Declarative mining of negative sequential patterns

Philippe Besnard, Thomas Guyet

Declarative mining of negative sequential patterns

Philippe Besnard [1] and **Thomas Guyet** [2]

Abstract. Declarative pattern mining consists in using declarative frameworks to solve pattern mining tasks. In this article, we address the task of mining negative sequential patterns in Answer Set Programming (ASP). A negative sequential pattern is specified by means of a sequence consisting of events to occur and of other events, called negative events, to be absent. For instance, containment of the pattern $\langle a\ \neg b\ c\rangle$ arises with an occurrence of a and a subsequent occurrence of c but no occurrence of b in between. Recent results shed light on the ambiguity of such a seemingly intuitive notation, exhibited three semantics of the negative events and proposed alternative notations for them. In this article, we propose Answer Set Programming encodings of these three semantics in order to extract frequent negative sequential patterns from a set of sequences. It relies on previous encodings of frequent sequential pattern mining. We experiment with our encoding on synthetic data and compare the numbers of extracted patterns and the computing time obtained for each kind of negation. Surprisingly, the semantics that has the best algorithmic properties for pattern mining is not associated to an encoding that is the most efficient.

1 Introduction

Pattern mining [11] is a data mining task that consists in extracting interesting structured patterns from a set of structured examples. The standard pattern mining tasks are itemset mining, sequence mining and graph mining. The interestingness measure of a pattern is, in most of the algorithms, the number of its occurrences in the set of examples. Given a threshold k, interesting patterns are those that occur at least in k examples. In this case, the task is known as *frequent pattern mining* for which many algorithms have been proposed. Most of the efficient algorithmic solutions rely on an antimonotonicity property of the support: the larger the pattern, the fewer it occurs.

Declarative pattern mining aims at encoding pattern tasks in a declarative framework, and more specifically the frequent pattern mining tasks. Declarative pattern mining addressed the tasks of frequent itemset mining [5, 13, 12], frequent sequential patterns [15, 2, 1] or frequent graph mining [6]. Different declarative frameworks have been explored: Constraint Programming (CP) [5, 15, 6], SAT [12, 1] and Answer Set Programming (ASP) [13, 2]. Declarative pattern mining does not expect to be competitive with dedicated algorithms, but to take advantage of the versatility of declarative frameworks to propose pattern mining tools that could exploit background knowledge during the mining process to extract less but meaningful patterns.

In this work, we explore negative sequential patterns [8, 10, 17]. Standard sequential pattern mining algorithms [14] extract sequential patterns that frequently occur in a set of sequences. A sequential pattern is a sequence of events or items. For example, the sequential pattern $\langle a\ c\ d\rangle$ is read as "a occurs and then c occurs and finally d occurs". Negative sequential patterns are sequential patterns that also specify non-occurring events. Intuitively, the syntax of a simple negative sequential pattern is as follows: $\langle a\ \neg b\ c\rangle$. This pattern is read as "a occurs and then c occurs, but b does not occur in between". A negative sequential pattern can also be the premise of a rule. Guyet and Besnard [8] highlighted that $\neg$ symbol has several semantics but only few of them have good properties for pattern mining.

In this article, we propose an ASP encoding of the mining of frequent negative sequential patterns. Our encoding relies on previous work on frequent sequential pattern mining in ASP [9]. The objective of this article is also to illustrate the semantics of negative sequential patterns through declarative encodings.

2 Background

2.1 Sequential pattern mining

Our terminology on sequence mining follows the one in [15]. Throughout this article, $[n] = \{1, \ldots, n\}$ denotes the set of the first n positive integers.

Let Σ be the set of items (alphabet). An *itemset* $A = \{a_1, a_2, \cdots, a_m\} \subseteq \Sigma$ is a finite set of items. The size of A, denoted $|A|$, is m. A *sequence* $\boldsymbol{s}$ is of the form $\boldsymbol{s} = \langle s_1\ s_2\ \cdots\ s_n\rangle$ where each s_i is an itemset. n is the length of the sequence.

A *database* $\mathcal{D}$ is a multiset of sequences over Σ.A sequence $\boldsymbol{s} = \langle s_1\ \ldots\ s_m\rangle$ with $s_i \in \Sigma$ is *contained* in a sequence $\boldsymbol{t} = \langle t_1\ \ldots\ t_n\rangle$ with $m \leq n$, written $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$, if $s_i \subseteq t_{e_i}$ for $1 \leq i \leq m$ and an increasing sequence $(e_1\ \ldots\ e_m)$ of positive integers $e_i \in [n]$, called an *embedding* of $\boldsymbol{s}$ in $\boldsymbol{t}$. For example, we have $\langle a\ (cd)\rangle \sqsubseteq \langle a\ b\ (cde)\rangle$ relative to embedding $(1, 3)$. (cd) denotes the itemset made of items c and d.

Given a database $\mathcal{D}$, the *cover* of a sequence $\boldsymbol{p}$ is the set of sequences in $\mathcal{D}$ that contain $\boldsymbol{p}$: $cover(\boldsymbol{p}, \mathcal{D}) = \{\boldsymbol{t} \in \mathcal{D} \mid \boldsymbol{p} \sqsubseteq \boldsymbol{t}\}$. The number of sequences in $\mathcal{D}$ containing $\boldsymbol{p}$ is called its *support*, that is, $support(\boldsymbol{p}, \mathcal{D}) = |cover(\boldsymbol{p}, \mathcal{D})|$. For an integer k, *frequent sequence mining* is about discovering all sequences $\boldsymbol{p}$ such that $support(\boldsymbol{p}, \mathcal{D}) \geq k$. We often call $\boldsymbol{p}$ a (sequential) *pattern*, and k is also referred to as the (minimum) *frequency threshold*.

2.2 Answer set programming

A *logic program* is a set of rules of the form

$$\mathtt{a_0} \mathtt{:-} \mathtt{a_1}; \ldots; \mathtt{a_m}; \mathtt{not}\ \mathtt{a_{m+1}}; \ldots; \mathtt{not}\ \mathtt{a_n}. \qquad (1)$$

where each $\mathtt{a_i}$ is a propositional atom for $\mathtt{0} \leq \mathtt{i} \leq \mathtt{n}$ and $\mathtt{not}$ stands for *default negation*. If $\mathtt{n} = \mathtt{0}$, rule (1) is called a *fact*. If $\mathtt{a_0}$ is omitted, (1) represents an integrity constraint. Semantically, a

[1] CNRS / IRIT, France, email: besnard@irit.fr
[2] Institut Agro / IRISA UMR6074, France, email: thomas.guyet@irisa.fr

```
seq(1,1,d). seq(1,2,a). seq(1,2,b). seq(1,3,c).
seq(2,1,a). seq(2,2,c). seq(2,3,b). seq(2,4,c).
seq(3,1,a). seq(3,2,b). seq(3,2,c). seq(3,3,a).
seq(4,1,a). seq(4,1,b). seq(4,2,c).
seq(5,1,a). seq(5,2,c).
seq(6,1,b).
seq(7,1,c).
```

Listing 1. Facts specifying a database of sequences

logic program induces a collection of so-called *answer sets*, which are distinguished models of the program determined by answer sets semantics; see [4] for details. To facilitate the use of ASP in practice, several extensions have been developed. First of all, rules with variables are viewed as shorthands for the set of their ground instances. Further language constructs include *conditional literals* and *cardinality constraints* [16]. The former are of the form $\mathtt{a} : \mathtt{b}_1, \ldots, \mathtt{b}_m$, the latter can be written as $\mathtt{s}\ \{\mathtt{c}_1, \ldots, \mathtt{c}_n\}\ \mathtt{t}$, where $\mathtt{a}$ and $\mathtt{b}_i$ are possibly default negated literals and each $\mathtt{c}_j$ is a conditional literal; $\mathtt{s}$ and $\mathtt{t}$ provide lower and upper bounds on the number of satisfied literals in a cardinality constraint. The practical value of both constructs becomes more apparent when used in conjunction with variables. For instance, a conditional literal like $\mathtt{a}(\mathtt{X}) : \mathtt{b}(\mathtt{X})$ in a rule's antecedent expands to the conjunction of all instances of $\mathtt{a}(\mathtt{X})$ for which the corresponding instance of $\mathtt{b}(\mathtt{X})$ holds. Similarly, $2\ \{\mathtt{a}(\mathtt{X}) : \mathtt{b}(\mathtt{X})\}\ 4$ holds whenever between two and four instances of $\mathtt{a}(\mathtt{X})$ (subject to $\mathtt{b}(\mathtt{X})$) are true. Specifically, we rely in the sequel on the input language of the ASP system *clingo* [3].

3 ASP-based Sequence Mining

In this section, we introduce the sequential pattern mining tasks in ASP. The following programs correspond to the notion of *skip-gaps* encodings in [2, 7]. Gebser et *al.* [2] uses a different encoding strategy for frequent sequential pattern that is not suitable for negative patterns.

3.1 Fact format

We represent a database $\mathcal{D}$ in terms of facts $\mathtt{seq}(\boldsymbol{t},p,e)$, saying that item e occurs at position p in a sequence $\boldsymbol{t}$. For instance, Listing 1 specifies a database of seven sequences:

Term	1	2	3	4	5	6	7
Seq.	$\langle d\ (ab)\ c\rangle$	$\langle a\ c\ b\ c\rangle$	$\langle a\ (bc)\ a\rangle$	$\langle (ab)\ c\rangle$	$\langle a\ c\rangle$	$\langle b\rangle$	$\langle c\rangle$

Considering the minimal frequency threshold $k = 2$, one can check that $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle a\ b\rangle$, $\langle (ab)\rangle$, $\langle a\ c\rangle$, $\langle b\ c\rangle$, and $\langle (ab)\ c\rangle$ are the frequent patterns of the database.

3.2 Mining frequent sequences: basic encoding

The encoding principle for frequent sequence mining follows the one of [13], that is, each answer set comprises a single pattern of interest. In contrast to itemset mining, however, we need to take the order of items in a pattern into account to determine its support and check its frequency.

Listing 2 provides our basic encoding of frequent sequence mining borrowed from [9]. It relies on two parameters: *max* determines a maximum length for patterns of interest, and *k* specifies the minimal frequency threshold. An answer set then represents a frequent pattern $\boldsymbol{p} = \langle p_1\ \ldots\ p_m\rangle$ such that $m \in [max]$ by atoms $\mathtt{pat}(m,p_1)$, $\ldots$, $\mathtt{pat}(1,p_m)$. That is, the first argument expresses the positions of items. For instance, the atoms $\mathtt{pat}(1,a)$, $\mathtt{pat}(2,b)$, and $\mathtt{pat}(2,c)$ stand for the (frequent) pattern $\langle a\ (bc)\rangle$ of the database given in Listing 1.

```
1  item(I) :- seq(T,P,I).

3  slot(1).
4  { slot(X+1) } :- slot(X); X<max.
5  1{ pat(X,E) : item(E) } :- slot(X).

7  patlen(L) :- pat(L,_); not pat(L+1,E):item(E).

9  emb(T,P,1) :- seq(T,P,E):pat(1,E); seq(T,P,_).
10 emb(T,P,X+1) :- emb(T,Q,X); seq(T,P,_); Q<P;
11                 seq(T,P,E):pat(X+1,E);
12                 patlen(L); X<L.

14 cover(T) :- patlen(L); emb(T,_,L).

16 :- not k { cover(T) }.
```

Listing 2. Basic encoding of frequent sequence mining

A complete specification of atoms used in our ASP encodings along with their meanings is given in Table 1. In more detail, the rule in Line 1 of Listing 2 provides items occurring in a given database $\mathcal{D}$. Lines 3 and 4 provides the slots for the itemset of a pattern with a varying length $m \in [max]$. Line 7 evaluates the concrete length of the candidate pattern. The (choice) rule in line 5 allows for picking a subset of Σ per slot to build a pattern. As a result, the rules from lines 1 to 5 establish a unique representation for a candidate pattern whose frequency remains to be checked.

To this end, the rules from Line 9 to 12 traverse each sequence $\boldsymbol{t} = \langle t_1\ \ldots\ t_n\rangle$ in $\mathcal{D}$ to derive atoms of the form $\mathtt{emb}(\boldsymbol{t},p,x)$ which represent all the embeddings of the pattern in a sequence. As noted in Table 1, for $p \in [n]$ and $x \in [max]$, such an atom expresses that it exists $(\epsilon_i)_{i\in[p-1]}$ such that $p_i \subseteq t_{\epsilon_i}$ for all $i \in [p-1]$ and $p_x \subseteq t_p$, where $p \in [n]$ and $x \in [m]$. Thus, it also means that $\langle p_1\ \ldots\ p_x\rangle \sqsubseteq \langle t_1\ \ldots\ t_p\rangle$. Line 9 locates the occurrences of the first itemset in the sequence. The expression $\mathtt{seq}(t,p,e)\mathtt{:pat}(1,e)$ ensures that every $e \in p_1$ must hold at position p of the sequence t. Line 10 finds out locations for p_x, the x-th itemset of the pattern, from a position q such that $q \geq \epsilon_i$ for all $i \in [p-1]$, where $(\epsilon_i)_{i\in[p-1]}$ is an embedding of $\langle p_1\ \ldots\ p_{x-1}$. The atoms $\mathtt{emb}(\boldsymbol{t},p,x)$ are further inspected, line 14, to determine the cover of a candidate pattern $\boldsymbol{p}$. In these lines, the sequence is covered when a complete embedding has been built, *i.e.* up to the length of the pattern. The frequency of $\boldsymbol{p}$ is then checked in Line 16, where the cardinality constraint "$k\ \{\ \mathtt{cover}(\boldsymbol{t})\ \}$" over atoms $\mathtt{cover}(\boldsymbol{t})$, signaling $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$, expresses that at least the frequency threshold k many sequences $\boldsymbol{t}$ must include $\boldsymbol{s}$.

Example 1. *For the database in Listing 1, the pattern $\langle a\ (bc)\rangle$ leads to atoms* `emb(3,1,1)`*,* `emb(3,1,3)` *and* `emb(3,2,2)` *among which* `emb(3,2,2)` *in turn yields* `cover(3)` *for the sequence $\langle a\ (bc)\ a\rangle$ denoted by 3. There is not any sequence for which a complete embedding can be built, and $\langle a\ (bc)\rangle$ turns out to not be frequent, given the threshold $k = 2$.*

4 Negative sequential patterns

In this section, we introduce the notion of negative sequential pattern by giving their syntax and semantics borrowed from [8].

Table 1. Atoms representing a pattern $\boldsymbol{p} = \langle p_i \rangle_{1 \leq i \leq m}$, where $1 \leq m \leq max$, and a given multiset $\mathcal{D}$ of sequences $\boldsymbol{t} = \langle t_j \rangle_{1 \leq j \leq n}$

Atom	Meaning
`slot(`x`)`	$1 \leq x \leq \mathtt{m}$ refers to the position x of an item s_x in $\boldsymbol{p}$
`pat(`x`,`e`)`	$p_x = e$ is the item at position x in $\boldsymbol{p}$, where $x \in [m]$
`item(`e`)`	item e belongs to some $\boldsymbol{t} \in \mathcal{D}$
`cover(`$\boldsymbol{t}$`)`	$\langle p_1 \ \ldots \ p_m \rangle \sqsubseteq \langle t_1 \ \ldots \ t_n \rangle$, that is, $\boldsymbol{p} \sqsubseteq \boldsymbol{t}$
`emb(`$\boldsymbol{t}$`,`p`,`x`)`	$\exists (e_i)_{i \in [p-1]}$ such that $p_i \subseteq t_{e_i}$ for all $i \in [p-1]$ and $p_x \subseteq t_p$, where $p \in [n]$ and $x \in [m]$.

Definition 1 (Negative sequential patterns (NSP))**.** *A negative sequential pattern* $\boldsymbol{p} = \langle p_1 \ \neg q_1 \ p_2 \ \neg q_2 \ \cdots \ p_{n-1} \ \neg q_{n-1} \ p_n \rangle$ *is a finite sequence where* $p_i \in 2^\Sigma \setminus \{\emptyset\}$ *for all* $i \in [n]$ *and* $q_i \in 2^\Sigma$ *for all* $i \in [n-1]$.

$\boldsymbol{p}^+ = \langle p_1 \ \ldots \ p_n \rangle$ *is called the positive part of the NSP.*

It can be noticed that Definition 1 introduces syntactic limitations on negative sequential patterns that are commonly encountered in the state of the art [17]: 1) a pattern can neither start or finish by a negative itemset, 2) a pattern cannot have two successive negative itemsets. But a pattern can have successive positive itemsets considering that a negative itemsets can be empty.

Example 2. *This example illustrates the notations introduced in Definition 1. Consider* $\Sigma = \{a, b, c, d\}$ *and* $\boldsymbol{p} = \langle a \ \neg(bc) \ (ad) \ d \ \neg(ab) \ d \rangle$. *Let* $p_1 = \{a\}$, $p_2 = \{ad\}$, $p_3 = \{d\}$, $p_4 = \{d\}$ *and* $q_1 = \{bc\}$, $q_2 = \emptyset$, $q_3 = \{ab\}$. $\boldsymbol{p}^+ = \langle a \ (ad) \ d \ d \rangle$.

4.1 Semantics of negative sequential patterns

The semantics of negative sequential patterns relies upon *negative containment*: a sequence $\boldsymbol{s}$ contains pattern $\boldsymbol{p}$ iff $\boldsymbol{s}$ contains a sub-sequence $\boldsymbol{s}'$ such that every positive itemset of $\boldsymbol{p}$ is included in some itemset of $\boldsymbol{s}'$ in the same order and for any negative itemset $\neg q_i$ of $\boldsymbol{p}$, q_i is *not included* in any itemset occurring in the sub-sequence of $\boldsymbol{s}'$ located between the occurrence of the positive itemset preceding $\neg q_i$ in $\boldsymbol{p}$ and the occurrence of the positive itemset following $\neg q_i$ in $\boldsymbol{p}$.

It turns out that the notion of negative containment has different semantics depending on the definition of "*not included*" itemsets and on the itemsets of the sequence to consider.

We introduce two relations comparing two itemsets $P \in 2^\Sigma \setminus \{\emptyset\}$ and $I \in 2^\Sigma$:

- partial non inclusion: $P \not\subseteq_G I \Leftrightarrow \exists e \in P, e \notin I$
- total non inclusion: $P \not\subseteq_D I \Leftrightarrow \forall e \in P, e \notin I$

Partial non-inclusion means that $P \setminus I$ is non-empty while total non-inclusion means that P and I are disjoint. By convention, $\emptyset \not\subseteq_D I$ and $\emptyset \not\subseteq_G I$ for all $I \subseteq \Sigma$.

In the sequel, we denote the general form of itemset non-inclusion by the symbol $\not\subseteq_*$, meaning either $\not\subseteq_G$ or $\not\subseteq_D$.

Intuitively, partial non-inclusion identifies the itemset P with a disjunction of negative constraints, *i.e.* at least one of the items (of P) has to be absent from I, and total non-inclusion consider the itemset P as a conjunction of negative constraints: all items (of P) have to be absent from I.

Choosing one non-inclusion interpretation or the other has consequences on extracted patterns as well as on pattern search. Let us illustrate this with the following dataset of sequences:

$$\mathcal{D} = \left\{ \begin{array}{l} \boldsymbol{s}_1 = \langle (bc) \ f \ a \rangle \\ \boldsymbol{s}_2 = \langle (bc) \ (cf) \ a \rangle \\ \boldsymbol{s}_3 = \langle (bc) \ (df) \ a \rangle \\ \boldsymbol{s}_4 = \langle (bc) \ (ef) \ a \rangle \\ \boldsymbol{s}_5 = \langle (bc) \ (cdef) \ a \rangle \end{array} \right\}.$$

Table 2 compares the support of patterns under the two semantics of itemset non-inclusion. Since the positive part of $\boldsymbol{p}_2$ is in $\boldsymbol{s}_2$, $\boldsymbol{p}_2$ occurs in the sequence iff $(cd) \not\subseteq_* (cf)$. As for total non-inclusion, it is false that $(cd) \not\subseteq_D (cf)$ because c occurs in (cf), and thus $\boldsymbol{p}_2$ does not occur in $\boldsymbol{s}_2$. As for partial non-inclusion, it is true that $(cd) \not\subseteq_G (cf)$, because d does not occur in (cf), and thus $\boldsymbol{p}_2$ occurs in $\boldsymbol{s}_2$.

Table 2. Lists of sequences in $\mathcal{D}$ supported by negative patterns $(\boldsymbol{p}_i)_{i=1..4}$ under the total and partial non-inclusion relations. Each pattern has the form $\langle b \ \neg q_i \ a \rangle$ where q_i are itemsets such that $q_i \subset q_{i+1}$.

	partial non-inclusion $\not\subseteq_G$	total non-inclusion $\not\subseteq_D$
$\boldsymbol{p}_1 = \langle b \ \neg c \ a \rangle$	$\{\boldsymbol{s}_1, \boldsymbol{s}_3, \boldsymbol{s}_4\}$	$\{\boldsymbol{s}_1, \boldsymbol{s}_3, \boldsymbol{s}_4\}$
$\boldsymbol{p}_2 = \langle b \ \neg (cd) \ a \rangle$	$\{\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, \boldsymbol{s}_4\}$	$\{\boldsymbol{s}_1, \boldsymbol{s}_4\}$
$\boldsymbol{p}_3 = \langle b \ \neg (cde) \ a \rangle$	$\{\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, \boldsymbol{s}_4\}$	$\{\boldsymbol{s}_1\}$
$\boldsymbol{p}_4 = \langle b \ \neg (cdeg) \ a \rangle$	$\{\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, \boldsymbol{s}_4, \boldsymbol{s}_5\}$	$\{\boldsymbol{s}_1\}$

Now, we formulate the notions of sub-sequence, non-inclusion and absence by extending the concept of embedding (see Section 2.1) to negative patterns.

Definition 2 (Strict and soft embeddings of negative patterns)**.** *Let* $\boldsymbol{s} = \langle s_1 \ \ldots \ s_n \rangle$ *be a sequence and* $\boldsymbol{p} = \langle p_1 \ \neg q_1 \ \ldots \ \neg q_{m-1} \ p_m \rangle$ *be a negative sequential pattern.*

An increasing[3] *tuple* $\boldsymbol{\epsilon} = (\epsilon_i)_{i \in [m]} \in [n]^m$ *is a soft-embedding of pattern* $\boldsymbol{p}$ *in sequence* $\boldsymbol{s}$ *iff:*

- $p_i \subseteq s_{e_i}$ *for all* $i \in [m]$
- $q_i \not\subseteq_* s_j$, *for all* $j \in [e_i + 1, e_{i+1} - 1]$ *and for all* $i \in [m-1]$

An increasing[3] *tuple* $\boldsymbol{e} = (e_i)_{i \in [m]} \in [n]^m$ *is a strict-embedding of pattern* $\boldsymbol{p}$ *in sequence* $\boldsymbol{s}$ *iff:*

- $p_i \subseteq s_{e_i}$ *for all* $i \in [m]$
- $q_i \not\subseteq_* \bigcup_{j \in [e_i+1, e_{i+1}-1]} s_j$ *for all* $i \in [m-1]$

Intuitively, the constraint of a negative itemset q_i is checked on the sequence's itemsets at positions in interval $[e_i + 1, e_{i+1} - 1]$, *i.e.* between occurrences of the two positive itemsets surrounding the negative itemset in the pattern. A soft embedding considers individually each of the sequence's itemsets of $[e_i + 1, e_{i+1} - 1]$ while a strict embedding consider them as a whole.

Example 3 (Itemset absence semantics)**.** *Let* $\boldsymbol{p} = \langle a \ \neg (bc) \ d \rangle$ *be a pattern and consider four sequences as follows:*

[3] By an increasing tuple $\boldsymbol{e}$, we mean a tuple such that $\epsilon_i < \epsilon_{i+1}$ (in particular, repetitions are not allowed).

Sequence	$\not\subseteq_D$ *strict*	$\not\subseteq_D$ *soft*	$\not\subseteq_G$ *strict*	$\not\subseteq_G$ *soft*
$\boldsymbol{s_1} = \langle a\ c\ b\ e\ d\rangle$				✓
$\boldsymbol{s_2} = \langle a\ (bc)\ e\ d\rangle$				
$\boldsymbol{s_3} = \langle a\ b\ e\ d\rangle$			✓	✓
$\boldsymbol{s_4} = \langle a\ e\ d\rangle$	✓	✓	✓	✓

Notice that each sequence contains a unique occurrence of $\boldsymbol{p}^+ = \langle a\ d\rangle$, the positive part of pattern $\boldsymbol{p}$. Considering soft-embedding and partial non-inclusion ($\not\subseteq_*:=\not\subseteq_G$), $\boldsymbol{p}$ occurs in $\boldsymbol{s_1}$, $\boldsymbol{s_3}$ and $\boldsymbol{s_4}$ but not in $\boldsymbol{s_2}$. Considering strict-embedding and partial non-inclusion, $\boldsymbol{p}$ occurs in $\boldsymbol{s_3}$ and $\boldsymbol{s_4}$. Indeed, items b and c occur between occurrences of a and d in $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$. Considering total non-inclusion ($\not\subseteq_*:=\not\subseteq_D$) and either type of embeddings, the absence of an itemset is satisfied if any of its items is absent. Hence, $\boldsymbol{p}$ occurs only in $\boldsymbol{s_4}$.

Let us now give some lemmata about the relations between the different types of embedding.[4]

Lemma 1. *In the case that $\not\subseteq_*$ is $\not\subseteq_D$, $\boldsymbol{e}$ is a soft-embedding iff $\boldsymbol{e}$ is a strict-embedding*

Lemma 2. *Let $p = \langle p_1\ \neg q_1\ \ldots\ \neg q_{n-1}\ p_n\rangle \in \mathcal{N}$ such that $|q_i| \leq 1$ for all $i \in [n-1]$, then $\boldsymbol{e}$ is a soft-embedding iff $\boldsymbol{e}$ is a strict-embedding.*

At this point, we have exhibited four semantics for negative pattern embeddings. Lemmata 1 and 2 concluded to the equivalence of soft and strict-embeddings when the itemset non-inclusion relation is $\not\subseteq_D$.

In [8], we proposed notations to disambiguate syntactically the different types of negation ($\neg(a_1,\ldots,a_{l_i})$ or $\neg\{a_1,\ldots,a_{l_i}\}$ or $\neg|a_1,\ldots,a_{l_i}|$). $]l,u[$ denotes the interval of integers $[l+1, u-1]$.

- $\neg(a_1,\ldots,a_{l_i})$ is evaluated as
$$\{a_1,\ldots,a_{l_i}\} \not\subseteq_G s_j,\ \forall j \in]e_i, e_{i+1}[\text{ for all } i \in [m-1] \quad (2)$$
Intuitively, you check that, in between s_{e_i} (*i.e.*, a match for p_i) and $s_{e_{i+1}}$ (i.e., a match for p_{i+1}), none of these s_j include all of $a_1,\ldots,a_{l_i}$.

- $\neg\{a_1,\ldots,a_{l_i}\}$ is evaluated as
$$\{a_1,\ldots,a_{l_i}\} \not\subseteq_G \bigcup_{j\in]e_i,e_{i+1}[} s_j \text{ for all } i \in [m-1] \quad (3)$$
Intuitively, we check that there exists some item in $a_1,\ldots,a_{l_i}$ that does not occur at all in between s_{e_i} (*i.e.*, a match for p_i) and $s_{e_{i+1}}$ (*i.e.*, a match for p_{i+1}).

- $\neg|a_1,\ldots,a_{l_i}|$ is evaluated as
$$\{a_1,\ldots,a_{l_i}\} \not\subseteq_D s_j,\ \forall j \in]e_i, e_{i+1}[\text{ for all } i \in [m-1] \quad (4)$$
Intuitively, we check that every item in $a_1,\ldots,a_{l_i}$ fails to occur in between s_{e_i} (*i.e.*, a match for p_i) and $s_{e_{i+1}}$ (*i.e.*, a match for p_{i+1}).

4.2 Multiple occurrences

Another point that determines the semantics of negative containment concerns the multiple occurrences of some pattern in a sequence: should at least one or should all occurrences of the pattern positive part in the sequence satisfy the non-inclusion constraints?

[4] Proofs of these Lemmata can be found in the extended version of [8].

The *strong* containment relation states that a negative pattern $\boldsymbol{p}$ occurs in a sequence $\boldsymbol{s}$ iff there exists at least one occurrence of the positive part of pattern $\boldsymbol{p}$ in sequence $\boldsymbol{s}$ and *every* such occurrence satisfies the negative constraints; the *strong* containment relation states $\boldsymbol{p}$ occurs in a sequence $\boldsymbol{s}$ iff there exists at least one occurrence of the positive part of pattern $\boldsymbol{p}$ in sequence $\boldsymbol{s}$ and *at least one* of these occurrences satisfies the negative constraints.

Example 4 (Strong vs weak occurrence semantics)**.** *Let $\boldsymbol{p} = \langle a\ b\ \neg c\ d\rangle$ be a pattern, $\boldsymbol{s_1} = \langle a\ b\ e\ d\rangle$ and $\boldsymbol{s_2} = \langle a\ b\ c\ a\ d\ e\ b\ d\rangle$ be two sequences. Thus, $\boldsymbol{p}^+ = \langle a\ b\ d\rangle$ occurs once in $\boldsymbol{s_1}$ hence there is no difference for occurrences of $\boldsymbol{p}$ in $\boldsymbol{s_1}$ under the two semantics. However, $\boldsymbol{p}^+$ occurs four times in $\boldsymbol{s_2}$ through embeddings $(1,2,5)$, $(1,2,8)$, $(1,7,8)$ and $(4,7,8)$. The first two occurrences do not satisfy the negative constraint ($\neg c$) but the last two occurrences do. Under the weak occurrence semantics, pattern $\boldsymbol{p}$ occurs in sequence $\boldsymbol{s_2}$ whereas it fails to do so under the strong occurrence semantics.*

5 ASP encodings of negative sequential patterns

In the following, we introduce new facts to encode negations in negative sequential patterns. The `pat(`x`,`e`)` atom now represents the positive part of a negative sequential pattern. The negated itemsets are encoded by terms of three predicates: `negpat/2` (for $\neg|\cdot|$), `cnegpat/2` (for $\neg\{\cdot\}$) and `csnegpat/2` (for $\neg(\cdot)$). For instance, the pattern $\langle a\neg|b|a\neg(bd)c\rangle$ is encoded by the facts `pat(1,a)`, `pat(2,a)`, `pat(3,c)`, `negpat(1,b)`, `csnegpat(2,b)` and `csnegpat(2,d)`.

The following choice rule illustrates the generation of negative itemsets of kind $\neg|\cdot|$. Similar rules yield the negative part of a candidate pattern for the two other kinds of negations.

```
{ negpat(X,E): item(E), not pat(X,E) } :-
                        slot(X), X<L, patlen(L).
```

Note that we assume that it is not possible to have several different kinds of negated itemsets at the same position. For instance, pattern $\langle a\neg|b|\neg\{c\}d\rangle$ is not possible.

In the next section, we give the rules for negative constraints. A *negative constraint* denotes a constraint upon an embedding $(\epsilon_i)_{i\in[n]}$ of the positive part that is expressed by a negated itemset in the candidate negative pattern. If all negative constraints are satisfied, then $(\epsilon_i)_{i\in[n]}$ is an embedding of the negative pattern. Then, it is further used to evaluate the containment of the candidate pattern $\boldsymbol{p}$ in a sequence $\boldsymbol{t}$.

The core of the encoding relies on the construction of an embedding. We adapt the definition of the predicate `emb/3` to negative sequential patterns as follow: let $\boldsymbol{p} = \langle p_1\ \neg q_1\ \ldots\ p_m\rangle$ be a negative sequential pattern, $\boldsymbol{t} = \langle t_1\ \ldots\ t_n\rangle$ be a sequence, and ν a Boolean value, then `emb(`$\boldsymbol{t}$`,`ρ`,`x`,`ν`)` means that $\exists(e_i)_{i\in[x-1]}$ such that $x \in [m]$, $\rho \in [n]$ and

1. $p_i \subseteq t_{e_i}$ for all $i \in [x-1]$,
2. $p_x \subseteq t_\rho$,
3. $\exists y \in [x-1]$ s.t. q_y is not satisfied for the partial embedding $(e_i)_{i\in[p-1]}$ if ν is true,
4. $\forall y \in [x-1]$ s.t. q_y is satisfied for the partial embedding $(e_i)_{i\in[p-1]}$ if ν is false,

Note that "q_y is not satisfied for the partial embedding $(e_i)_{i\in[p-1]}$" refers to the semantics of negation (see previous section) whatever the kind of occurrence (weak or strong).

```
1  emb(T,P,1,0)    :- seq(T,P,E):pat(1,E); seq(T,P,_).

3  emb(T,P,X+1,1) :- emb(T,Q,X,1); Q<P;
4                    seq(T,P,E):pat(X+1,E); seq(T,P,_);
5                    patlen(L); X<L.

7  emb(T,P,X+1,0) :- emb(T,Q,X,0);
8                    seq(T,P,E):pat(X+1,E); seq(T,P,_); P>Q;
9                    #count{ R : not seq(T,R,Ep), csnegpat(X,Ep), R=(Q+1)..(P-1)}=(P-Q-1);
10                   patlen(L); X<L.

12 emb(T,P,X+1,1) :- emb(T,Q,X,0);
13                   seq(T,P,E):pat(X+1,E); seq(T,P,_); P>Q;
14                   #count{ R : not seq(T,R,Ep), csnegpat(X,Ep), R=(Q+1)..(P-1)}!=(P-Q-1);
15                   patlen(L); X<L.
```

Listing 3. Encoding for embedding with $\neg(\cdot)$/`csnegpat/2` negative itemsets

```
1 emb(T,P,X+1,1) :- emb(T,Q,X,0); Q<P;
2                   seq(T,P,E):pat(X+1,E); seq(T,P,_);
3                   #count{ R : seq(T,R,Ep), negpat(X,Ep), Q<R, R<P}!=0;
4                   patlen(L); X<L.

6 emb(T,P,X+1,0) :- emb(T,Q,X,0); Q<P;
7                   seq(T,P,E):pat(X+1,E); seq(T,P,_);
8                   #count{ R : seq(T,R,Ep), negpat(X,Ep), Q<R, R<P}=0;
9                   patlen(L); X<L.
```

Listing 4. Encoding for embedding with $\neg|\cdot|$/`negpat/2` negative itemsets

ν is a Boolean value that stores the information for an embedding of the positive part that these embedding satisfies or not the negative constraints.

With this definition it happens that if there is an atom of the embedding `emb(`$\boldsymbol{t}$`,`p`,`l`,`ν`)` where l is the size of the positive part of the pattern whatever the ν value, then the positive part occurs in the sequence $\boldsymbol{t}$. If ν is false, this means that there exists an embedding for the negative sequential pattern, thus $\boldsymbol{p}$ weakly occurs in $\boldsymbol{t}$. If there is no embedding such that ν is true, then it means that no embedding of the positive part is an embedding of the negative sequential pattern. Thus $\boldsymbol{p}$ strongly occurs in $\boldsymbol{t}$ if it is not the case. The following rules encode the rules of pattern covering for the two different definitions of weak and strong occurrences of a candidate pattern.

```
weakcover(T) :- patlen(L); emb(T,_,L,0).
strongcover(T) :- patlen(L); emb(T,_,L,_);
                  #count{ T : emb(T,_,L,1) }=0.
```

Then, depending on the kind of covering considered, line 16 of Listing 2 is adapted to the minimal occurrence number constraint.

The remainder of this section details the encoding for each semantics of negative itemsets.

5.1 Encoding of $\neg(\cdot)$ negative constraints

Listing 3 gives the encoding yielding the `emb/4` atoms considering patterns with $\neg(\cdot)$/`csnegpat/2` negative itemsets.

In this encoding, the Line 1 states that ϵ_1 of an embedding is a position p in the sequence $\boldsymbol{t}$ such $p_1 \subseteq t_p$. For the first pattern itemset, there are no negative constraints at previous position. Thus, ν is false (`0`).

The rule in lines 3-5 is similar to the rule in lines 10-12 of Listing 2. They yield the ϵ_{x+1} of an embedding of the positive part of the pattern where $(\epsilon_i)_{i\in[x]}$ is an embedding of $\langle p_1\ \neg q_1\ \ldots\ p_x\rangle$ s.t. there exists $y \in [x-1]$ s.t. q_y is not satisfied (ν is true). In this case, $(\epsilon_i)_{i\in[x+1]}$ is an embedding of $\langle p_1\ \neg q_1\ \ldots\ p_{x+1}\rangle$ that neither satisfies all the negative constraints. Thus, the truth value of ν is propagated.

The rules in lines 7-15 consider the case of an embedding $(\epsilon_i)_{i\in[x]}$ of the negative pattern $\langle p_1\ \neg q_1\ \ldots\ p_x\rangle$ such that all the negative constraints are satisfied. Then, there are two cases for the embedding $(\epsilon_i)_{i\in[x+1]}$. If there exists a position p in the sequence $\boldsymbol{t}$ such that $p_{x+1} \subseteq t_p$ and the negative constraint q_x is satisfied, then the atom `emb(`$\boldsymbol{t}$`,`p`,`l`,`ν`)` is yielded with ν set to false (0). That is, the rule in lines 7-10. But, if there exists a position p, $p > \epsilon_x$ in the sequence $\boldsymbol{t}$ such that $p_{x+1} \subseteq t_p$ and the negative constraint q_x is not satisfied, then the atom `emb(`$\boldsymbol{t}$`,`p`,`l`,`ν`)` is yielded with ν set to true (1). That is, the rule in lines 12-15. Note that these two rules could be factorised to avoid redundancies. For sake of readability, we prefer to present this redundant syntax.

Let us now detail how to evaluate the negative constraint q_x (*i.e.*, the set of atoms `csnegpat(`x`,`e`)`). According to Equation 2, the negative constraint q_x is satisfied iff $\forall r \in]\epsilon_i, \epsilon_{i+1}[$, $\exists e \in q_x$, s.t. $e \notin t_r$. It is equivalent to evaluate whether $|\{r \in]\epsilon_i, \epsilon_{i+1}[| \exists e \in q_x, e \notin t_r\}|$ is equal to $\epsilon_{i+1} - \epsilon_i + 1$. This latter constraint is encoded line 9 and its opposite is encoded line 14. The `#count` is an aggregate to compute the number of different `R` values satisfying conditions after the colon.

5.2 Encoding of $\neg|\cdot|$ negative constraints

The negative constraints q_x with $\neg|\cdot|$/`negpat/2` is satisfied iff $\forall r \in]\epsilon_i, \epsilon_{i+1}[$, $\forall e \in q_x$, $e \notin t_r$ (see Equation 4). This means that it is not satisfied iff $\{r \in]\epsilon_i, \epsilon_{i+1}[| e \in q_x \wedge e \in t_r\}$ is empty.

Listing 4 yields atoms of predicate `emb/4` with the same principle of Listing 3. The lines 3 or 8 evaluate $|\{r \in]\epsilon_i, \epsilon_{i+1}[| e \in q_x \wedge \in$

```
 1 negislen(X,N) :- X=1..(L-1); patlen(L); N=#count{E : cnegpat(X,E)}.

 3 emb(T,P,X+1,1) :- emb(T,Q,X,0); Q<P;
 4                   seq(T,P,E):pat(X+1,E); seq(T,P,_);
 5                   negislen(X,N);
 6                   N=#count{ Ep : seq(T,R,Ep), cnegpat(X,Ep), Q<R, R<P};
 7                   patlen(L); X<L.

 9 emb(T,P,X+1,0) :- emb(T,Q,X,0); Q<P;
10                   seq(T,P,E):pat(X+1,E); seq(T,P,_);
11                   negislen(X,N);
12                   N!=#count{ Ep : seq(T,R,Ep), cnegpat(X,Ep), Q<R, R<P};
13                   patlen(L); X<L.
```

Listing 5. Encoding for embedding with $\neg\{\cdot\}$/`cnegpat/2` negative itemsets

$t_r\}|$. It is compared to 0 to assign the truth value to ν for ϵ_{x+1} (if exists, *i.e.*, $p_{x+1} \subseteq t_{\epsilon_{x+1}}$).

5.3 Encoding of $\neg\{\cdot\}$ negative constraints

Finally, the negative constraint q_x with $\neg\{\cdot\}$/`cnegpat/2` is satisfied iff $\exists e \in q_x,\ e \notin \bigcup_{r\in]\epsilon_i,\epsilon_{i+1}[} t_r$ (see Equation 3). This means that it is not satisfied iff $\forall e \in q_x,\ e \in \bigcup_{r\in]\epsilon_i,\epsilon_{i+1}[} t_r$ or in other words that $|\{e \in q_x | e \in \bigcup_{r\in]\epsilon_i,\epsilon_{i+1}[} t_r\}| = |q_{x-1}|$.

Listing 5 yields atoms of predicate `emb/4` with the same principle of Listing 3. Line 1 evaluates the size of negative itemsets. `negislen(`x`,`n`)` states that $|q_x| = n$. Lines 6 and 12 evaluate $|\{e \in q_x | e \in \bigcup_{r\in]\epsilon_i,\epsilon_{i+1}[} t_r\}|$. It is compared to $|q_x|$ to assign the truth value to ν for ϵ_{x+1} (if exists, *i.e.*, $p_{x+1} \subseteq t_{\epsilon_{x+1}}$).

6 Experiments and results

The previous section exhibited the ease of modeling of the different semantics of negative sequential patterns in order to extract the frequent ones. It is worth noticing that only few of these semantics have properties to design algorithms that are both efficient and complete [8]. More precisely, the anti-monotonicity property holds with weak occurrences of $\neg|\cdot|$, and weaker versions of anti-monotonicity hold with the strong occurrences of $\neg|\cdot|$.

The question investigated in these experiments is whether the solver takes advantage of these properties to achieve the mining task within a lower computation time.

6.1 Experimental setting

We conducted experiments on synthetic sets of sequences generated with the generator of [2], without hidden patterns (pure random sequences).

The usage of synthetic sets of sequences allows us to control and analyze the most important characteristics of data wrt the computing time. To be more precise, we randomly generated sets of 100 sequences. Each sequence contains 10 itemsets. Each itemset is of size 2. Items are randomly generated according to a Gaussian law (some items are more frequent than others) over a vocabulary of 20 items. For each data point, we then give average computing times over five similar sets of sequences.

All the experiments have been using the *clingo* solver[5] on a desktop computer with enough memory to prevent from using cache memory and ran with a single thread. A timeout was set up to 5 minutes.

For sake of better interpretability of the results and of computation time, the negative sequential patterns contain only one kind of negated itemset ($\neg|\cdot|$, $\neg(\cdot)$ or $\neg\{\cdot\}$), the length of the positive part of a pattern is at most 3, and the size of negated itemsets is limited to 2 items.

Note that we do not compare our approach with dedicated algorithms, *e.g.* [10, 17] that are undoubtly significantly more efficient for this task. We remind that declarative pattern mining aims at achieving reasonable computation time for mid-sized set of examples. Thus, it can be used to inject some expert knowledge in the mining process to reduce the number of patterns.

6.2 Results

Figure 1 compares the number of patterns that are extracted through the different semantics of negative patterns. Note that it is very interesting to have this study as no algorithm can extract a complete set of frequent negative sequential patterns for all negation semantics.

The figures take into account only the number of patterns for runs ended before timeout (5 minutes). It explains that there are only few results for $\neg\{\cdot\}$ which is often out of the timeout.

The results illustrated in Figure 1 are consistent with theoretical results [8]. First, we notice that there are more patterns extracted with weak-occurrences than with strong-occurrences. For identical settings, there are in average 2.80 ± 2.13 times more patterns with weak-occurrences than with strong-occurrences. Second, $\neg|\cdot|$ is the strongest negative constraint after $\neg\{\cdot\}$ and finally $\neg(\cdot)$ is the weakest. As a consequence, there are more extracted patterns for $\neg(\cdot)$, than for $\neg|\cdot|$. The number of patterns for $\neg\{\cdot\}$ is in-between when available.

Figure 2 compares computing times of ASP-based negative sequential pattern mining with different semantics of negative patterns.

We can first notice that extracting negative sequential patterns with $\neg\{\cdot\}$ lacks efficiency and the mining tasks does not finish before timeout in most cases.

Then, despite the larger number of occurrences of weak-occurrences, evaluating weak occurrences is more efficient that evaluating the strong ones. It is unsurprising. Indeed, as soon as a weak-occurrence has been found, the atom `cover(`$\boldsymbol{t}$`)` is proved to be true without necessarily exploring the other embeddings of the positive part of the pattern. For strong occurrences, it is mandatory to evaluate negative patterns for every embedding of the positive part of the candidate pattern.

[5] *clingo* version 5.3.1

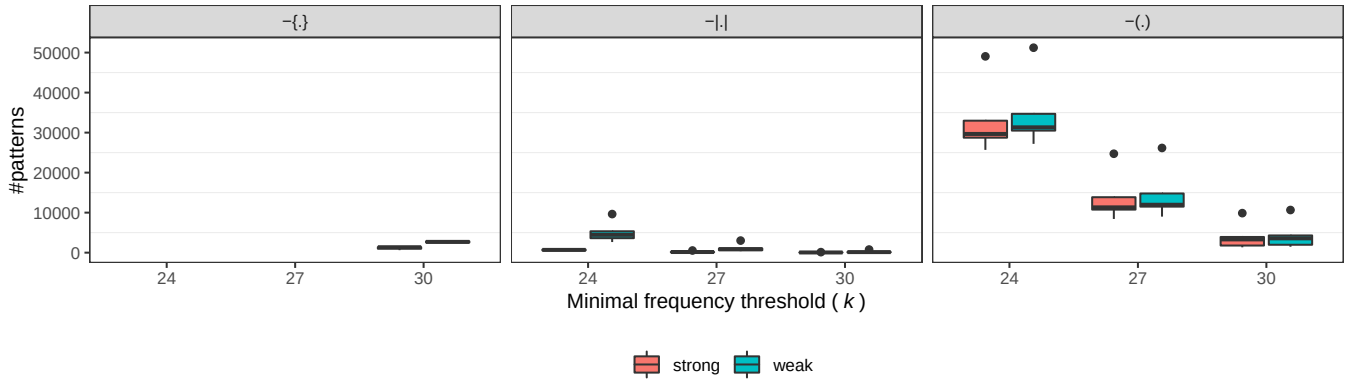

Figure 1. Boxplots of the numbers of patterns for computing all frequent negative sequential patterns wrt different thresholds (24, 27 and 30) using ASP. Each plot corresponds to one specific semantic, from left to right $\neg\{\cdot\}$, $\neg|\cdot|$ and $\neg(\cdot)$. Blue (resp. red) boxplots correspond to weak (resp. strong) occurrences).

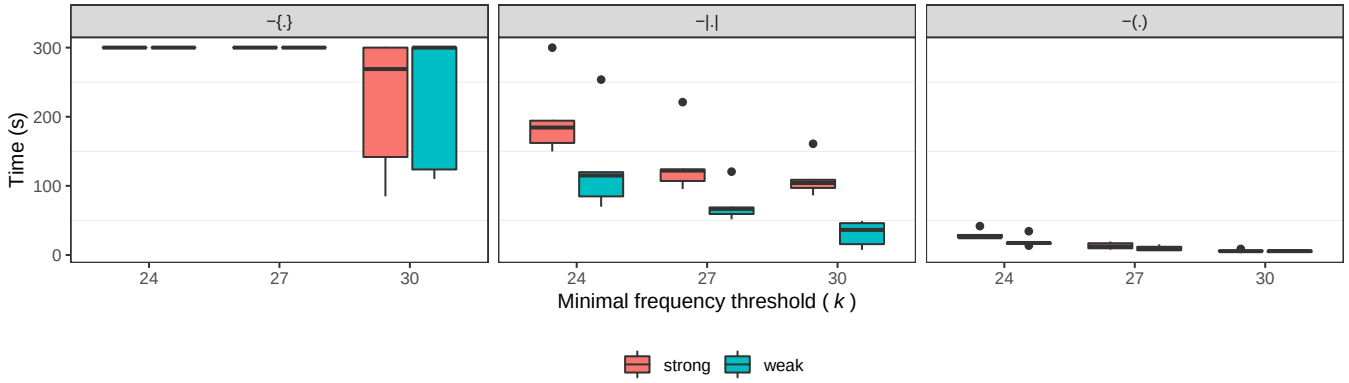

Figure 2. Boxplots of times for computing all frequent negative sequential patterns wrt different thresholds (24, 27 and 30) using ASP. Each plot corresponds to one specific semantic, from left to right $\neg\{\cdot\}$, $\neg|\cdot|$ and $\neg(\cdot)$. Blue (resp. red) boxplots correspond to weak (resp. strong) occurrences).

The surprising result of this experiment is in the comparison between $\neg(\cdot)$ and $\neg|\cdot|$. It turns out that mining negative sequential patterns with $\neg(\cdot)$ is significantly more efficient than with $\neg|\cdot|$. This result is enhanced by the fact that there are many more patterns to extract with $\neg(\cdot)$ than with $\neg|\cdot|$. So there are more patterns but more efficiently extracted. The opposite outcome was expected due to our theoretical results that exhibited anti-monotonicity for $\neg|\cdot|$, but not for $\neg(\cdot)$.

7 Conclusion

This article extends the framework of mining sequential patterns with Answer Set Programming to negative sequential patterns. We encoded the three different semantics of absent events proposed in [8]. The first interest of the approach is to evaluate the impact of the choice of one of the semantics on the set of extracted patterns. In this article, we focused on the size of this set. Deeper experiments on real datasets would be interesting to identify the differences between these sets of patterns.

Our experiments on synthetic data show that the kind of negated itemset for which the encoding is the most efficient is not the expected one. Indeed, we expected to have better efficiency with $\neg|\cdot|$. A possible explanation of this observation may be that there are also anti-monotonicity properties of $\neg(\cdot)$ for partial orders that are not usually investigated for pattern mining algorithms [10]. More investigation is required to confirm this intuition. But it encourages us to think that there are potentially better encodings for $\neg|\cdot|$. One of our perspectives is to investigate saturation programming techniques that may be suitable and efficient to encode weak-occurrences of negative sequential patterns.

In addition, these encodings may be used to characterize what patterns can still be mined by incomplete mining heuristics. Wang et *al.* [17] reviewed such algorithms but they do not characterise the extracted patterns among the complete set of patterns. Comparing their outputs with the outputs of our encodings makes it possible to investigate this question.

REFERENCES

[1] Emmanuel Coquery, Said Jabbour, Lakhdar Sais, and Yakoub Salhi, 'A SAT-based approach for discovering frequent, closed and maximal patterns in a sequence', in *Proceedings of the European Conference on Artificial Intelligence*, volume 242, pp. 258–263, (2012).

[2] Martin Gebser, Thomas Guyet, René Quiniou, Javier Romero, and Torsten Schaub, 'Knowledge-based sequence mining with ASP', in *Proceedings of International Join Conference on Artificial Intelligence*, pp. 1497–1504, (2016).

[3] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub, '*Clingo* = ASP + Control: Preliminary report', in *Technical Communications of the Thirtieth International Conference on Logic Programming*, (2014).

[4] Mickael Gelfond and Vladimir Lifschitz, 'Classical negation in logic

programs and disjunctive databases', *New Generation Computing*, **9**, 365–385, (1991).
[5] Tias Guns, Anton Dries, Siegfried Nijssen, Guido Tack, and Luc De Raedt, 'Miningzinc: A declarative framework for constraint-based mining', *Artificial Intelligence*, **244**, 6–29, (2017).
[6] Tias Guns, Sergey Paramonov, and Benjamin Negrevergne, 'On declarative modeling of structured pattern mining', in *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, (2016).
[7] Thomas Guyet, *Enhancing sequential pattern mining with time and reasoning*, Habilitation à diriger des recherches, Université de Rennes 1, 2020.
[8] Thomas Guyet and Philippe Besnard, 'Semantics of negative sequential patterns', in *Proceedings of European Conference on Artificial Intelligence (ECAI)*, p. to appear, Santiago de Compostela, Spain, (2020).
[9] Thomas Guyet, Yves Moinard, René Quiniou, and Torsten Schaub, 'Efficiency analysis of ASP encodings for sequential pattern mining tasks', in *Advances in Knowledge Discovery and Management*, eds., Bruno Pinaud, Fabrice Guillet, Bruno Cremilleux, and Cyril de Runz, volume 7, 41–81, Springer, (2017).
[10] Thomas Guyet and René Quiniou, 'NegPSpan: efficient extraction of negative sequential patterns with embedding constraints', *Data Mining and Knowledge Discovery*, **34**(2), 563–609, (2020).
[11] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan, 'Frequent pattern mining: current status and future directions', *Data Mining and Knowledge Discovery*, **15**(1), 55–86, (2007).
[12] Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi, 'Decomposition based SAT encodings for itemset mining problems', in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 662–674. Springer, (2015).
[13] Matti Järvisalo, 'Itemset mining as a challenge application for answer set enumeration', in *Proceedings of the conference on Logic Programming and Nonmonotonic Reasoning*, pp. 304–310, (2011).
[14] Carl H. Mooney and John F. Roddick, 'Sequential pattern mining – approaches and algorithms', *ACM Computing Survey*, **45**(2), 1–39, (2013).
[15] Benjamin Negrevergne and Tias Guns, 'Constraint-based sequence mining using constraint programming', in *Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR*, pp. 288–305, (2015).
[16] Patrik Simons, Ilkka Niemelá, and Timo Soininen, 'Extending and implementing the stable model semantics', *Journal of Artificial Intelligence*, **138**(1-2), 181–234, (2002).
[17] Wei Wang and Longbing Cao, 'Negative sequence analysis: A review', *ACM Computing Survey*, **52**(2), 32:1–32:39, (2019).