present the results achieved by applying this methodology to the discovery of frequent curricula instantiations from the data collected from IST student's performance in one undergraduate program on information technology and computer science. For this purpose, we will evaluate our methodology by comparing: the performance of each mining process; the number of discovered patterns and the ability to answer the relevant questions.

The curriculum has a duration of 10 semesters with 36 required subjects, a final thesis and 4 optional subjects in the last year. Four specialty areas are offered: PSI – Programming and Information Systems; SCO – Computer Systems; IAR – Artificial Intelligence and IIN – Information Systems for Factory Automation.

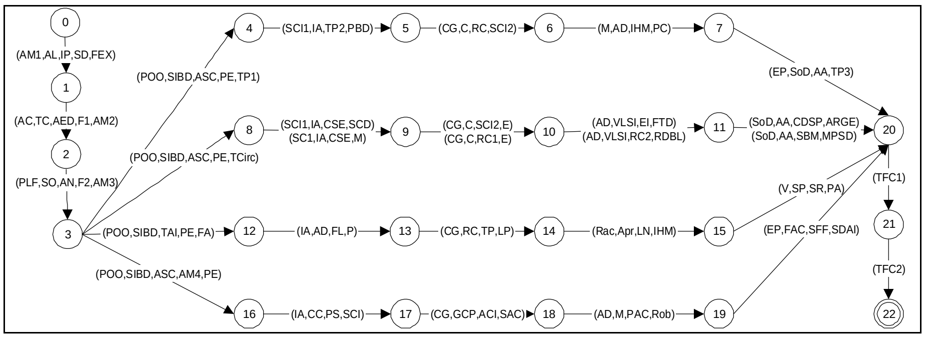


Fig. 5 DFA for specifying the model curriculum for LEIC specialty areas

There are 20 common subjects in the curriculum model, 18 on the first three semesters and the other 2 on the following two semesters. The enrollment on a specialty area was made on the fourth semester. There are 47 other subjects, distributed by each specialty area. The deterministic finite automaton on Fig. 5 shows the model curriculum for each specialty area. (The existence of two different transitions per semester for SCO students, are due to a minor reorganization of the SCO curriculum on 1995/1996.)

Data statistics. The dataset used to analyze those questions consists on the set of sequences corresponding to the curriculum followed by students that made at least 8 enrollments. In this manner, the dataset is composed of 1440 sequences, with an average sequence length equal to 11.58 semesters. Most of the students (72%) have between 8 and 12 enrollments (they had attended classes between 8 and 12 semesters). Naturally, the number of students with an odd sequence length is reduced, since this situation corresponds to students that have registered in only one semester on that year. In terms of the number of enrollments per semester, its mean is 4.82 enrollments on subjects per semester, with most students (75%) enrolling on between 4 and 6 units.

Another interesting issue is the distribution of students per specialty area: 56% follow the PSI specialty area, 19% the SCO specialty area and the remaining 26% are equally distributed by IAR and IIN specialty areas. This distribution conditions the number of enrollments per subject. For example, subjects exclusive to Artificial Intelligence and IIN have at most 13% of support.

It is interesting to note that only 823 students (57%) have concluded the final work (`TFC1` and `TFC2`). Since it is usual that students only took optional subjects in parallel or after finishing the final work, the support for optional subjects is at most 57%. Since the options are chosen from a large set of choices (130 subjects), their individual support is considerably lower. Indeed the subject on Management (G) is the optional subject with more students, about 40%.

4.1 Evaluation of Discovered Information

The analysis of the data referring to students' performance has essentially two main reasons: to explain the low levels of success and to identify the most common profiles of students. In this manner, we will try to answer three questions: 'what are the most common patterns on each scientific area?', 'what is the impact of some subjects on others?' and 'what are the common curricula instantiations for each specialty area, including optional subjects?'.

Finding frequent curricula on scientific areas. The discovery of the most common patterns on each scientific area is easily achieved if we look for students, who conclude the sequence of subjects in the same scientific area in at most four semesters. This constraint can be specified by the extended pushdown automaton represented on Fig. 6. In this figure, each transition represents four transitions, one per scientific area. For example, `[~X2,sa(X2)]→push sa(X2)` represents `[~AED,MTP]→push MTP`, `[~AC,ASO]→push ASO`, `[~AM2,AM]→push AM` and `[~F1,F]→push F`. Consider that X_1, X_2 and X_3 are the first, second and third subjects on some of the following scientific areas: `MTP`, `ASO`, `Physics` and `Mathematical Analysis`. Also, consider that `sa` is a function from the set of subjects to their scientific area, for example `sa(IP)=MTP`.

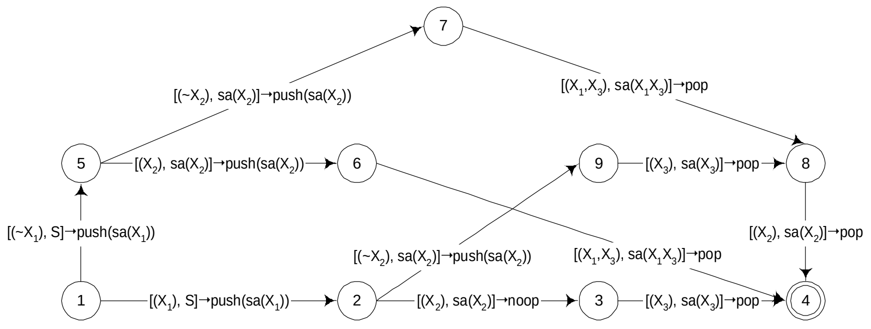

7
[(~X$_2$), sa(X$_2$)]→push(sa(X$_2$))
[(X$_1$,X$_3$), sa(X$_1$X$_3$)]→pop
5
[(X$_2$), sa(X$_2$)]→push(sa(X$_2$))
6
9
[(X$_3$), sa(X$_3$)]→pop
8
[(~X$_1$), S]→push(sa(X$_1$))
[(X$_2$), sa(X$_2$)]→pop
[(~X$_2$), sa(X$_2$)]→push(sa(X$_2$))
[(X$_1$,X$_3$), sa(X$_1$X$_3$)]→pop
1
[(X$_1$), S]→push(sa(X$_1$))
2
[(X$_2$), sa(X$_2$)]→noop
3
[(X$_3$), sa(X$_3$)]→pop
4

Fig. 6 ePDA for specifying the curricula on scientific areas, where students conclude three subjects in a specific scientific area at most on 4 semesters

The first thing that we are able to discover, using a constraint defined over the ePDA on Fig. 6 (with a gap equal to zero) is that the majority of students are able to conclude the sequence of MTP (61%) and ASO (57%) subjects without any failure.

Additionally, 6% of students are also able to conclude all but one of those subjects in four semesters (see shadowed patterns in Table 2-left).

Table 2 – Discovered Patterns per Scientific Areas

Discovered Patterns			
With Constraints	Sup	With Approx Relaxation	Sup
<(IP),(AED),(PLF)>	61%	<(AM1),(~AM2),(~AM3),(AM2)>	6%
<(IP),(~AED),(PLF),(AED)>	6%	<(AM1),(~AM2),(AM2)>	6%
<(SD),(AC),(SO)>	57%	**<(AM1),(AM2),(~AM3,AM3)>**	6%
<(SD),(~AC),(SO),(AC)>	6%	<(FEX),(F1),(~F2)>	22%
<(FEX),(F1),(F2)>	35%	<(FEX),(~F1),(~F2),(F1)>	8%
<(FEX),(~F1),(F2),(F1)>	5%	<(FEX),(~F1),(F2),(F1)>	5%
<(AM1),(AM2),(AM3)>	31%	<(~F2),(F1),(F2)>	6%
		<(IP),(AED),(~PLF)>	12%
		<(IP),(~AED),(PLF),(POO)>	6%
		<(~IP),(~AED),(IP),(AED)>	5%
		<(SD),(AC),(~SO)>	13%
		<(~SD),(~AC),(SD),(AC)>	6%

It is important to note that there is no other trivial way to perform an identical analysis. Usually, the results are stored in separate records, making more difficult the sequential analysis with simple queries. Remember that those queries require several natural joins, one for each constraint on the required sequence of results. In fact, the queries must define the entire automata with those operations, which is not a simple task. No other kind of queries will be able to address this problem, because without the sequence information, we are just able to discover how many students have approved or failed in each subject.

In this manner, constrained sequential pattern mining is a natural way to perform this kind of analysis, only requiring that "experts" design the possible curricula, which is trivial, since they are publicly known.

When applying an *Approx* relaxation, we are able to discover part of the patterns followed by students that are not able to conclude all subjects in four semesters, as specified in the previous automaton. For example, one of the causes of failure on the sequence of ASO subjects is failing on the subject of Operative Systems (SO). Since this subject is the third one in the sequence and it is only offered in the *Fall* semester, students in that situation are not able to conclude the three subjects in four semesters. Similarly, students that fail on the subjects of Physics 2 (F2), Mathematical Analysis 3 (AM3) and Functional and Logic Programming (PLF) are not able to conclude the corresponding sequences on 4 semesters, as shown in Table 2-right.

Another interesting pattern found is that 6% of students fail in the first opportunity to conclude Mathematical Analysis 3 (AM3), but seize the second opportunity, concluding that subject in the first enrollment (shadowed line in Table 2-right).

Additionally, the use of the approx relaxation also contributes to analyze the impact of some subjects on others. For example, an *approx* relaxation with one error discovers that 49% of the students that conclude MTP subjects in 3 semesters fail on AM3 and 40% on F2. Similarly, 45% of students that conclude ASO subjects in 3 semesters fail on AM3 and 39% on F2 (shadowed patterns in Table 3).

Table 3 – Patterns in scientific areas with one error

Patterns	Sup	Patterns	Sup
<(IP,SD),(AED),(PLF)>	57%	<(IP,~SD),(AED),(PLF)>	8%
<(IP),(AED),(PLF,SO)>	53%	<(IP),(AED),(PLF,~SO)>	10%
<(IP),(AED,AC),(PLF)>	53%	<(IP),(AED,~AC),(PLF)>	8%
<(SD),(AC),(PLF,SO)>	51%	<(SD),(AC),(~PLF,SO)>	8%
<(IP,AM1),(AED),(PLF)>	50%	<(IP,~AM1),(AED),(PLF)>	15%
<(SD,AM1),(AC),(SO)>	47%	<(SD,~AM1),(AC),(SO)>	15%
<(IP),(AED,AM2),(PLF)>	45%	<(IP),(AED,~AM2),(PLF)>	16%
<(IP),(AED,F1),(PLF)>	45%	<(IP),(AED,~F1),(PLF)>	16%
<(SD),(AC,F1),(SO)>	41%	<(SD),(AC,~F1),(SO)>	16%
<(SD),(AC,AM2),(SO)>	41%	<(SD),(AC,~AM2),(SO)>	16%
<(IP),(AED),(PLF,F2)>	**36%**	**<(IP),(AED),(PLF,~F2)>**	**24%**
<(SD),(AC),(SO,F2)>	**34%**	**<(SD),(AC),(SO,~F2)>**	**23%**
<(FEX,AM1),(F1),(F2)>	31%	<(~AM1,FEX),(F1),(F2)>	6%
<(FEX),(F1),(SO,F2)>	31%	<(FEX),(F1),(~SO,F2)>	5%
<(IP),(AED),(PLF,AM3)>	**29%**	**<(IP),(AED),(PLF,~AM3)>**	**30%**
<(FEX),(F1,AM2),(F2)>	28%	<(FEX),(F1,~AM2),(F2)>	7%
<(SD),(AC),(SO,AM3)>	**27%**	**<(SD),(AC),(SO,~AM3)>**	**26%**
<(AM1),(AM2),(F2,AM3)>	23%	<(AM1),(AM2),(~F2,AM3)>	8%
<(FEX),(F1),(F2,AM3)>	21%	<(FEX),(F1),(F2,~AM3)>	14%

Finding Common Curricula Instantiations with Optional Subjects. The challenge on finding which students choose what optional subjects is a non-trivial task, especially 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 other usual approach would be to query the database to count the support of each subject, knowing that students have followed some given curriculum. However, this approach 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.

In order to discover the optional subjects frequently chosen by students, we have used the methodology previously proposed – the use of constraint relaxations, defining a constraint based on the DFA shown in Fig. 7.

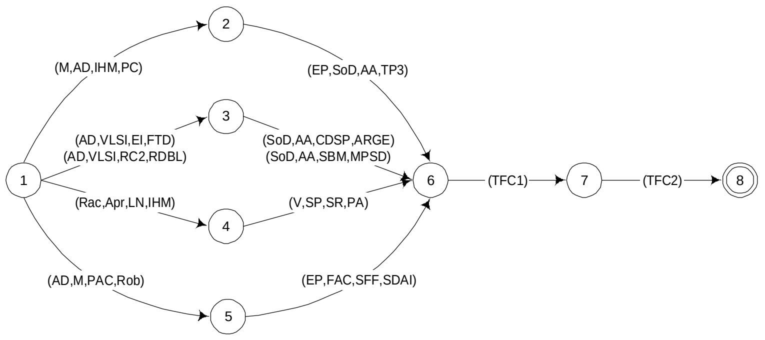


Fig. 7 DFA for finding optional subjects

This automaton accepts sequences that represent the curricula on the fourth curricular year for each specialty area (the first for PSI students, the second one for SCO, the third for IAR and the fourth for IIN). In practice, a constraint defined over this DFA filters all patterns that do not respect the model curriculum for the last two curricular years.

The use of constrained sequential pattern mining (with the specified constraint) would not contribute significantly to answer the initial question, since it would only achieve results similar to the ones obtained by the query above.

However, the use of the *Approx* relaxation described enables the discovery of several patterns. If the relaxation accepts at most two errors (ε=2) chosen from a restricted alphabet, composed by every non-common subject, we are able to find the frequent curricula instantiations with optional subjects. In general, students mostly attend Computer Graphics (PAC–Computed Assisted Project; IHM–Human Machine Interfaces) and Management subjects (Economy–E; Economical Theory 1–TE1; Financial Management–GF; Management–G; Management Introduction–IG), as shown in Table 4.

It is interesting to note that whenever IIN students have failed on some subject on the 4th year, they choose a specific optional subject in Economy (`TE1` or `IG`). The same happens for PSI and IAR students (behavior identified by shadowed rules). Note that in order to discover these rules, we have to be able to admit some errors on the sequence of subjects per specialty area, which is not easily done by specifying a query to a database.

Table 4 – Patterns with optional subjects attended by LEIC students

SA	Curricula Instantiations	LEIC sup	SA sup
PSI	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**PAC**)(TFC2,**GF**)>:22	1.5%	5%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**Econ**)(TFC2,**GF**)>:33	2.5%	8%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**Econ**)(TFC2,**IG**)>:28	2%	7%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1)(TFC2,**GF**,**IG**)>:33	2.5%	8%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**G**,**PAC**)(TFC2)>:22	1.5%	5%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**G**)(TFC2,GF)>:32	2%	8%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**G**)(TFC2,GCP)>:19	1%	5%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**G**)(TFC2,GEC)>:23	1.5%	5%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**G**)(TFC2,ARGE)>:18	1%	4%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**G**,**TE1**)(TFC2)>:29	2%	7%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**TE1**,**Econ**)(TFC2)>:21	1.5%	5%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**TE1**)(TFC2,**GF**)>:44	3%	10%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(TFC1,**TE1**)(TFC2,**IG**)>:27	1,5%	6%
	<(M,AD,IHM,PC)(AA,SoD,EP,TP3)(**TE1**)(**GEC**)>:15	1%	4%
	<(M,AD,IHM,PC)(AA,SoD,EP)(TFC1)(TFC2,**TE2**)>:15	1%	4%
IIN	<(M,AD,PAC,Rob)(EP,SDAI,SFF)(TFC1)(TFC2,**IG**)>:15	1%	8%
	<(M,AD,PAC,Rob)(EP,FAC,SDAI,SFF)(TFC1,**IHM**)(TFC2,**GF**)>:18	1%	10%
	<(M,AD,PAC,Rob)(EP,FAC,SDAI,SFF)(TFC1,**Econ**)(TFC2,**GF**)>:18	1%	10%
	<(M,PAC,Rob)(EP,FAC,SDAI,SFF)(TFC1,**G**)(TFC2)>:17	1%	10%
	<(M,PAC,Rob)(EP,FAC,SDAI,SFF)(TFC1,**TE1**)(TFC2)>:18	1%	10%
IAR	<(IHM,Rac,LN)(SP,V,PA,SR)(TFC1,**TE1**)(TFC2)>:15	1%	10%
	<(IHM,A,Rac,LN)(SP,V,PA,SR)(TFC1)(TFC2,**GF**)>:17	1%	10%
SCO	<(C)(VLSI,RC2,RDBL,AD)(AA,CDPSD,ARGE,SoD)(TFC1,**G**)(TFC2)>:23	1.5%	8%
	<(Elect)(VLSI,RC2,RDBL,AD)(AA,CDPSD,ARGE,SoD)(TFC1,**G**)(TFC2)>:27	1.5%	10%
	<(RC1)(VLSI,RC2,RDBL,AD)(AA,CDPSD,ARGE,SoD)(TFC1,**G**)(TFC2)>:19	1%	7%

Another interesting issue is the inexistence of frequent optional subjects among IAR and SCO students. Indeed, for the last ones there is only one frequent optional subject (Management – G).

Finding Artificial Intelligence curricula. As can be seen in previous analysis, the subjects exclusive to AI students have very low supports (about 13%). Naturally, the sequences of consecutive subjects have supports even lower. Indeed, the discovery of Artificial Intelligence (IAR) frequent curricula, like for IIN, is non-trivial, since the number of students in these specialty areas is reduced.

Given that the application of unconstrained sequential pattern mining algorithms found 5866 patterns in this dataset (for 20% of support, since we were not able to try lower supports due the memory requirements), and we want to find the sequence of subjects followed by IAR students, the use of constrained or unconstrained sequential pattern mining does not help to answer this second question.

However, if we use the proposed methodology, we have two alternatives: using a DFA specifying the IAR model curriculum and an *Approx-Accepted* relaxation as above, or using a DFA specifying PSI and SCO curricula models and a *Non-Accepted* relaxation with a restricted alphabet.

The patterns discovered by the second alternative (using a minimum support threshold equal to 2.5%) answer the question. (Table 5 shows the discovered patterns, excluding 8 patterns that are shared by PSI students).

Note that we were not able to find the entire model curriculum for Artificial Intelligence, because of the reduced number of students that have concluded each subject on the first enrollment. This fact, explains the number of discovered patterns (24). However, it is smaller than the number of patterns discovered with an unconstrained approach, confirming our claim about constraint relaxations.

The Non-Accepted relaxation was defined using a constraint similar to the previous one with the DFA represented in Fig. 5 without the model curriculum for IAR and IIN. Additionally, the relaxation alphabet was composed of all the common subjects and the advanced subjects specific to the Artificial Intelligence specialty area (38 subjects).

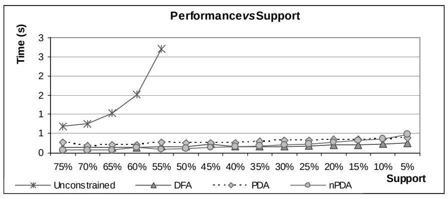
Table 5 – Artificial Intelligence frequent curricula

Patterns on Artificial Intelligence	
(AD,FL,P)(RC,TP)(IHM,A,R,LN)(SP,PA)	(FA,TAI)(AD,P)(RC,LP)(IHM,A,R,LN)(SP,V,PA,SR)
(FA,TAI)(FL,P)(RC,LP,TP)(IHM,A,R,LN)(SP,V,PA,SR)	(FA)(AD,FL,P)(RC,TP)(IHM,A,R,LN)(SP)
(FA,TAI)(AD,FL,P)(RC,LP,TP)(IHM,R,LN)(SP,V,SR)	(FA)(AD,FL,P)(RC)(IHM,A,R,LN)(SP,PA)
(FA,TAI)(AD,FL,P)(RC,LP,TP)(IHM,A,R,LN)(V,PA,SR)	(TAI)(AD,P)(RC,LP,TP)(IHM,A,R,LN)(SP,V,PA,SR)
(FA,TAI)(AD,FL,P)(RC,LP)(IHM,R,LN)(SP,V,PA,SR)	(TAI)(AD,FL,P)(RC,LP,TP)(IHM,R,LN)(SP,V,PA)
(FA,TAI)(AD,FL,P)(RC,LP)(IHM,A,R,LN)(SP,V,SR)	(TAI)(AD,FL,P)(RC,LP,TP)(IHM,A,R,LN)(SP,V,SR)
(FA,TAI)(AD,FL)(RC,LP,TP)(IHM,A,R,LN)(SP,V,PA,SR	(TAI)(AD,FL,P)(RC,LP)(IHM,A,R,LN)(SP,V,PA,SR)
(FA,TAI)(AD,P)(RC,LP,TP)(IHM,R,LN)(SP,V,PA,SR)	(FA,TAI)(AD,P)(RC,LP,TP)(IHM,A,R,LN)(SP,V,SR)

4.2 Efficiency Evaluation

Next, we compare the efficiency of constrained and unconstrained sequential pattern mining, and we assess the efficiency of the usage of the different constraint relaxations.

Comparison between Constrained and Unconstrained Mining. To compare the efficiency of constrained and unconstrained sequential pattern mining, we compare the time spent by constrained and unconstrained mining, using deterministic finite automata and deterministic and non-deterministic pushdown automata that discover a similar number of patterns (they differ on one or two patterns). (The DFA used in this comparison accepts the curricula on MTP and ASO scientific areas, of students that have failed at most once per subject. The ePDA used is the one shown in Fig. 6. The nPDA used accepts sequences in each scientific area that mimic the sequence of subjects attended by students, including the possibility of failure in the first two subjects of each scientific area).



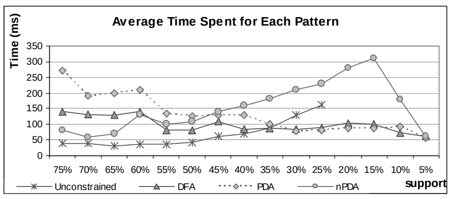


Fig. 8 Performance (on the left) and average time spent for each pattern (on the right), using different types of content constraints

It is interesting to note that the use of context-free languages implies a considerable increase on the time spent for each pattern. In fact, the time spent for each pattern may be five times slower than for unconstrained mining, and two times slower than for constrained mining with regular languages (see Fig. 8-right). The time spent for each pattern decreases with the number of discovered patterns, which is due to the decrease of the percentage of discarded sequences, the sequences that are not frequent and accepted by the constraint.

Evaluation of Constraint Relaxations. In general, mining with conservative relaxations is as efficient as mining with the entire constraint. However, the average time spent per discovered pattern is lower (Fig. 9). The results were obtained with the constraint based on the ePDA in Fig. 6.

Naturally, Non-Accepted and Approx relaxations spent much more time than the other relaxations, but this difference is mostly due to the number of patterns they discover.

It is important to note that mining with *Non-accepted* and *Approx* relaxations can be less efficient than unconstrained mining. This happens when the number of patterns discovered by these relaxations is similar to the number of discovered unconstrained patterns, as is usual for Non-accepted relaxations with a very restrictive constraint (as the ones used in this chapter).

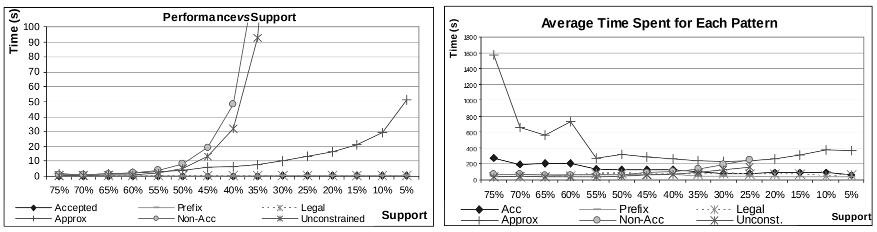


Fig. 9 Performance (on the left) and average time spent for each pattern (on the right), using different constraint relaxations

As Fig. 9– right shows *Approx* relaxations are the most expensive per discovered pattern. In fact, even when the number of discovered patterns is considerably lower than the number of unconstrained patterns, it is possible that *Approx* relaxations spent more time than unconstrained mining.

Approx Variants. When applied with a restricted alphabet *Approx* relaxations may present better performances.

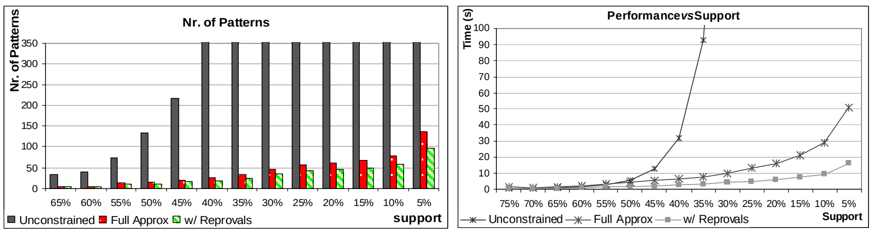


Fig. 10 Number of discovered patterns (on the left) and performance (on the right), using approx variants

Even when non-deterministic pushdown automata are used, *approx* relaxation outperforms unconstrained mining in the generality of situations, as shown in Fig. 10. These results were achieved using the constraint defined over the ePDA in Fig. 6, and restricting the items to failures**.**

Non-Accepted Variants. The results achieved with non-accepted relaxations are similar. When combined with an item constraint, it can reduce the number of patterns that it discovers. For example, due to the memory requirements of the large number of discovered patterns, it was not possible to discover unconstrained and non-accepted patterns for supports below 25%.

However, with items restricted to the subjects exclusive to Artificial Intelligence (as in section Finding Artificial Intelligence Curricula) it is possible to discover part of the non-accepted patterns, in an acceptable time. Fig. 11 shows the number of discovered patterns (on the left) and the corresponding performance (on the right), with the same constraint with an alphabet composed of the subjects exclusive to IAR specialty area.

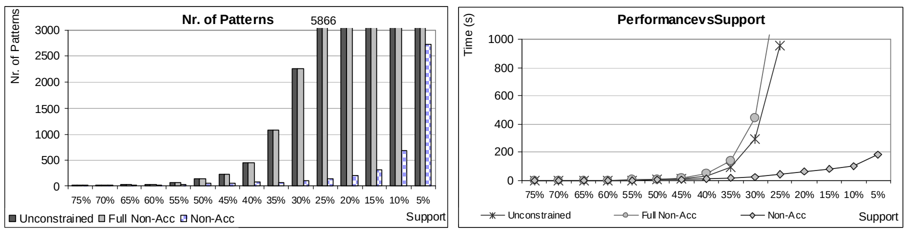


Fig. 11 Number of discovered patterns (on the left) and performance (on the right), using Non-accepted variants

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. This is achieved since relaxations extend the mining process to the discovery of patterns that are not directly inferred from the background knowledge represented in the system.

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, it is shown that the processing times spent in the mining process can be reduced if constraint relaxations are used.

The experiments reported in the case study show that the use of relaxations is of great help when there is not a precise knowledge about the behaviors in analysis, but can be of particular interest when there is an accurate knowledge about the models behind the data. In fact, the success of the use of *Non-Accepted* relaxation is conditioned by the existence of this knowledge, since it represents the only way to find some specific but interesting patterns.

At last, experiments have also shown that the ability to deal with errors is a real advantage, which makes possible the discovery of unknown patterns that are similar to accepted ones (with respect to some constraint), giving the user the ability to choose the level of similarity, by defining the number of errors accepted.

One important challenge created by this work, is to apply this methodology to the extraction of intra-transactional patterns, where there are no constraints to specify the structure of the transactions. Indeed, the definition of the corresponding relaxations would contribute to guide the traditional pattern mining processes, which may contribute to reduce the number of discovered patterns, and, consequently, to focus the process on user expectations.

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