Sequence Mining under Multiple Constraints

Nicolas Béchet
Université de Bretagne-Sud
IRISA, Campus de Tohannic
56017 Vannes, France
nicolas.bechet@irisa.fr

Peggy Cellier
INSA de Rennes, IRISA
Campus de Beaulieu
35042 Rennes, France
peggy.cellier@irisa.fr

Thierry Charnois
Université Paris 13
Sorbonne Paris cité, LIPN
93430 Villetaneuse, France
thierry.charnois@lipn.univ-paris13.fr

Bruno Crémilleux
Université de Caen Basse-Normandie, GREYC
14032 Caen Cedex 5, France
bruno.cremilleux@unicaen.fr

ABSTRACT

In this paper, we address the problem of mining sequential patterns under multiple constraints. Unlike classical algorithms, our approach handles various types of constraints which are not only numeric but also symbolic and syntactic. These multiple constraints enable us to express a large scope of knowledge to focus on interesting patterns. We illustrate our approach with the detection of gene–rare disease relationships from biomedical texts for the documentation of rare diseases.

Keywords

Sequential data mining, constraint-based data mining, pattern discovery, information extraction, natural language processing

1. INTRODUCTION

Rare diseases are a major public health issue. A rare disease (RD) is a disease affecting fewer than 1 in 2,000 persons. There are between 6,000 and 8,000 RDs affecting about 30 million people in Europe and much more in the rest of the world. This context represents a strong motivation to address the problem of extracting gene–RD relationships from text collection such as the PubMed repository dealing with more than 24 million biomedical publications.

The discovery of gene–RD relationships requires to take into account the order of appearance between the elements of a text. Sequential pattern mining is a well-known data mining technique [13] that aims at extracting correlations between events (called items) through their order of appearance (i.e. the so-called *sequential patterns*) in a database of sequences. It is one of the most studied and challenging tasks in data mining with a wide range of applications and domains. For effectiveness and efficiency considerations, many authors [2, 17] have promoted the use of constraints to focus on the most promising patterns according to the interests given by the final user. Many kinds of constraints are tackled such as regular expression constraints in SPIRIT [4], aggregate constraints [10], gap constraints [7]. A survey of various constraints can be found in [2]. Several classes of constraints have been studied [2] and there are efficient methods for specific kinds of constraints such as the well-known (anti-)monotonic constraints [10] or constraints based on frequency measures [11]. Another direction of improvements is to reduce the set of patterns by selecting a small subset of patterns with the same expressiveness power. Closed patterns are often used to this end because a closed pattern summarizes an equivalence class, i.e. a set of patterns which are mapped to the same set of objects (or transactions) of a database. Our method takes benefit of these two research directions by combining multiple constraints with closed patterns.

The recent trend of hybridization of data mining and Natural Language Processing (NLP) techniques has shown the interest of handling various constraints to put characteristics coming from the NLP area into the data mining process in order to discover meaningful sequential patterns from textual literature. As an example, dealing with the biomedical literature asks constraints addressing features linked to text characteristics such as scope, length, gap, membership (specify a subset of items), association (a relation between two classes of items must be satisfied[1]). Moreover, as the number of sequential patterns generally remains large, the user prefers the safe summary produced by the closed patterns. As far as we know, there is no method to extract sequential patterns when the above constraints are combined together. Our goal is to fill this gap.

The contribution of this paper is to design a method providing the correct and complete set of sequential patterns satisfying conjunctions of various syntactic and symbolic constraints as exemplified above and including the closedness constraint. Extracting patterns under such constraints provides a major result to discover linguistic patterns, as for example patterns highlighting gene–RD relationships. Con-

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695889

[1]An example of the association constraint in NLP is "for each itemset, a category of words like a verb must be associated to its canonical form (called lemma, for instance *have* is a lemma of *having*)".

Sequence identifier	Sequence
1	$\langle(a)\ (b)\ (c)\ (d)\rangle$
2	$\langle(a)\ (b)\ (c)\rangle$
3	$\langle(a\ b)\ (d)\ (b)\ (c)\ (d)\rangle$

Table 1: Example of a sequential database.

trary to other related work on the topic, this work addresses the problem in a broad scope (i.e. a large set of combinations of constraints are tackled). Experiments on the detection of gene–RD relationships in text show the impact of different kinds of constraints offered by our method on the performance of this task. In addition, the presented method is general and can be reused in other applicative contexts.

The rest of the paper is organized as follows. Section 2 presents background knowledge about sequential pattern mining. Section 2 describes our algorithm performing with combinations of several kinds of constraints in the sequential pattern mining process. Section 4 gives the results of the experimental evaluation.

2. BACKGROUND KNOWLEDGE

Sequential pattern mining is a data mining technique introduced in [13] to find regularities in a sequence database. Many algorithms have been developed to extract sequential patterns [9, 13, 16, 18].

In sequential pattern mining, an *itemset* I is a set of literals called *items*. For example, $(a\ b)$ is an itemset with two items: a and b. A *sequence* S is an ordered list of itemsets, denoted by $s = \langle I_1 \dots I_m\rangle$. For instance, $\langle(a\ b)\ (d)\ (b)\ (c)\ (d)\rangle$ is a sequence of five itemsets. A sequence $S_1 = \langle I_1 \dots I_n\rangle$ is *included* in a sequence $S_2 = \langle I'_1 \dots I'_m\rangle$ if there exist integers $1 \leq j_1 < ... < j_n \leq m$ such that $I_1 \subseteq I'_{j_1}$,..., $I_n \subseteq I'_{j_n}$. The sequence S_1 is called a *subsequence* of S_2, and we note $S_1 \preceq S_2$. For example, $\langle(a)\ (c)\rangle$ is included in $\langle(a\ b)\ (d)\ (b)\ (c)\ (d)\rangle$. 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 depicts a sequence database of three sequences. A tuple (sid, S) *contains* a sequence S_1, if $S_1 \preceq S$. The *support*[2] of a sequence S_1 in a sequence database SDB, denoted $sup(S_1)$, is the number of tuples in the database containing S_1. For example, in Table 1 $sup(\langle(b)\ (d)\rangle) = 2$, since Sequences 1 and 3 contain $\langle(b)\ (d)\rangle$. A *frequent sequential pattern* is a sequence such that its support is greater or equal to a given support threshold *minsup*.

The set of frequent sequential patterns can be very large. Pattern condensed representations, such as closed sequential patterns [16], have been proposed in order to eliminate redundancy without loss of information. A frequent sequential pattern S is closed if there is no other frequent sequential pattern S' such that $S \preceq S'$ and $sup(S) = sup(S')$. For instance, with $minsup = 1$, the sequential pattern $\langle(a\ b)\rangle$ from Table 1 is not closed because $sup(\langle(a\ b)\rangle) = sup(\langle(a\ b)(d)\rangle)$ and $\langle(a\ b)\rangle \preceq \langle(a\ b)(d)\rangle$.

The frequency constraint is likely the most widespread constraint in the constraint-based pattern paradigm [2]. Unfortunately, used alone, frequent patterns generally lack of interest and in practice many other constraints are useful. Many constraints will be presented in the following. As an example, we explain the gap constraint which is less intuitive than other constraints. A sequential pattern with a gap constraint $[M, N]$, denoted by $P_{[M,N]}$, is a pattern such as at least M itemsets and at most N itemsets are allowed between every two neighbor itemsets in the matched sequences. For instance, in Table 1, $P_{[0,1]} = \langle(a)(c)\rangle$ and $P_{[0,2]} = \langle(a)(c)\rangle$ are two patterns with gap constraints. $P_{[0,2]}$ matches three sequences (1, 2 and 3) whereas $P_{[0,1]}$ matches only two sequences (1 and 2). Indeed, in Sequence 3 there are two itemsets between the itemset that contains a and the itemset that contains c and the maximal gap value (there 1) is not satisfied.

To compute those different kinds of sequential patterns, there exist several approaches which can be divided into two groups: pattern growth approaches, such as CloSpan [16], Bide [14], GAPBIDE [7], CTSP [8], and depth-first search approaches based on a vertical database format, such as SPADE [18], CLaSP [5] or [3]. The contribution of this paper is based on a pattern growth approach, thus more details about the pattern growth strategy are given in the following section. The vertical database format strategy avoids several scans of the database and has good performance to compute frequent or closed sequential pattern. However, existing algorithms do not include any constraint except the closure.

3. EXTRACTION OF CLOSED SEQUENTIAL PATTERNS UNDER MULTIPLE CONSTRAINTS

This section describes our method, CloSPEC (Closed Sequential Pattern Extraction under Constraints), to extract closed sequential patterns satisfying conjunctions of various syntactic and symbolic constraints. We will see that combining constraints like the gap with closed patterns arises algorithmic challenges. In addition to the large scope of these constraints, our method runs on sequences made of itemsets and not simple items. In the NLP applicative context, it means that a word can be represented by multiple information (e.g., the word itself, its lemma, its categorical grammar). It enables to provide more useful patterns because several levels of abstraction can be mixed. As far as we know, there is no other method combining so many constraints on sequences made of itemsets.

3.1 The CloSPEC Algorithm

CloSPEC is given in Algorithm 1. It is based on the *pattern-growth* approach [6]. The first step extracts the frequent items satisfying the given C set of constraints in order to build the patterns of size 1. Then, for each frequent pattern of length 1, the *projected database* is built by removing infrequent items. The projected database of a pattern P is the set of sequences of SDB that contain P, removing P itself. It means that the projected database of a pattern P contains the suffixes of the sequences of SDB where P is the prefix. For the sake of reducing the memory space, in our algorithm the pseudo-projected databases are used. The pseudo-projected database of a pattern P is the projected database of P where sequences are stored thanks to pointers on the SDB instead of a physical copy [3]. The process is recursively executed in the **PGrowthConstraint** procedure (cf. Algorithm 2).

[2]The *relative support* is also used: $sup_{rel}(S_1) = \frac{sup(S_1)}{|SDB|}$

[3]For more information, see [6].

Algorithm 1 CloSPEC(*SDB*,*C*)

Inputs: *SDB*: a sequential database,
C: a set of constraints
Output: The set of closed sequential patterns under constraints
1. Build all frequent itemsets of size 1 *1-length-Patterns* verifying *C*
2. For each pattern *P* of *1-length-Patterns* **do**
(2.1) Build the projected database of *P*, *ProjectedDB[P]*, where infrequent items have been removed
(2.2) PGrowthConstraint(P, *ProjectedSDB[P]*, *C*)

In order to explain the **PGrowthConstraint** procedure, we present some notations defined in [15]. An *I-extension* extends a sequential pattern P by adding an item to an itemset of P. A *S-extension* extends P by adding a new itemset made of 1 item. Let M be the pattern $M = \langle(a)\ (b)\ (c)\rangle$. Examples of extensions of M when the item d is added are:

right I-extension	*left I-extension*
$\langle(a)\ (b)\ (c, \boldsymbol{d})\rangle$	$\langle(a, \boldsymbol{d})\ (b)\ (c)\rangle$
right S-extension	*left S-extension*
$\langle(a)\ (b)\ (c)\ (\boldsymbol{d})\rangle$	$\langle(\boldsymbol{d})\ (a)\ (b)\ (c)\rangle$

An *occurrence* of a sequential pattern P is defined as a sub-sequence of the SDB matching with P. Algorithm 2 describes the **PGrowthConstraint** procedure. Its principle is to verify if a candidate pattern P satisfies all the constraints (including the closure) (steps 1-4) and build a right (I or S)-extension of P with a new item (step 5). First, **PGrowthConstraint** checks if (1) there is a sequential pattern P' with the same number of occurrences as P[4], (2) $P \preceq P'$ and (3) P' is a left (I or S)-extension of P. If such a pattern P' exists, P is not closed and the closed pattern associated to P will be extracted from P', the process can thus stop (step 1) with a safe pruning approach (the break statement). Otherwise, the procedure tests if P is closed according to the CloSpan definition [16]. If not, and if P satisfies the given constraints, P is added to $ClosedHash$ (step 4). On the contrary, if it exists a pattern P'' with the same support as P, $P \preceq P''$, and P'' is an (I or S)-extension of P, then P is not closed and thus it is not added to $ClosedHash$. Finally, the **PGrowthConstraint** procedure is recursively called with the right extensions of the sequential pattern P (step 5).

3.2 Constraint Checking

This section explains how the constraint checking is performed. We start by specifying the (anti-)monotonic constraints.

Definition 1 ((Anti-)monotonic constraints). *A constraint C is **monotonic** if for any pattern P satisfying C and any pattern P' such that $P \preceq P'$ then P' also satisfies C. Respectively, a constraint C' is **anti-monotonic** if for any pattern P satisfying C and any pattern P' such that $P' \preceq P$ then P' also satisfies C.*

According to this definition, it is trivial to prove that the minimal length constraint is a monotonic constraint and the minimal support is an anti-monotonic constraint.

[4]The $\cup_{IorS}$ operator means that item x can be an I- or S-extension of pattern P. nb_occ is the number of occurrences of P.

Algorithm 2 PGrowthConstraint(P, $ProjectedDB$, C)

Inputs: P: a sequential pattern, $ProjectedDB$: a projected database, C: a set of constraints
1. If there exists an item x such as $nb\_occ(x \cup_{IorS} P) = nb\_occ(P)$ **then Break**
2. Compute the right extension of P (I-extension and S-extension), I_R
3. Compute the left extension of P (I-extension and S-extension), I_L
4. If $\nexists y \in I_R$ such as $sup(P \cup_{IorS} y) = sup(P)$ and $\nexists z \in I_L$ such as $sup(z \cup_{IorS} P) = sup(P)$ and P verifying constraints C **then**
(4.1) Add ($sup(P)$,P) to $ClosedHash$
5. For each item i of I_R **do**
(5.1) Compute $ProjectedDB_i$, the projected database of $P \cup_{IorS} y$
(5.2) PGrowthConstraint($P \cup_{IorS} y$,$ProjectedDB_i$,C)

Lemma 1. *In a pattern-growth based algorithm, if a pattern P satisfies a monotonic constraint, it is useless to test the extended patterns of P because all the extended patterns of P satisfy the constraint.*

This lemma is inferred from the monotonic constraint definition.

Lemma 2. *In a pattern-growth based algorithm, if a pattern P does not satisfy an anti-monotonic constraint, it is useless to generate the extended patterns of P because all the extended patterns of P do not satisfy the constraint.*

Again, this lemma is trivial to prove by referring to the definition of an anti-monotonic constraint.

These lemma are precious because they avoid to generate the extended patterns of P when P satisfies an (anti-)monotonic constraint. Monotonic constraints allow to prune patterns before the mining process (by removing some sequences from the SDB). Anti-monotonic constraints are more useful by allowing an efficient pruning during the mining process.

Other constraints handled by CloSPEC are not monotonic or anti-monotonic. However, a nice property of the pattern growth approach is to enable similar pruning techniques. Introduced in [10], prefix-monotonic and prefix-anti-monotonic constraints have the same behavior as respectively monotonic and anti-monotonic constraints.

Table 2 summarizes the properties of constraints tackled by our approach. Most of these constraints are defined in [10]. The association constraint is not included in [10] and is neither monotonic nor anti-monotonic. This constraint ensures that a relation between two classes of items must be satisfied. It is particularly useful in applications such as the case study presented in Section 4.1.

3.3 Closure Computation

Combining the computation of the closure of a pattern with constraints like the gap is a tricky task. As illustrated by the following example coming from Table 1, the anti-monotonic property of the support is lost when the support

Constraint	Monotonic	Anti-monotonic	Prefix-anti-monotonic
Membership	Yes	No	No
Maximal support	Yes	No	No
Minimal length	Yes	No	No
Minimal scope	Yes	No	No
Minimal support	No	Yes	Yes
Maximal scope	No	Yes	Yes
Maximal length	No	Yes	Yes
Gap	No	No	Yes
Association	No	No	No

Table 2: Monotonic and anti-monotonic properties of constraints

is combined with the gap. Let us consider a gap constraint $[0, 1]$ and $minsup = 2$. With data on Table 1, the support of the closed pattern $P_1 = \langle(a)\ (b)\ (c)\rangle$ is 3 and the support of the closed pattern $P_2 = \langle(a)\ (c)\rangle$ is 2 (in Sequence 3, there are two items between a and c and thus Sequence 3 does not support P_2). Thus $P_2 \preceq P_1$ whereas $sup(P_2) < sup(P_1)$. In this example, we see that the combination of the support and gap constraints does not give an anti-monotonic constraint. As the anti-monotonic property of support is usually used to compute the closed patterns, we understand that we have to design other techniques. In the literature, the problem is circumvented by technique such a new definition of the closure like in the GAPBIDE algorithm [7] or CTSP [8]. In those approaches the notion of closed pattern is based on the contiguously inclusion between patterns leading to recovering the anti-monotonic property of support. However, this new definition provides extra patterns that are not useful. Following our running example, GAPBIDE produces the patterns $P_3 = \langle(b)\ (c)\ (d)\rangle$ and $P_4 = \langle(b)\ (d)\rangle$ whereas $P_4 \preceq P_3$ and these two patterns have the same support: P_4 should not be produced. Another drawback of GAPBIDE is that it is not able to address itemsets, only simple items in sequences are tackled [7]. On the contrary, CloSPEC keeps the usual closure and thus avoiding redundancy.

This section shows how CloSPEC computes the usual closure for itemset sequences and various constraints given by the user. The closure computation of a sequential pattern with CloSPEC has two steps. The first step is the computation of the extended patterns of a sequential pattern (Section 3.3.1). The second step manages the closed sequential patterns in the hash-table $ClosedHash$ which contains trees of closed sequential patterns (Section 3.3.2).

3.3.1 Right and Left Extension Computations of a Sequential Pattern

To check if a sequential pattern is closed, we need to compute its possible extended patterns w.r.t. the SDB. Let us consider the SDB which contains $S_1 = \langle\ (a)\ (d\ e)\ (a\ b\ c\ d)\ (e)\ \rangle$ and $S_2 = \langle\ (a\ b)\ (d)\ (a\ b\, c\ d)\ (e\ f)\ \rangle$; and the pattern $P_5 = \langle\ (d)\ (b\ c)\ \rangle$. The possible extended patterns of P_5 w.r.t SDB are:

- with a left I-extension: none ;
- with a left S-extension, two patterns: $\langle\ (a)\ (d)\ (b\ c)\ \rangle$ and $\langle\ (b)\ (d)\ (b\ c)\ \rangle$;
- with a right I-extension, one pattern: $\langle\ (d)\ (b\ c\ d)\ \rangle$;
- with a right S-extension, two patterns: $\langle\ (d)\ (b\ c)\ (e)\ \rangle$ and $\langle\ (d)\ (b\ c)\ (f)\ \rangle$.

The new pattern obtained by adding the item d with the right I-extension has the same support as P_5, thus P_5 is not closed.

Dealing with itemsets instead of items in sequences implies to modify the pruning step to take into account the occurrences of sequential patterns. If there is a pattern P' such that **for each occurrence** of a pattern $P_A = \langle\ I_1 \dots I_n\rangle$ with $I_1 = (\ i_1 \dots i_k)$ we get: **either** $P' = \langle I'_1 \dots I_n\rangle$ with $I'_1 = (\mathbf{x}\ i_1 \dots i_k)$, **or** $P' = \langle(\mathbf{x})\ I_1 \dots I_n\rangle$, with x an item, then P_A can be safely pruned (step 1 in Algorithm 2).

3.3.2 Insertion in Tree of Closed Sequential Patterns

The left and right extension computation (I-extension and S-extension) allows a first pruning by reducing the number of patterns. It follows the principle used by the GAPBIDE algorithm (the latter only tackles the gap constraint). However, unclosed patterns may still exist as illustrated by the following example. Let us consider the SDB which contains $S_3 = \langle\ (a)(a\ b\ c)(c)\ \rangle$ and $S_4 = \langle\ (a)(a\ b\ c)(c\ e)\ \rangle$; and the pattern $P_6 = \langle\ (a)(c)\ \rangle$. P_6 has neither left nor right extension. However, P_6 is not closed because there exists the pattern P_7=$\langle\ (a)(a\ b\ c)(c)\ \rangle$ with the same support and $P_6 \preceq P_7$.

To solve this issue, we introduce an hash-table *ClosedHash* whose key is the support of a pattern and the value a lexicographical tree gathering all sequential patterns with the same support as the key value of the hashtable, satisfying all the constraints and with no extended patterns. Three situations are encountered to insert a pattern P in the tree:

1. if P is included in an already existing pattern in the tree, nothing to do ;
2. if it exists in the tree a pattern P' such that $P' \preceq P$, we need to reorganize the tree in order to add the pattern ;
3. if P is not included in the tree, we need to add it.

For instance, let us consider the tree in Figure 1(a). This tree[5] contains the current patterns having a support = 2. The three situations described above are respectively depicted by Figures 1(b), (c) and (d). Added patterns from Figure 1(a): $\langle\ (a)(e)\ \rangle$, $\langle\ (b)(a)(d)\ \rangle$, $\langle\ (d)(e)\ \rangle$ are depicted by the dark gray nodes. Finally, a closed sequential pattern is given by browsing the tree from the root node to a leaf node. As a result, all paths of all trees of the hash-table from the root node to the leaf nodes provide the correct and complete set of closed sequential patterns and then also satisfying the given multiple constraints.

[5]In order to simplify the example, we only present the insertion of sequential patterns of *items* (instead of itemsets).

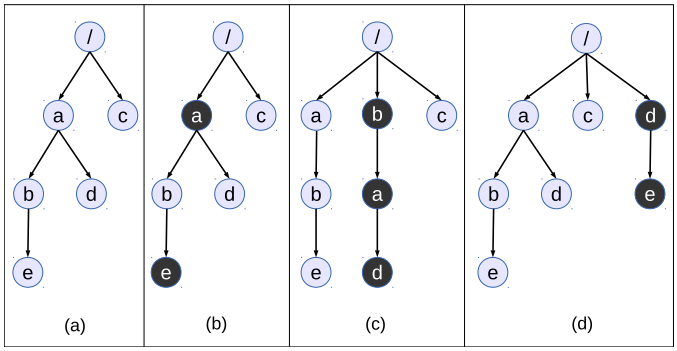

Figure 1: Insertion examples of patterns in a tree of closed patterns

4. EXPERIMENTATIONS

This section presents the extraction of closed sequential patterns under multiple constraints to discover relations between genes and rare diseases. Experiments have been conducted on texts from biological and medical literature.

4.1 Case Study

4.1.1 Settings

We experiment our method on a corpus created from the PubMed database by using the HUGO[6] dictionary and the Orphanet dictionary to query the database to get sentences containing at least one gene name and one disease name. 17,527 sentences were extracted in this way. We labelled the gene and rare disease (RD) names thanks to these 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. From the 17,527 sentences, we randomly extract 200 sentences as a testing corpus, the remaining sentences being the training corpus.

4.1.2 Sequential Pattern Extraction

Sequences of the SDB are the sentences of the training corpus: an itemset corresponds to a word of the sentence. We carry out a Part-Of-Speech tagging of the sentences thanks to the TreeTagger tool [12]. 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 approaches, relations (e.g. gene-disease relations) are not annotated, but 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* such as the *membership* and the *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 them.

[6]www.genenames.org

- *The minimal frequency.* Three values of minimal frequency have been experimented: 0.5% (88 sequences), 0.2% (35 sequences), and 0.05% (8 sequences).
- *The minimal length.* The aim of this constraint is to remove sequential patterns that are too small with respect to the number of items (number of words) to provide relevant linguistic patterns. We tested this constraint with a value set to 4 and without this constraint (i.e., no minimal size value).
- *The membership.* This constraint enables to filter out sequential patterns that do not contain some selected items. For example, we expressed that the extracted patterns must contain at least three items expressing the linguistic relation: $GENE$, $DISEASE$ and (noun or verb).
- *The association.* This constraint expresses that all sequential patterns must satisfy a relation between two classes of items. As an example, this constraint enables to express that pattern containing a verb item or a noun item must also contain its grammatical category or its lemma.
- *The maximal scope.* This constraint corresponds to the maximal value of the linguistic scope of a pattern. We set a maximal scope value of 20. It means that the maximal number of itemsets between the first itemset and the last itemset of pattern having gene-RD relationships is 20 (corresponding to 20 words in the sentence).
- *The gap.* We conducted experiments with a gap value (chosen empirically at [0,10]) and without this constraint.
- *The closure.* As already said, in order to exclude redundancy between patterns, we used *closed* patterns.

4.1.3 Applying patterns

The patterns extracted from the training corpus are applied onto the testing corpus as linguistic patterns to discover relations between genes and rare diseases.

4.2 Quantitative Results

This section reports experiments showing the impact of the constraints on the number of extracted patterns and the

quality of the results. We also provide quantitative results w.r.t. CloSpan[7] a well-known and state-of-the-art algorithm to extract closed sequential patterns [16].

4.2.1 Impact of Constraints

Table 3 indicates the number of extracted sequential patterns with respect to several values of gap, minsup and minlgth. Table 4 gives the number of extracted sequential patterns that are validated by an expert. In order to simplify the pattern validation, we first gather sequential patterns by nouns and by verbs. Then, the expert has to validate nouns and verbs expressing the notion of causality (with the possibility to directly access to sequential patterns having the given noun or verb). Finally, only sequential patterns having a verb or a noun expressing the notion of causality are considered like validated sequential patterns.

As expected on this example, the constraint with the most important impact on the reduction of the number of patterns is the support threshold minsup (reduction between 80% and 90%). This is easily explained by the choice of the threshold values. The two other constraints have a rather minor impact on the number of patterns (reduction between 2% and 8%). Note that the number of patterns with a gap constraint can be higher than the number of patterns extracted with no gap constraint. The explanation is the application of the closure operator after the application of the gap constraint.

	gap=[0,10]		no gap	
minsup	minlgth=4	no minlgth	minlgth=4	no minlgth
0.50%	22 794	24 888	22 084	23 823
0.20%	126 777	133 533	130 579	138 175
0.05%	1 493 914	1 530 085		

Table 3: Number of extracted sequential patterns with respect to three constraints: minsup, gap and minlgth.

	gap=[0,10]		no gap	
minsup	minlgth=4	no minlgth	minlgth=4	no minlgth
0.50%	6 310	6 346	6 156	6 193
0.20%	54 429	54 512	56 290	56 404
0.05%	416 786	416 533		

Table 4: Number of validated sequential patterns with respect to three constraints: minsup, gap and minlgth.

Table 5 presents the evaluation of the validated patterns extracted according to the support and gap constraints. Table 6 gives the evaluation of the validated patterns extracted with the support and minimum length constraints.

About the support threshold, the lower the support threshold is, the greater the recall. Indeed, the lower the support threshold is, the greater the number of extracted patterns. It implies potentially more discovered biomedical relations in the testing corpus and thus less false negative errors. About the gap constraint, two cases arise. When the number of patterns is greater with a gap constraint than without gap constraint, the recall value can be greater without gap constraint (see minsup=0.5%). When the number of patterns is lower with a gap constraint than without gap constraint, some irrelevant patterns can be extracted without gap constraint, which implies false positive errors and thus the precision value can be greater with a gap constraint (see minsup=0.2%). About the minimum length constraint, the longer the pattern is, the greater the recall, but the lower the precision.

[7]We use the implementation of Illimine available there: `http://illimine.cs.uiuc.edu`

minsup	gap	recall	precision	f-score
0.50%	[0,10]	0.37	0.67	0.48
0.50%	no gap	0.46	**0.69**	0.55
0.20%	[0,10]	0.50	0.65	0.56
0.20%	no gap	0.53	0.64	0.58
0.05%	[0,10]	**0.65**	0.66	**0.65**

Table 5: Evaluation of the validated patterns extracted with a support threshold and a gap constraint.

minsup	minlgth	recall	precision	f-score
0.50%	all	0.37	0.67	0.48
0.50%	4	0.36	0.68	0.47
0.20%	all	0.50	0.65	0.56
0.20%	4	0.48	0.67	0.56
0.05%	all	0.65	0.66	0.65
0.05%	4	0.64	0.66	0.65

Table 6: Evaluation of the validated patterns extracted with a support threshold and a minimum length constraint.

4.2.2 Comparison with CloSpan

CloSpan being one of the state-of-the-art algorithms to extract closed sequential patterns [16], it appears interesting to provide results on a quantitative comparison between CloSpan and CloSPEC. Nevertheless, CloSpan extracts only closed sequential patterns according to the minimal support constraint. Thus, CloSpan is unable to provide the various sets of patterns under multiple constraints processed by CloSPEC (except if a post-treatment is performed on the output of CloSpan). On the following, we compare, according to the number of extracted patterns and the runtime, CloSPEC (which produces only useful patterns according to the user's defined constraints) and CloSpan which can be considered as a baseline method.

The dataset for these experiments is the testing corpus presented in Section 4.1. In order to evaluate the interest of the different constraints, three versions of our algorithm are used:

- CloSPEC M: the proposed algorithm with monotonic constraints: minlgth=3 and two membership constraints (item GENE and item RARE_DISEASE) ;
- CloSPEC A: the proposed algorithm with anti-monotonic constraints: maxlgth=5 and gap=[2,4] ;
- CloSPEC M+A: the proposed algorithm with a monotonic constraint (minlgth=3) and an anti-monotonic constraint (gap=[2,4]).

minsup	CloSpan	CloSPEC M	CloSPEC A	CloSPEC M+A
30%	7 354	40	125	23
25%	16 473	40	125	23
20%	42 692	98	171	42
15%	137 178	277	265	81
10%	658 947	4 424	469	208
5%	8 469 431	58 285	951	544

Table 7: Number of extracted sequential patterns.

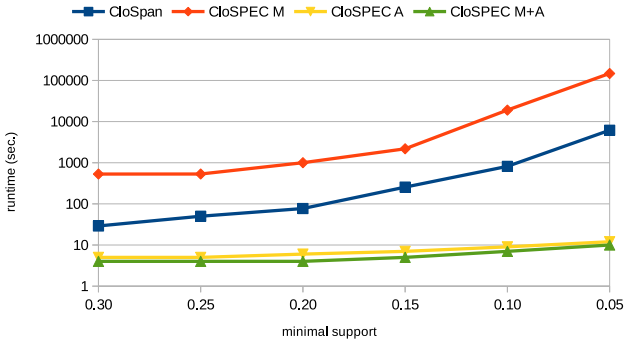

Figure 2: Runtimes according to the support.

Table 7 reports the number of extracted sequential patterns for each approach. A major result is that the number of extracted patterns is significantly reduced when applying constraints (the reduction is about 99%). This result is highly important when the patterns will be examined by a human like in the NLP application presented in Section 4.1. The advantage of our method w.r.t CloSpan is that the constraints are directly integrated into the computation without any post-treatment.

Figure 2 depicts the runtimes of each version of our method and CloSpan in order to compare the mutual benefits (note the logarithmic scale on the Y-axis). With anti-monotonic constraints, our method significantly reduces the runtime (up to 3 order of magnitude with low support values). The major result is the comparison between CloSPEC M+A and CloSpan: CloSPEC M+A runs faster than CloSpan (again, up to 3 orders of magnitude). This is a notable result because the task performed by CloSPEC M+A – which is a combination of monotonic and anti-monotonic constraints – is much more complex than the task done by CloSpan Recalling that a combination of a monotonic and an anti-monotonic constraints does not provide a constraint with (anti-)monotonic property.

Taken as a whole, these experiments show the interest of CloSPEC which extracts useful patterns (according to the user's defined constraints) and in a shorter runtime than CloSpan.

5. CONCLUSION

In this paper, we have proposed CloSPEC, a method providing the correct and complete set of patterns satisfying conjunctions of various syntactic and symbolic constraints, including the closedness constraint. These constraints, such as the scope, length, gap or membership, are crucial for NLP tasks based on pattern mining, like the discovery of linguistic patterns highlighting gene–RD relationships. We have conducted experiments showing the impact of different kinds of constraints and quantifying the advantages (number of extracted patterns, running time) w.r.t. the state-of-the-art. The method is general and can be used in other applicative contexts. A further work is to combine in a general framework our method handling multiple constraints with techniques based on data reduction allowing to reduce the search space with specific kinds of constraints [1].

6. REFERENCES

[1] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Exante: Anticipated data reduction in constrained pattern mining. In *PKDD*, 2003.

[2] G. Dong and J. Pei. *Sequence Data Mining*, volume 33 of *Advances in Database Systems*. Kluwer, 2007.

[3] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *PAKDD*, 2014.

[4] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: sequential pattern mining with regular expression constraints. In *Proceedings VLDB'99*, 1999.

[5] A. Gomariz, M. Campos, R. Marín, and B. Goethals. ClaSP: An efficient algorithm for mining frequent closed sequences. In *PAKDD*, 2013.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, 2000.

[7] C. Li and J. Wang. Efficiently mining closed subsequences with gap constraints. In *SDM*, 2008.

[8] M.-Y. Lin, S.-C. Hsueh, and C.-W. Chang. Mining closed sequential patterns with time constraints. *J. Inf. Sci. Eng.*, 24(1):33–46, 2008.

[9] M. Nanni and C. Rigotti. Extracting trees of quantitative serial episodes. In *Proc. of KDID*, 2007.

[10] J. Pei, J. Han, and W. Wang. Constraint-based sequential pattern mining: the pattern-growth methods. *J. Intell. Inf. Syst.*, 28(2):133–160, 2007.

[11] M. Plantevit and B. Crémilleux. Condensed representation of sequential patterns according to frequency-based measures. In *IDA*, 2009.

[12] H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proc. of the Int. Conf. on New Methods in Language Processing*, 1994.

[13] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, pages 3–17, 1996.

[14] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *ICDE*, 2004.

[15] J. Wang, J. Han, and C. Li. Frequent closed sequence mining without candidate maintenance. *IEEE TKDE*, 19(8):1042–1056, 2007.

[16] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large databases. In *SDM*, 2003.

[17] M. J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *CIKM*. ACM, 2000.

[18] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, 2001.