A Constraint Programming Approach for Mining Sequential Patterns in a Sequence Database

Jean-Philippe Métivier[1], Samir Loudni[1], and Thierry Charnois[2]

[1] GREYC (CNRS UMR 6072) – University of Caen
Campus II Côte de Nacre, 14000 Caen - France
[2] LIPN CNRS (UMR 7030) – University PARIS 13
99, avenue Jean-Baptiste Clment 93430 Villetaneuse - France

Abstract. Constraint-based pattern discovery is at the core of numerous data mining tasks. Patterns are extracted with respect to a given set of constraints (frequency, closedness, size, etc). In the context of sequential pattern mining, a large number of devoted techniques have been developed for solving particular classes of constraints. The aim of this paper is to investigate the use of Constraint Programming (CP) to model and mine sequential patterns in a sequence database. Our CP approach offers a natural way to simultaneously combine in a same framework a large set of constraints coming from various origins. Experiments show the feasibility and the interest of our approach.

1 Introduction

Sequential pattern mining is a well-known data mining technique introduced in [1] to find regularities in a database of sequences. This problem is central in many application domains, such as web usage mining [7], bioinformatics and text mining [3]. For effectiveness and efficiency considerations, many authors [10, 24] have promoted the use of constraint to focus on the most promising knowledge by reducing the number of extracted patterns to those of a potential interest given by the final user. The most popular example is the minimal frequency constraint: it addresses all sequences having a number of occurrences in the database exceeding a given minimal threshold.

There are already in the literature many algorithms to extract *sequential patterns* (e.g. `GSP` [19], `SPADE` [25], `PrefixSpan` [15]), *closed* sequential patterns (e.g. `CloSpan` [23], `BIDE` [22]) or sequential patterns satisfying *regular expression* (e.g. `SPIRIT` [11]). All the above methods, though efficient, suffer from two major problems. First, they tackle particular classes of constraints (i.e. *monotonic* and *anti-monotonic* ones) by using devoted techniques. However, several practical constraints required in data mining tasks, such as regular expression, aggregates, do not fit into these classes. Second, they lack of generic methods to push various constraints into the sequential pattern mining process. Indeed, adding and handling simultaneously several types of constraints in a nice and elegant way beyond the few classes of constraints studied is not still trivial. The lack of generic approaches restrains the discovery of useful patterns because

the user has to develop a new method each time he wants to extract patterns satisfying a new type of constraints. In this paper, we address this open issue by proposing a generic approach for modelling and mining sequential patterns under various constraints using Constraint Programming (CP).

Our proposition benefits from the recent progress on cross-fertilization between data mining and CP for itemset mining [12, 14, 17]. The common point of all these methods is to model in a declarative way pattern mining as Constraint Satisfaction Problems (CSP), whose resolution provides the complete set of solutions satisfying all the constraints. The great advantage of this modelling is its flexibility, it enables to define and to push new constraints without having to develop new algorithms from scratch.

The key contribution of this paper is to propose a CP modelling for the problem of mining sequential patterns in a sequence database [1]. Our approach addresses in a unified framework a large set of constraints. This includes constraints such as frequency, closedness and size, and other constraints such as regular expressions, gap and constraints on items. Moreover, our approach enables to combine simultaneously different types of constraints. This leads to the first CP-based model for discovering sequential patterns in a sequence database under various constraints. Experiments on a case study on biomedical literature for discovering gene-RD relations from PubMed articles show the feasibility and the interest of our approach.

This paper is organized as follows. Section 2 gives the necessary definitions and presents the problem formulation. Section 3 introduces the main principles of constraint programming. Section 4 describes our CP model for mining sequential patterns in a sequence database. We review some related work in Section 5 and Section 6 reports in depth a case study from the biomedical literature domain for discovering gene-RD relations from PubMed articles. Finally, we conclude and draw some perspectives.

2 Sequential Pattern Mining

In this section, we introduce sequential patterns defined by Agrawal et al. [1].

2.1 Sequential Patterns

Sequential pattern mining [1] is a data mining technique that aims at discovering correlations between events through their order of appearance. Sequential pattern mining is an important field of data mining with broad applications (e.g., biology, marketing, security) and there are many algorithms to extract frequent sequences [19, 25, 23].

In the context of sequential patterns extraction, a *sequence* is an ordered list of literals called *items*. A sequence s is denoted by $\langle i_1, i_2 \dots i_n \rangle$ where i_k, $1 \leq k \leq n$, is an item. Let $s_1 = \langle i_1, i_2 \dots i_n \rangle$ and $s_2 = \langle i'_1, i'_2 \dots i'_m \rangle$ be two sequences. s_1 is *included* in s_2 if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $i_1 = i'_{j_1}$, $i_2 = i'_{j_2}$, ..., $i_n = i'_{j_n}$. s_1 is called a *subsequence* of s_2. s_2

Table 1. SDB_1: a sequence database

Sequence identifier	Sequence
1	$\langle a\ b\ c\ d\ a\rangle$
2	$\langle d\ a\ e\rangle$
3	$\langle a\ b\ d\ c\ \rangle$
4	$\langle c\ a\rangle$

is called a *super-sequence* of s_1, denoted by $s_1 \preceq s_2$. For example the sequence $\langle a\ b\ d\ c\rangle$ is a super-sequence of $\langle b\ c\rangle$: $\langle b\ c\rangle \preceq \langle a\ b\ d\ c\rangle$. A *sequence database* SDB is a set of tuples (sid, s), where sid is a sequence identifier and s a sequence. For instance, Table 1 represents a sequence database of four sequences. A tuple (sid, s) *contains* a sequence s_1, if $s_1 \preceq s$. The *support* of a sequence s_1 in a sequence database SDB, denoted $sup(s_1)$, is the number of tuples containing s_1 in the database[1]. For example, in Table 1, $sup(\langle c\ a\rangle) = 2$.

A *frequent sequential pattern* is a sequence such that its support is greater or equal to a given threshold: *minsup*. The *sequential pattern mining* problem is to find the *complete* set of frequent sequential patterns with respect to a given sequence database SDB and a support threshold *minsup*.

2.2 Sequential Pattern Mining under Constraints

In order to drive the mining process towards the user objectives and to eliminate irrelevant patterns, one can define constraints [10]. The most commonly used constraint is the frequency constraint (that assigns a value to *minsup*). We review some of the most important constraints for the sequential mining problem [10].

Closedness Constraint The *closed sequential patterns* [23] are a condensed representation of the whole set of sequential patterns. This condensed representation eliminates redundancies according to the frequency constraint. A frequent sequential pattern s is closed if there exists no other frequent sequential pattern s' such that $s \preceq s'$ and $sup(s) = sup(s')$. For instance, with $minsup = 2$, the sequential pattern $\langle b\ c\rangle$ from Table 1 is not closed whereas the pattern $\langle a\ b\ c\rangle$ is closed.

Item constraint. An item constraint specifies subset of items that should or should not be present in the sequential patterns. For instance, if we impose the constraint $C_{item} \equiv sup(p) \geq 2 \wedge (a \in p) \wedge (b \in p)$, three sequential patterns are mined from Table 1: $p_1 = \langle a\ b\rangle$, $p_2 = \langle a\ b\ c\rangle$ and $p_3 = \langle a\ b\ d\rangle$.

Size constraint. The aim of this constraint is to limit the length of the patterns, the length being the number of occurrences of items. The length of a pattern

[1] The relative support is also used:
$sup_{SDB}(T) = \frac{|\{(sid, s)\ s.t.\ (sid, s) \in SDB \wedge (T \preceq s)\}|}{|SDB|}$

p will be denoted by $len(p)$. For example, if $len(p) \geq 3 \wedge sup(p) \geq 2$, only two sequential patterns are extracted (p_2 and p_3).

Gap constraint. Another widespread constraint is the gap constraint. A sequential pattern with a gap constraint $C_{gap} \equiv [M, N]$, denoted by $p_{[M,N]}$, is a pattern such as at least M items and at most N items are allowed between every two neighbor items, in the original sequences. For instance, let $p_{[0,2]} = \langle c\ a \rangle$ and $p_{[1,2]} = \langle c\ a \rangle$ be two patterns with two different gap constraints and let us consider the sequences of Table 1. Sequences 1 and 4 support pattern $p_{[0,2]}$ (sequence 1 contains one item between (c) and (a) whereas sequence 4 contains no item between (c) and (a)). But only Sequence 1 supports $p_{[1,2]}$ (only sequences with one or two items between (c) and (a) support this pattern).

Regular expression constraint. A regular expression constraint C_{RE} is a constraint specified as a regular expression over the set of items. A sequential pattern satisfies C_{RE} if and only if the pattern is accepted by its equivalent deterministic finite automata [11]. For instance, the two sequential patterns $\langle a\ b\ c \rangle$ and $\langle a\ d\ c \rangle$ from Table 1 satisfy the regular expression constraint $C_{RE} = a * \{bb|bc|dc\}$.

3 Constraint Programming

In this section, we first introduce basic constraint programming concepts and then present two constraints of interest: `Among` and `Regular`.

3.1 Preliminaries

Constraint programming (CP) is a generic framework for solving combinatorial problems modelled as Constraint Satisfaction Problems (CSP). The key power of CP lies in its declarative approach towards problem solving: in CP, the user specifies the set of constraints which has to be satisfied, and the CP solver generates the correct and complete set of solutions. In this way, the specification of the problem is separated from the search strategy.

A *Constraint Satisfaction Problem* (CSP) consists of a finite set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ with finite domains $\mathcal{D} = \{D_1, \ldots, D_n\}$ such that each D_i is the set of values that can be assigned to X_i, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{X}$. A constraint $C \in \mathcal{C}$ is a subset of the cartesian product of the domains of the variables that are in C. The goal is to find an assignment $(X_i = d_i)$ with $d_i \in D_i$ for $i = 1, \ldots, n$, such that all constraints are satisfied. This assignment is called a solution to the CSP. For a given assignment t, $t[X_i]$ denotes the value assigned to X_i in t.

In Constraint Programming (see [2]), the solution process consists of iteratively interleaving search phases and propagation phases. The search phase essentially consists of enumerating all possible variable-value combinations, until we find a solution or prove that none exists. It is generally performed on a tree-like structure. In order to avoid the systematic generation of all the combinations and reduce the search space, the propagation phase shrinks the search

space: each constraint propagation algorithm removes values that a priori cannot be part of a solution w.r.t. the partial assignment built so far. The removal of inconsistent domain values is called *filtering*.

An important modelling technique from CP are the *global constraints* that provide shorthands to often-used combinatorial substructures. Global constraints embed specialized filtering techniques that exploit the underlying structure of the constraint to establish stronger levels of consistency much more efficiently. Nowadays, global constraints are considered to be one of the most important components of CP solvers in practice.

3.2 The Among and Regular Global Constraints

Among Constraint. This constraint restricts the number of occurrences of some given values in a sequence of n variables [5]:

Definition 1 (`Among` constraint, [5]). *Let X=X_1,...,X_n be a sequence of n variables, S a set of values, D^X the cartesian product of the variable domains in X. Let l and u be two integers s.t. $0 \leq l \leq u \leq n$.*

$$\texttt{Among}(X, S, l, u) = \{t \in D^X \mid l \leq |\ \{i, t[X_i] \in S\}\ | \leq u\}$$

The `Among` constraint can be encoded by channelling into 0/1 variables using the sum constraint [6]: $\forall i \in \{1, \ldots, n\}\ B_i = 1 \leftrightarrow t[X_i] \in S \wedge l \leq \sum_{i=1}^{n} B_i \leq u$.

Regular Constraint. Given a deterministic finite automaton M describing a regular language, constraint $\texttt{Regular}(X, M)$ ensures that every sequence of values taken by the variables of X have to be a member of the regular language recognised by M:

Definition 2 (`Regular` constraint, [16]). *Let M be a Deterministic Finite Automaton (DFA), $\mathcal{L}(M)$ the language defined by M, X a sequence of n variables.* $\texttt{Regular}(X, M)$ *has a solution iff $\exists\, t \in D^X$ s.t. $t \in \mathcal{L}(M)$.*

In [16], `Regular` constraint over a sequence of n variables is modelled by a layered directed graph $\mathcal{G} = (V, U)$, and a solution to $\texttt{Regular}(X,M)$ corresponds to an s-t path in graph $\mathcal{G}$, where s is the "source" node and t the "sink" node.

4 Modeling Sequential Patterns using CP

This section presents our CP modelling for the sequential pattern mining problem. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items, EOS a symbol not belonging to I ($EOS \notin I$) denoting the end of a sequence, SDB a set of m sequences and ℓ the maximal length of the sequence in SDB.

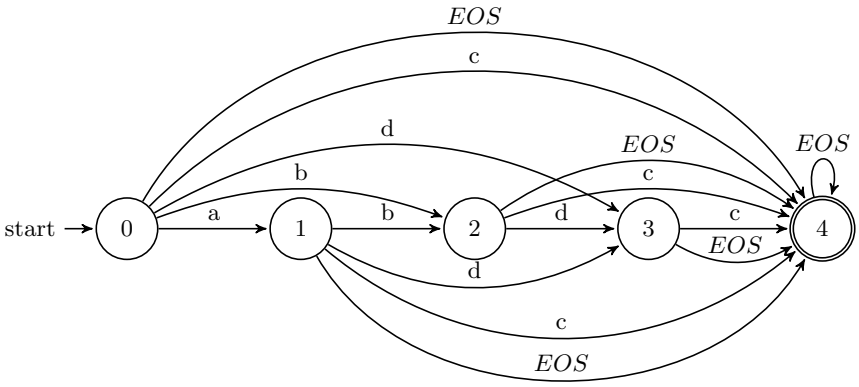

Fig. 1. The automaton modelling all subsequences of the sequence $\langle a\ b\ d\ c \rangle$.

4.1 Modelling an Unknown Sequential Pattern

Let p be the unknown sequential pattern we are looking for. First, ℓ variables $\{P_1, P_2, \ldots, P_\ell\}$ having $D_i = I \cup \{EOS\}$ for domain are introduced to represent p. Second, m boolean variables S_s (having $\{0, 1\}$ for domain) are used such that $(S_s = 1)$ iff sequence s contains the unknown sequential pattern p:

$$(S_s = 1) \Leftrightarrow (p \preceq s) \tag{1}$$

In equation (1), $(S_s = 1)$ if pattern p is a *subsequence* of s; 0 otherwise. So, $sup(p) = \Sigma_{s \in SDB}\, S_s$.

4.2 Modelling Sequential Pattern Mining

Let SDB be a sequence database and let a support threshold $minsup$. To encode that "$p \preceq s$", we first have to generate an automaton A_s capturing all subsequences that can be found inside a given sequence s. Then, we have to impose a `Regular` constraint stating that the unknown pattern p must be recognized by the automaton A_s.

To reduce the number of states of the automaton, for each sequence, we consider only its frequent items in the SDB w.r.t. $minsup$. Indeed, any super-pattern of an infrequent item cannot be frequent. Figure 1 shows an example of automaton generated for the third sequence of Table 1. Algorithm 1 depicts the pseudo-code for generating the automaton A_s.

To enumerate the complete set of frequent sequential patterns with respect to a given sequence database SDB and a support threshold $minsup$, we need to express that the unknown sequential pattern p occurs at least $minsup$ times. This problem is modelled by the following constraints:

Algorithm 1: Pseudo-code for generating A_s.

```
function generateAutomaton(Sequence s)
Automaton A_s;
A_s.nbState ← length(s) + 1;
A_s.addInitialState(0);
A_s.addAcceptingState(length(s));
foreach (state ∈ [0,length(s)]) do
    foreach (position ∈ [state+1,length(s)]) do
        item ← getItem(s,position);
        A_s.addTransition(state,item,position);
    A_s.addTransition(state,EOS,length(s));
return A_s;
```

Theorem 1 (Frequent Sequential Pattern Mining). *Sequential pattern mining is expressed by the following constraints:*

$$\forall s \in SDB : S_s = 1 \leftrightarrow \texttt{Regular}(p, A_s) \tag{2}$$

$$sup(p) = \sum_{s \in SDB} S_s \geq minsup \tag{3}$$

Proof. The reified constraint (2) models the support constraint. By construction, the automaton A_s encodes all sequential patterns that are subsequences of the sequence s. If the `Regular` constraint is satisfied (resp. not satisfied), then S_s must be equal to 1 (resp. must be equal to 0). The propagation is also performed, in a same way, from the equality constraint toward the `Regular` constraint.

The frequency constraint (3) enforces that at least *minsup* variables from S must take value 1. Together with constraint (2), it enforces that at least *minsup* sequences must support the sequential pattern described by p. □

4.3 Modelling other Constraints

This section shows how our CP approach enables us to express in a straightforward way constraints presented in Section 2.2.

Closedness Constraint. By definition a *closed* sequential pattern is the largest pattern that is contained in all selected sequences. Intuitively, in our encoding this corresponds to the sequential pattern having the less number of variables P_i instantiated to EOS. Thus, the satisfaction problem is turned into an optimization one: *minimize* the numbers of variables P_i instantiated to EOS. To express this minimization problem, we first have to add for each variable P_i an unary constraint c_i stating that if $(P_i = EOS)$ we have to pay a cost 1; 0 otherwise.

Then, we have to minimize the cost function $c(p) = \sum_{P_i \in p} c_i$ to obtain a closed sequential pattern:

$$\begin{array}{ll} \texttt{minimize}_p & c(p) \\ sup(p) & \geq minsup \end{array} \tag{4}$$

To enumerate the complete set of closed sequential patterns with respect to a given sequence database SDB and a support threshold $minsup$, we need to avoid future patterns to be equal to the previously found closed patterns. So, each time a frequent sequential pattern p is proven closed, we dynamically add a new constraint to forbid it.

Item Constraint. In order to specify that a subset of items should or should not be present in the sequential patterns, we have to add the following constraint:

$$\texttt{Among}(p, V, [l, u]) \tag{5}$$

where V is a subset of items, l and u are two integers s.t. $0 \leq l \leq u \leq \ell$. The `Among` constraint enforces that the number of variables of p that take a value from V is at least l and at most u. Since p represents the sequential patterns we are looking for, the previous constraint ensures that items of V should be present at least l times in the mined patterns.

To express the fact that the sequential patterns should not contain any item of V, we just have to set l and u to 0.

Size Constraint. In order to consider the frequent sequential patterns of size k, we just have to add the following constraints:

$$\forall i \in [1 \dots k] : P_i \neq EOS \tag{6}$$

$$\forall i \in [k+1 \dots \ell] : P_i = EOS \tag{7}$$

The previous constraints enforce that the k first variables of p must be different from EOS, while the $(\ell - k)$ remaining variables of p must be equal to EOS.

The minimum size constraint (i.e. $len(p) \geq k$) can be formulated by the following constraint:

$$\forall i \in [1 \dots k] : P_i \neq EOS \tag{8}$$

For the maximum size constraint (i.e. $len(p) \leq k$), this can be modelled as follows:

$$\forall i \in [k+1 \dots \ell] : P_i = EOS \tag{9}$$

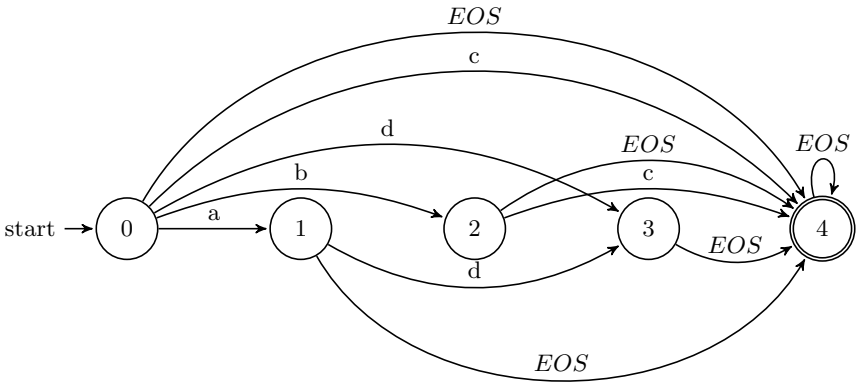

Fig. 2. The new automaton modelling all subsequences of the sequence $\langle a\ b\ d\ c\rangle$ satisfying the gap constraint [1,1].

Gap Constraint. Let $p_{[M,N]}$ be the sequential pattern satisfying the gap constraint $[M, N]$. To encode this constraint we have to modify the construction of the automaton A_s (cf. Theorem 1) in a such way that only transitions respecting the gap constraint will be kept. Let $A_s^{[M,N]}$ be the new resulting automaton representing all sequential patterns that are subsequence of the sequence s and satisfying the gap constraint. Finally, the reified constraint (2) is rewrited as follows:

$$\forall s \in SDB : S_s = 1 \leftrightarrow \texttt{Regular}(p, A_s^{[M,N]}) \tag{10}$$

Theorem 2 (Frequent Sequential Pattern Mining with Gap). *Sequential pattern mining with a gap constraint is expressed by constraints (3) and (10).*

Figure 2 gives the new automaton obtained from the automaton of Figure 1 with the gap constraint $[1, 1]$. For instance, the sequential pattern $\langle a\ b\rangle$ does not satisfy the gap constraint; it is not recognized by the new automaton.

To generate the automaton $A_s^{[M,N]}$ for a sequence s, we need to modify Algorithm 1 in a such way that only valid transitions satisfying the gap constraint $[M, N]$ are considered . This is done by adding the following condition inside the second loop foreach: $(state == 0 \ ||\ M \leq position - state \leq N)$ (see Algorithm 1).

Regular Expression Constraint. Let A_{re} be an automaton encoding a regular expression over the set of items. Then, the regular expression constraint can be formulated as follows:

$$\texttt{Regular}(p, A_{re}) \tag{11}$$

As presented in [8], a regular expression can be translated into a deterministic finite automaton. Thus, the `Regular` constraint over p ensures that every

sequence of values taken by the variables of p have to be a member of the regular language recognised by A_{re}, therefore recognized by the regular expression associated to A_{re}.

5 Related Work

Computing Sequential Patterns. In the context of constraint-based sequential pattern mining, several algorithms have been proposed [19, 25, 15, 23, 22, 11]. All these algorithms exploit properties of the constraints (i.e., monotonicity, anti-monotonicity or succinctness) to perform effective pruning. For constraints that do not fit in these categories, they are handled by relaxing them to achieve some nice property (like anti-monotonicity) facilitating the pruning. For instance, Garofalakis et al. [11] proposed regular expressions as constraints and developed a family of `SPIRIT` algorithms each achieving a different kind of relaxation of the regular expression constraint. Such a method, though interesting, makes tricky the integration of such constraints in a nice and elegant way. So, unlike these algorithms, our approach enables to address in a unified framework a broader set of constraints, and more importantly, to combine them simultaneously.

CP for Pattern Mining. In the context of local patterns, an approach using CP for itemset mining has been proposed in [17]. This approach addresses in a unified framework a large set of local patterns and constraints such as frequency, closedness, maximality, constraints that are monotonic or anti-monotonic. To deal with richer patterns satisfying properties involving several local patterns, different extensions have been proposed, such as pattern sets [13], n-ary patterns [14], top-k patterns [20] or skypatterns [21]. Our approach also benefits from the recent progress on cross-fertilization between data mining and CP for itemset mining, but it addresses a different problem with a different modelling.

CP for Sequence Mining. More recently, Coquery et al. [9] have proposed a SAT-Based approach for Discovering frequent, closed and maximal sequential patterns with wildcards in only a single sequence of items. However, unlike [9], our approach considers a database of sequences of items. Moreover, in [9], the sequential patterns with non-contiguous items are modelled using empty items as wildcards. But the gap between the items have to be fixed. Then, for instance the two sequential patterns $\langle a\ o\ b\rangle$ and $\langle a\ o\ o\ b\rangle$ are considered different. On the contrary, our modelling enables us to define any (minimal or maximal) value for the gap.

6 Experimentations

Experiments are conducted on texts from biological and medical texts. The goal is to discover relations between genes and rare diseases. The details of this application is given in [4]. In this section, we focus on the extraction of sequential patterns using our CP approach, and we give quantitative results showing the relevant of the approach.

6.1 Case Study

Settings. We created a corpus from the PubMed database using HUGO[2] dictionary and Orphanet dictionary to query the database to get sentences having at least one rare disease and one gene. 17,527 sentences have been extracted in this way and we labelled the gene and rare disease (RD) names thanks to the two dictionaries. For instance, the sentence "*<**disease**>**Muir-Torre syndrome**<**disease**> is usually inherited in an autosomal dominant fashion and associated with mutations in the mismatch repair genes, predominantly in <**gene**>**MLH1**<**gene**> and <**gene**>**MSH2**<**gene**> genes.*" contains one recognized RD, and two recognized genes. These 17,527 sentences are the training corpus from which we experiment the sequential pattern extraction.

Sequential Pattern Extraction. Sequences of the SDB are the sentences of the training corpus: an item corresponds to a word of the sentence. We carry out a POS tagging of the sentences thanks to the TreeTagger tool [18]. In the sentences, each word is replaced by its lemma, except for gene names (respectively disease names) which are replaced by the generic item $GENE$ (respectively $DISEASE$). Note that unlike machine learning based methods, our approch does not require to annotate the relations: they are discovered.

In order to discover sequential patterns, we use usual constraints such as *the minimal frequency* and *the minimal length* constraints and other useful constraints expressing some *linguistic knowledge* (e.g. *membership* and *association* constraints). The goal is to retain sequential patterns which convey linguistic regularities (e.g., gene-rare disease relationships). Our method offers a natural way to simultaneously combine in a same framework these constraints coming from various origins. We briefly sketch these constraints.

- *The minimal frequency constraint.* Three values of minimal frequency have been experimented: 2%, 5%, and 10%.
- *The minimal length constraint.* The aim of this constraint is to remove sequential patterns that are too small w.r.t. the number of items (number of words) to be relevant linguistic patterns. We tested this constraint with a value set to 3.
- *The membership constraint.* This constraint enables to filter out sequential patterns that do not contain some selected items. For example, we express that the extracted patterns must contain at least three items (expressing the linguistic relation): $GENE$, $DISEASE$ and noun or verb[3]. We used the item constraint to enforce this constraint.
- *The association constraint.* This constraint expresses that all sequential patterns that contain the verb item must contain its lemma and its grammatical category. We used the item constraint to enforce this constraint.
- *The closedness constraint.* In order to exclude redundancy between patterns, we used *closed* sequential patterns.

[2] www.genenames.org

[3] For each word (i.e. item), its grammatical category is stored in a base.

#sentences	50		100		150		200		250	
	#sol.	time	#sol.	time	#sol.	time	#sol.	time	#sol.	time
freq > 2%	129	1,105	329	12,761	441	35,164	(89)	–	(34)	–
freq > 5%	47	285	67	1,571	81	2,091	94	4,119	119	8,516
freq > 10%	4	53	21	251	26	577	29	1,423	28	2,764

#sentences	300		350		400		450		500	
	#sol.	time	#sol.	time	#sol.	time	#sol.	time	#sol.	time
freq > 2%	(129)	–	(45)	–	(10)	–	(1)	–	(0)	–
freq > 5%	101	9,620	93	16057	83	21,764	84	35,962	(26)	–
freq > 10%	30	5147	24	4,493	23	7,026	20	13,744	21	17,708

Table 2. Results obtained on different subsets of the PubMed dataset.

Experimental Protocol. We carried out experiments on several subsets of the PubMed dataset with different sizes ranging from 50 to 500 sentences. A timeout of 10 hours has been used. For each subset, we report the number of extracted closed sequential patterns and the CPU-times to extract them (in seconds). When the timeout is reached, the number of extracted patterns (until the timeout) is given in parenthesis.

All experiments were conducted on AMD Opteron 2.1 GHz and a RAM of 256 GB. We implemented our proposal in C++ using the library toulbar2[4] for solving constrained optimization problems modelled as *Cost Function Network* (CFN).

6.2 Results

Table 2 reports the results we obtained on different subsets of the PubMed dataset with values of *minsup* ranging from 2% to 10%. From these results, we can draw the following remarks.

i) Soundness and Flexibility. As the resolution performed by the CP solver is sound and complete, our approach is able to mine the correct and complete set of sequential patterns satisfying the different constraints. We compared the sequential patterns extracted by our approach with those found by [4], and the two approaches return the same set of patterns. Table 2 depicts the number of closed sequential patterns according to *minsup*. As expected, the lower *minsup* is, the larger the number of extracted sequential patterns.

ii) Highlighting useful sequential patterns. Our approach allowed to extract several relevant *linguistic patterns*. Such patterns can be used to explain RD-gene relationships from PubMed articles. For instance, three sequential patterns of great interest were highlighted by the expert:

1. $\langle (DISEASE)\ (be)\ (cause)\ (by)\ (mutation)\ (in)\ (the)\ (GENE)\rangle$
2. $\langle (GENE)\ (occur)\ (in)\ (DISEASE)\rangle$

[4] `https://mulcyber.toulouse.inra.fr/projects/toulbar2`.

3. $\langle (DISEASE)\ (be)\ (an)\ (mutation)\ (in)\ (GENE) \rangle$

From a biomedical point of vue, these sequential patterns are interesting since they convey a notion of *causality* (i.e. gene *cause* rare disease).

iii) Computational Efficiency. This experiment quantify runtimes and the scalability of our approach. In practice, runtimes vary according to the size of the datasets. For datasets with size up to 200, the set of all solutions is computed. We observe that runtimes vary from few seconds for high frequency thresholds to about few hours for low frequency thresholds. However, for large size datasets (≥ 200) and low frequency thresholds (i.e. $minsup = 2\%$), the CP approach does not succeed to complete the extraction of all closed sequential patterns within a timeout of 10 hours. Indeed, with the increase of the size of the dataset, the search space and the runtime increase drastically, and the solver spends much more time to find the first solution.

Finally, note that comparing runtimes with those obtained by ad hoc approaches would be rather difficult. In fact, these approaches use devoted techniques and do not offer the same level of genericity and expressivity as in our CP approach. Moreover, they cannot push in depth simultaneously different categories of constraints. Above all, there does not exist any algorithm, neither tool, for extracting sequential patterns under all the contraints proposed in our work.

7 Conclusion

We have proposed a flexible approach to mine sequential patterns of items in a sequence database. The declarative side of the CP framework easily enables us to address a large set of constraints and leads to a unified framework for extracting sequential patterns under constraints. Finally, the feasibility and the interest of our approach has been highlighted through experiments on a case study in biomedical literature for discovering gene-RD relations from PubMed articles.

We are currently investigating a new direction to enhance the efficiency of our approach: instead of constructing an automaton for every sequence, it would be more efficient to build some variant of a Prefix Tree Automata on the original dataset to avoid some redundancies. Furthermore, we intend to extend our approach to discover sequential patterns of itemsets in a sequence database. Discovering pattern sets is an attractive road to propose actionable patterns and the CP modelling is a proper paradigm to tackle this challenge [13, 14]. Further work is to address this issue.

Acknowledgements. This work is partly supported by the ANR (French Research National Agency) funded project FiCOLOFO ANR-10-BLA-0214. We would like to thank Bruno Crémilleux for his valuable comments.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995.

2. K. R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.
3. N. Béchet, P. Cellier, T. Charnois, and B. Crémilleux. Discovering linguistic patterns using sequence mining. In *CICLing (1)*, pages 154–165, 2012.
4. N. Béchet, P. Cellier, T. Charnois, and B. Crémilleux. Sequential pattern mining to discover relations between genes and rare diseases. In *IEEE Int. Symp. on Computer-Based Medical Systems (CBMS)*, pages 1–6, 2012.
5. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
6. C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Among, common and disjoint constraints. In *CSCLP*, pages 29–43, 2005.
7. I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *KDD*, pages 280–284. ACM, 2000.
8. Chia-Hsiang Chang and Robert Paige. From regular expressions to dfa's using compressed nfa's. *Theor. Comput. Sci.*, 178(1-2):1–36, 1997.
9. E. Coquery, S. Jabbour, L. Saïs, and Yakoub Salhi. A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 258–263. IOS Press, 2012.
10. G. Dong and J. Pei. *Sequence Data Mining*, volume 33 of *Advances in Database Systems*. Kluwer, 2007.
11. Minos N. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Trans. Knowl. Data Eng.*, 14(3):530–552, 2002.
12. T. Guns, S. Nijssen, and L. De Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983, 2011.
13. T. Guns, S. Nijssen, and L. De Raedt. k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25(2):402–418, 2013.
14. M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2010.
15. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In D. Georgakopoulos and A. Buchmann, editors, *ICDE*, pages 215–224. IEEE Computer Society, 2001.
16. G. Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP'04*, volume 2239 of *LNCS*, pages 482–495. Springer, 2004.
17. L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In Ying Li, Bing Liu, and Sunita Sarawagi, editors, *KDD'08*, pages 204–212. ACM, 2008.
18. H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, September 1994.
19. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996.

20. W. Ugarte, P. Boizumault, S. Loudni, and B. Crémilleux. Soft threshold constraints for pattern mining. In Jean-Gabriel Ganascia, Philippe Lenca, and Jean-Marc Petit, editors, *Discovery Science*, volume 7569 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2012.
21. W. Ugarte, P. Boizumault, S. Loudni, B. Crémilleux, and A. Lepailleur. Découverte des soft-skypatterns avec une approche PPC. In Christel Vrain, André Péninou, and Florence Sèdes, editors, *EGC*, volume RNTI-E-24 of *Revue des Nouvelles Technologies de l'Information*, pages 217–228. Hermann-Éditions, 2013.
22. J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *ICDE*, pages 79–90. IEEE Computer Society, 2004.
23. X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In Daniel Barbará and Chandrika Kamath, editors, *SDM*. SIAM, 2003.
24. M. J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *CIKM*, pages 422–429. ACM, 2000.
25. M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.