Knowledge-Based Sequence Mining with ASP

Martin Gebser,[3] Thomas Guyet,[1] René Quiniou,[2] Javier Romero,[3] Torsten Schaub[2,3]
[1]AGROCAMPUS-OUEST/IRISA, France
[2]Inria – Centre de Rennes Bretagne Atlantique, France
[3]University of Potsdam, Germany

Abstract

We introduce a framework for knowledge-based sequence mining, based on Answer Set Programming (ASP). We begin by modeling the basic task and refine it in the sequel in several ways. First, we show how easily condensed patterns can be extracted by modular extensions of the basic approach. Second, we illustrate how ASP's preference handling capacities can be exploited for mining patterns of interest. In doing so, we demonstrate the ease of incorporating knowledge into the ASP-based mining process. To assess the trade-off in effectiveness, we provide an empirical study comparing our approach with a related sequence mining mechanism.

1 Introduction

Sequential pattern mining is about identifying frequent subsequences in sequence databases [Shen *et al.*, 2014]. Modern miners exhibit a compelling effectiveness and allow for high-throughput extraction from big data sources. But this often brings about a profusion of patterns, which becomes a problem to analysts, who, although no longer swamped by data, are now swamped by patterns. The emerging challenge is thus to trade quantity for quality and to extract fewer patterns of greater relevance to the analyst. To this end, the analyst must be empowered to express her criteria of interest. However, this turns out to be rather cumbersome with existing mining technology. First, it is difficult to customize highly optimized miners, and second, modifications need knowledge about their inner workings. Unlike this, our goal is to conceive an approach to sequence mining that allows for an easy integration of expert knowledge, without presupposing deep insights into the mining mechanism. We address this by means of Answer Set Programming (ASP; [Lifschitz, 2008]), a prime tool for knowledge representation and reasoning. Technically, the idea is to represent a sequence mining task as a logic program such that each of its answer sets comprises a pattern of interest, similar to the seminal work on ASP-based itemset mining in [Järvisalo, 2011]. Refining the considered patterns then corresponds to the addition of constraints or preferences, respectively, to an ASP encoding.

Even without the deluge of patterns, mining interesting patterns by incorporating expert knowledge is a long standing issue. For one, [Muggleton and De Raedt, 1994] used Inductive Logic Programming with expert knowledge to improve the learning process. The data mining community addresses the issue algorithmically by defining search spaces corresponding to expert expectations, while being efficiently traversable. A common approach is to use constraints on patterns (eg. regular expressions) or on their occurrences (eg. max-gaps, max-duration). Also, interestingness measures have been defined [De Bie, 2011]. In particular, skypatterns [Ugarte *et al.*,] were introduced to manage complex preferences on patterns. Other approaches use data [Padmanabhan and Tuzhilin, 1998] or process [Flouvat *et al.*, 2014] models to integrate expert knowledge into the mining process, for instance, in order to extract unexpected patterns. Recent works on pattern mining with Constraint Programming (CP) [Guns *et al.*, 2011; Coquery *et al.*, 2012; Negrevergne and Guns, 2015; Kemmar *et al.*, 2015] step up the ladder by providing an effective and unified mining framework.[1] However, expressing knowledge in terms of such constraints still remains difficult for domain experts.

To the best of our knowledge, no existing framework addresses all aforementioned dimensions at once. In most cases, the addition of knowledge to the mining process remains a specific and tedious task that often results in dedicated algorithms. We address this shortcoming by proposing ASP as a uniform framework for intensive knowledge-based sequence mining. To this end, we develop concise yet versatile encodings of frequent (closed or maximal) sequential pattern mining as well as encodings for managing complex preferences on patterns. While the primary objective of our approach is not to tackle vast amounts of data, in the first place, we aim at a unified framework in which domain knowledge can easily be incorporated to process mid-size databases more effectively. In this way, we provide means enabling experts to extract small sets of patterns that they are really interested in.

2 Background

Our terminology on sequence mining follows the one in [Negrevergne and Guns, 2015]. A *database* $\mathcal{D}$ is a multi-set of sequences over some given set Σ of items.[2] A sequence

[1]We return to these approaches in Section 6 in more detail.

[2]The generalization to sequences of itemsets, ie., sequences of sets of items, is straightforward and not considered here.

$\boldsymbol{s} = \langle s_i \rangle_{1 \le i \le m}$ with $s_i \in \Sigma$ is *included* in a sequence $\boldsymbol{t} = \langle t_j \rangle_{1 \le j \le n}$ with $m \le n$, written $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$, if $s_i = t_{e_i}$ for $1 \le i \le m$ and an increasing sequence $\langle e_i \rangle_{1 \le i \le m}$ of positive integers $1 \le e_i \le n$, called an *embedding* of $\boldsymbol{s}$ in $\boldsymbol{t}$. For example, we have $\langle a, c \rangle \sqsubseteq \langle a, b, c \rangle$ relative to embedding $\langle 1, 3 \rangle$. We write $\boldsymbol{s} \sqsubseteq_b \boldsymbol{t}$, if $\boldsymbol{s}$ is a *prefix* of $\boldsymbol{t}$. Given a database $\mathcal{D}$, the *cover* of a sequence $\boldsymbol{s}$ is the set of sequences in $\mathcal{D}$ that include $\boldsymbol{s}$: $cover(\boldsymbol{s}, \mathcal{D}) = \{\boldsymbol{t} \in \mathcal{D} \mid \boldsymbol{s} \sqsubseteq \boldsymbol{t}\}$. The number of sequences in $\mathcal{D}$ including $\boldsymbol{s}$ is called its *support*, that is, $support(\boldsymbol{s}, \mathcal{D}) = |cover(\boldsymbol{s}, \mathcal{D})|$. For an integer k, *frequent sequence mining* is about discovering all sequences $\boldsymbol{s}$ such that $support(\boldsymbol{s}, \mathcal{D}) \ge k$. We often call $\boldsymbol{s}$ a (sequential) *pattern*, and k is also referred to as the (minimum) *frequency threshold*. Condensed representations of frequent patterns are defined as follows. A pattern $\boldsymbol{s}$ is *maximal*, if there is no other pattern $\boldsymbol{t}$ such that $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$ and $support(\boldsymbol{t}, \mathcal{D}) \ge k$; $\boldsymbol{s}$ is *closed*, if there is no other pattern $\boldsymbol{t}$ such that $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$ and $support(\boldsymbol{s}, \mathcal{D}) = support(\boldsymbol{t}, \mathcal{D})$. Analogous properties are obtained for the prefix relation $\sqsubseteq_b$. Additional constraints over patterns (item constraints and regular expressions) or over embeddings (maximum gaps and maximal span) are discussed in [Negrevergne and Guns, 2015] but beyond the focus of this paper.

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

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

where each $\mathtt{a_i}$ is a propositional atom for $0 \le \mathtt{i} \le \mathtt{n}$ and `not` stands for *default negation*. If $\mathtt{n} = 0$, rule (1) is called a *fact*. If $\mathtt{a_0}$ is omitted, (1) represents an integrity constraint. Semantically, a logic program induces a collection of so-called *answer sets*, which are distinguished models of the program determined by answer sets semantics; see [Gelfond and Lifschitz, 1991] 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* [Simons *et al.*, 2002]. The former are of the form $\mathtt{a : b_1, \ldots, b_m}$, the latter can be written as $\mathtt{s}\ \{\mathtt{c_1, \ldots, c_n}\}\ \mathtt{t}$, 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 $\mathtt{a(X) : b(X)}$ in a rule's antecedent expands to the conjunction of all instances of $\mathtt{a(X)}$ for which the corresponding instance of $\mathtt{b(X)}$ holds. Similarly, $2\ \{\mathtt{a(X) : b(X)}\}\ 4$ holds whenever between two and four instances of $\mathtt{a(X)}$ (subject to $\mathtt{b(X)}$) are true. Specifically, we rely in the sequel on the input language of the ASP system *clingo* [Gebser *et al.*, 2014].

Moreover, we draw upon *asprin* [Brewka *et al.*, 2015], an ASP-based system for expressing combinations of qualitative and quantitative preferences among answer sets. A *preference relation* is defined via a declaration of the form

$$\mathtt{\#preference(p,t)}\{\mathtt{c_1, \ldots, c_n}\}.$$

where `p` and `t` are the name and type of the preference relation, respectively, and each $\mathtt{c_j}$ is a conditional literal.[3] The directive `#optimize(p)` instructs *asprin* to search for `p`-optimal answer sets, being maximal wrt the strict preference relation associated with `p`. While *asprin* already comes with a library of predefined primitives and aggregate preference types, like `subset` or `pareto`, respectively, it also allows for adding customized preferences. To this end, users provide rules defining an atom `better(p)` that indicates whether an answer set X is preferred to another Y wrt the preference relation associated with `p`. Both sets X and Y are reified by *asprin* via unary predicates `holds` and `holds'`,[4] whose instances are drawn upon in the definition of `better(p)`.

[3]See [Brewka *et al.*, 2015] for more general preference elements.

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

3 ASP-based Sequence Mining

3.1 Fact format

We represent a database $\mathcal{D}$ in terms of facts `seq(`$\boldsymbol{t}$`,`p`,`e`)`, saying that item e occurs at position p in a sequence $\boldsymbol{t}$, denoted by a common term in the facts providing its elements. For instance, Listing 1 specifies a database of seven sequences:

Term	1	2	3	4	5	6	7
Seq.	$\langle d, a, b, c \rangle$	$\langle a, c, b, c \rangle$	$\langle a, b, c \rangle$	$\langle a, b, c \rangle$	$\langle a, c \rangle$	$\langle b \rangle$	$\langle c \rangle$

Note that different terms can refer to the same elements for distinguishing multiple occurrences of a sequence, such as the identifiers `3` and `4` for $\langle a, b, c \rangle$. Considering the frequency threshold $k = 2$, one can check that $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle a, b \rangle$, $\langle a, c \rangle$, $\langle b, c \rangle$, and $\langle a, b, c \rangle$ are the frequent patterns of the database. Since all of them are included in $\langle a, b, c \rangle$, the latter is the unique maximal pattern. Moreover, the patterns $\langle b \rangle$, $\langle c \rangle$, $\langle a, c \rangle$, and $\langle a, b, c \rangle$ are closed, while $\langle a \rangle$, $\langle a, b \rangle$, and $\langle b, c \rangle$ are not because they have the same support as respective extensions to $\langle a, c \rangle$ or $\langle a, b, c \rangle$.

3.2 Mining frequent patterns: basic encoding

The encoding principle for frequent sequence mining follows the one of [Järvisalo, 2011], that is, each answer set comprises a single pattern of interest. (This allows for a compact representation of exponentially many candidate patterns.) 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. It relies on two parameters: *max* determines a maximum length for patterns of interest, and *k* specifies the frequency threshold. While longest sequences in a given database yield a natural maximum length for patterns that

[4]That is, `holds(a)` (or `holds'(a)`) is true iff $\mathtt{a} \in X$ (or $\mathtt{a} \in Y$).

Table 1: Atoms representing a pattern $\boldsymbol{s} = \langle s_i \rangle_{1 \le i \le m}$, where $1 \le m \le \mathit{max}$, and a given multiset $\mathcal{D}$ of sequences $\boldsymbol{t} = \langle t_j \rangle_{1 \le j \le n}$

Atom	Meaning
`slot(`x`)`	$1 \le x \le \mathit{max}$ may refer to the position $m{+}1 - x$ of an item s_{m+1-x} in $\boldsymbol{s}$, which exists whenever $x \le m$
`pos(`x`)`	$1 \le x \le m$ refers to the position $m{+}1 - x$ of an item s_{m+1-x} in $\boldsymbol{s}$
`pat(`x`,`e`)`	$s_{m+1-x} = e$ is the item at position $m{+}1 - x$ in $\boldsymbol{s}$, where $1 \le x \le m$
`item(`e`)`	item e belongs to some $\boldsymbol{t} \in \mathcal{D}$
`cover(`$\boldsymbol{t}$`)`	$\langle s_i \rangle_{1 \le i \le m} \sqsubseteq \langle t_j \rangle_{1 \le j \le n}$, that is, $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$
`inc(`$\boldsymbol{t}$`,`p`,`x`)`	$\langle s_i \rangle_{1 \le i \le m-x} \sqsubseteq \langle t_j \rangle_{1 \le j < p}$, where $1 \le p \le n+1$ and $0 \le x \le \mathit{max}$
`tail(`$\boldsymbol{t}$`,`p`,`x`)`	$\langle s_i \rangle_{m+1-x \le i \le m} \not\sqsubseteq \langle t_j \rangle_{p < j \le n}$ (or $m < x$), where $1 \le p \le n$ and $1 \le x \le \mathit{max}$
`ins(`$\boldsymbol{t}$`,`x`,`e`)`	$t_p = e$, $\langle s_i \rangle_{1 \le i \le m-x} \sqsubseteq \langle t_j \rangle_{1 \le j < p}$, and $\langle s_i \rangle_{m+1-x \le i \le m} \sqsubseteq \langle t_j \rangle_{p < j \le n}$ (or $\boldsymbol{s} \not\sqsubseteq \boldsymbol{t}$ in case of closed patterns), where $1 \le p \le n$ and $0 \le x \le m$

```
 1 slot(1..max).
 2 item(E) :- seq(T,P,E).
 3 { pat(X,E) : item(E) } 1 :- slot(X).

 5 pos(X) :- pat(X,E).
 6 :- not pos(1).
 7 :- pos(X), 2 < X, not pos(X-1).

 9 inc(T,1,X)      :- seq(T,1,E), slot(X),
10                    not pos(X+1).
11 inc(T,P+1,X)    :- seq(T,P,E), inc(T,P,X).
12 inc(T,P+1,X-1) :- seq(T,P,E), inc(T,P,X),
13                    pat(X,E).

15 cover(T) :- inc(T,P,0), { seq(T,P,E) } 0.
16 :- not k { cover(T) }.
```

Listing 2: Basic encoding of frequent sequence mining

may be frequent, picking smaller values for *max* can significantly reduce the size of ground programs, so that the parameter allows for tuning towards efficiency. An answer set then represents a frequent pattern $\boldsymbol{s} = \langle s_i \rangle_{1 \le i \le m}$ such that $1 \le m \le \mathit{max}$ by atoms `pat(`m`,`s_1`)`, …, `pat(1,`s_m`)`. That is, the first argument expresses the positions of items in decreasing order, where m can vary, while `1` always indicates the last item in a pattern. (Our encoding utilizes this invariant to detect sequences covering the entire pattern.) For instance, the atoms `pat(3,`a`)`, `pat(2,`b`)`, and `pat(1,`c`)` stand for the (frequent) pattern $\langle a, b, c \rangle$ of the database given in Listing 1.

A complete specification of atoms used in our ASP encodings along with their meanings is given in Table 1. In more detail, the rules in Line 1 and 2 of Listing 2 provide integers in the interval denoted by `1..`*max* as candidate positions and items occurring in a given database $\mathcal{D}$. The (choice) rule in Line 3 allows for picking at most one item per position to build a pattern, and occupied positions are extracted in Line 5. Given this, the integrity constraints in Line 6 and 7 make sure that the last position `1` is non-empty and that other positions are consecutive up to the pattern length m. As a result, the rules from Line 1 to 7 establish a unique representation for a candidate pattern whose frequency remains to be checked.

To this end, the rules from Line 9 to 13 traverse each sequence $\boldsymbol{t} = \langle t_j \rangle_{1 \le j \le n}$ in $\mathcal{D}$ to derive atoms of the form `inc(`$\boldsymbol{t}$`,`p`,`x`)`. As noted in Table 1, for $1 \le p \le n+1$ and $0 \le x \le \mathit{max}$, such an atom expresses that $\langle s_i \rangle_{1 \le i \le m-x} \sqsubseteq \langle t_j \rangle_{1 \le j < p}$. (In other words, some embedding $\langle e_i \rangle_{1 \le i \le m-x}$ of $\langle s_i \rangle_{1 \le i \le m-x}$ in $\boldsymbol{t}$ such that $1 \le e_i < p$ exists, while respective integers e_i are left implicit.) Of particular interest is the condition $\langle s_i \rangle_{1 \le i \le m} \sqsubseteq \langle t_j \rangle_{1 \le j < n+1}$, that is, $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$, obtained for $p = n+1$ and $x = 0$. In fact, corresponding atoms `inc(`$\boldsymbol{t}$`,`$n+1$`,0)` are in Line 15 distinguished via a cardinality constraint of the form "`{ seq(`$\boldsymbol{t}$`,`$n+1$`,`e`) } 0`", indicating the absence of any item e at position $n+1$, and further inspected to determine the cover of a candidate pattern $\boldsymbol{s}$. The frequency of $\boldsymbol{s}$ is then checked in Line 16, where the cardinality constraint "k `{ cover(`$\boldsymbol{t}$`) }`" over atoms `cover(`$\boldsymbol{t}$`)`, signaling $\boldsymbol{s} \sqsubseteq \boldsymbol{t}$, expresses that at least the frequency threshold k many sequences $\boldsymbol{t}$ must include $\boldsymbol{s}$.

For the database in Listing 1, the pattern $\langle a, b, c \rangle$ leads to atoms `inc(2,1,3..`*max*`)`, `inc(2,2,2..`*max*`)`, `inc(2,3,2..`*max*`)`, `inc(2,4,1..`*max*`)`, and `inc(2,5,0..`*max*`)`, among which `inc(2,5,0)` in turn yields `cover(2)` for the sequence $\langle a, c, b, c \rangle$ denoted by `2`.[5] In a similar fashion, `cover(1)`, `cover(3)`, and `cover(4)` are concluded, so that $\langle a, b, c \rangle$ turns out to be frequent, given the threshold $k = 2$.

3.3 Mining condensed frequent patterns

The notion of maximal (or closed) patterns suppresses redundant outcomes by requiring that a frequent pattern of interest is not strictly included in another frequent pattern (or another pattern with same support). In the following, we show how such additional requirements can be addressed via extensions of the basic encoding in Listing 2.

The main idea of encoding parts for selecting either maximal or closed frequent patterns is to investigate the effect of adding any single item to a candidate pattern $\boldsymbol{s} = \langle s_i \rangle_{1 \le i \le m}$. To this end, we can reuse the information $\langle s_i \rangle_{1 \le i \le m-x} \sqsubseteq \langle t_j \rangle_{1 \le j < p}$ expressed by atoms of the form `inc(`$\boldsymbol{t}$`,`p`,`x`)` for each sequence $\boldsymbol{t} = \langle t_j \rangle_{1 \le j \le n}$ in $\mathcal{D}$. In fact, such an atom tells us that the extended pattern $\langle s_1, \ldots, s_{m-x}, t_p, s_{m+1-x}, \ldots, s_m \rangle$ is included in $\boldsymbol{t}$, provided that $\langle s_i \rangle_{m+1-x \le i \le m} \sqsubseteq \langle t_j \rangle_{p < j \le n}$. The complement of the latter condition (which also applies in case $m < x$ yields void items in suffix $\langle s_{m+1-x}, \ldots, s_m \rangle$) is expressed by atoms of the form `tail(`$\boldsymbol{t}$`,`p`,`x`)`, defined by the rules from Line 17 to 22 in Listing 3. All prerequisites for inserting an item $t_p = e$ in-between the $(m-x)$-th (or at the front, if $x = m$) and

[5]Atoms `inc(`$\boldsymbol{t}$`,`p`,`x`)` such that $p + x > n+1$, eg. `inc(2,5,1..`*max*`)`, can be omitted, but are kept here for ease of presentation.

```
17 tail(T,P,X)    :- seq(T,P,E), slot(X),
18                   { seq(T,P+1,F) } 0.
19 tail(T,P-1,X) :- tail(T,P,X-1), 1 < P,
20                   slot(X).
21 tail(T,P-1,X) :- tail(T,P,X), 1 < P,
22                   seq(T,P,E), not pat(X,E).

24 ins(T,X,E) :- seq(T,P,E), inc(T,P,X),
25               not tail(T,P,X).
26 :- item(E), X = 0..max, k { ins(T,X,E) }.
```

Listing 3: Encoding part for selecting maximal patterns

the $(m{+}1-x)$-th (or at the end, if $x=0$) item into an extended pattern included in $\boldsymbol{t}$ are pulled together by the rule in Line 24 and 25, providing atoms `ins(`$\boldsymbol{t},x,e$`)`. Finally, the integrity constraint in Line 26 denies candidate patterns to which an item can be added without falling below the frequency threshold, so that only maximal patterns remain.

For the database in Listing 1, $\langle a,b,c\rangle$ is the single maximal pattern. Potential additions of items from the (longer) sequences $\langle d,a,b,c\rangle$ and $\langle a,c,b,c\rangle$, denoted by `1` and `2`, are indicated by the atoms `ins(1,3,`d`)` and `ins(2,2,`c`)`. That is, d can be inserted before a or c after a to obtain an extended pattern including $\langle d,a,b,c\rangle$ or $\langle a,c,b,c\rangle$, respectively. However, in both cases, the support drops to 1 and thus below the threshold $k=2$, which shows that $\langle a,b,c\rangle$ is maximal.

Listing 4 provides a modification of the encoding part from Listing 3 for selecting closed instead of maximal patterns. The difference is that the redundancy of a pattern is witnessed by an extended pattern with same support. Concerning sequences $\boldsymbol{t}$ that do not include a candidate pattern at hand, the additional rule in Line 26 and 27 yields `ins(`$\boldsymbol{t},x,e$`)` for any item e at any position x. Hence, the integrity constraint in Line 28 and 29, which replaces its counterpart from Listing 3, denies any candidate pattern such that all sequences including it agree on the insertion of some item.

For instance, the frequent pattern $\langle b,c\rangle$ of the example database in Listing 1 is denied because atoms `ins(`$\boldsymbol{t}$`,2,`a`)` for $\boldsymbol{t}\in\{$`1,2,3,4,5,6,7`$\}$ indicate that the sequences including $\langle b,c\rangle$, denoted by `1` to `4`, comply with the insertion of a before b, while the remaining three sequences do not include $\langle b,c\rangle$, so that they tolerate arbitrary additions of items. Similarly, the frequent patterns $\langle a\rangle$ and $\langle a,b\rangle$ are discarded, thus leaving the closed patterns $\langle b\rangle$, $\langle c\rangle$, $\langle a,c\rangle$, and $\langle a,b,c\rangle$.

Relaxed notions of maximal or closed frequent patterns based on the prefix relation $\sqsubseteq_b$ are easily obtained by dropping the rules for `tail(`$\boldsymbol{t},p,x$`)` from Line 17 to 22 in Listing 3 and substituting variable `X` with `0` in the remaining parts of Listing 3 and 4. This restricts the consideration of extended patterns to the addition of some item at the end. For instance, the frequent pattern $\langle b,c\rangle$ of the example database in Listing 1, which is neither maximal nor closed, becomes infrequent when extended at the end, so that it remains as maximal and closed under the relaxed notions. On the other hand, the closed pattern $\langle b\rangle$ is not maximal, even under the relaxation, because adding c yields the extended frequent pattern $\langle b,c\rangle$. While the four notions of maximal or closed frequent patterns are distinct, either of them can be imposed by augmenting the

```
26 ins(T,X,E) :- item(E), X = 0..max,
27               seq(T,1,F), not cover(T).
28 :- item(E), X = 0..max,
29    ins(T,X,E) : seq(T,1,F).
```

Listing 4: Modifications for selecting closed patterns

basic encoding in Listing 2 with respective rules. In the next section, we further show how preferences can be utilized to find best patterns among those satisfying hard requirements.

4 Preference-based Mining

We have demonstrated above how easily a basic encoding of frequent sequence mining can be extended in order to refine the patterns of interest. This can be taken one step further by exploiting ASP's preference handling capacities to extract preferred patterns [Cho *et al.*, 2005]. The idea is to map a preference on patterns to one among answer sets (cf. Section 2), which is accomplished in two steps with the *asprin* system. At first, we have to provide implementations of pattern-specific preference types that are plugged into *asprin*. Once done, the defined preference types can be combined freely with those available in *asprin*'s library to form complex preferences among the patterns given by answer sets.

Let us illustrate our approach by modeling the extraction of skypatterns [Ugarte *et al.*,], which follow the Pareto principle in combining several quality measures. To illustrate skypatterns, we consider the exemplary pattern-oriented preference types *area* and *aconf*, measuring the product of a pattern's support and length or the ratio of support and the highest frequency of its contained items, respectively. More precisely, given a database $\mathcal{D}$ and a (frequent) pattern $\boldsymbol{s}=\langle s_i\rangle_{1\leq i\leq m}$, the associated *area* is $m\times \mathit{support}(\boldsymbol{s},\mathcal{D})$, and the *aconf* measure is determined by

$$\frac{\mathit{support}(\boldsymbol{s},\mathcal{D})}{\max_{1\leq i\leq m}|\{\boldsymbol{t}\in\mathcal{D}\mid\langle s_i\rangle\sqsubseteq\boldsymbol{t}\}|}\,, \tag{2}$$

where greater values are preferred in both cases. We further consider *Pareto efficiency* to aggregate the two measures.[6]

While the *area* preference can be directly mapped to *asprin*'s built-in `cardinality` type (see below), a dedicated implementation of the *aconf* measure is given in Listing 5. To begin with, the auxiliary rules from Line 1 to 3 provide atoms of the form `cont(`e,f`)` and `freq(`f`)`. The former express that f sequences in $\mathcal{D}$ contain the item e, and the latter indicate that some item belongs to f or more sequences. Moreover, the rules in Line 5 and 6 derive atoms `hasfreq(1..`f`)` for items e in a candidate pattern, so that the number of such atoms corresponds to the highest frequency.

As described in Section 2, it remains to define an atom `better(p)` to indicate that an answer set X, reified in terms of `holds(a)` atoms, is preferred over (a previous) one, Y, given by `holds'(a)`, according to a preference `p` of type *aconf*. This is accomplished by the rule from Line 8 to 12, whose precondition of the form `preference(p,aconf)`

[6]That is, a pattern is preferred over another one if it is at least as good as the other in all criteria, and strictly better in some criterion.

```
cont(E,F)   :- item(E),
               F= #count{ T : seq(T,P,E) }.
freq(1..M) :- M = #max{ F : cont(E,F) }.

hasfreq(F)   :- cont(E,F), pat(X,E).
hasfreq(F-1) :- hasfreq(F), 1 < F.

better(P) :- preference(P,aconf),
   #sum{ 1,F,T : holds'(hasfreq(F)),
                 holds (cover(T));
        -1,F,T : holds (hasfreq(F)),
                 holds'(cover(T)) } > 0.
```

Listing 5: Preference type implementation

checks the type of preference p. The other literal is a `#sum` aggregate that casts ratios (2) to corresponding products via multiplication of the denominators. That is, each sequence $\boldsymbol{t}$ in $\mathcal{D}$ including the pattern comprised in answer set X contributes f' times the addend `1`, where f' is the highest frequency of items in the pattern given by Y. In turn, each sequence $\boldsymbol{t}$ including the pattern in Y amounts to f instances of the addend `-1` when f is the highest frequency for X. As a consequence, the `#sum` aggregate evaluates to a positive integer iff the ratio (2) obtained with the pattern in X is greater than the one for Y, which is then signaled by `better(p)`. With the rules in Listing 5 at hand, the preference type denoted by `aconf` is ready for use in preference declarations.[7]

Listing 6 shows an *asprin* declaration of the preference relations specified above. In Line 1 and 2, the *area* preference is mapped to the built-in `cardinality` type, used to count conjunctions (`&`) of item positions in a pattern and sequences including it. The preference denoted by `p1` thus aims at maximizing $m \times \mathit{support}(\boldsymbol{s}, \mathcal{D})$. The second preference `p2` in Line 3 and 4 refers to the `aconf` type, whose implementation as above addresses the maximization of the ratio (2). In Line 5, both preference relations, referenced via dedicated atoms `name(p1)` and `name(p2)`, are aggregated by means of the built-in preference type `pareto`. This yields the nested preference `p3`, made subject to optimization by *asprin* in Line 7. For the database in Listing 1, answer sets comprising the (frequent) patterns $\langle c\rangle$, $\langle a, c\rangle$, and $\langle a, b, c\rangle$ turn out to be optimal in terms of Pareto efficiency. Their associated *area* and *aconf* measures are 6, 10, and 12 or 1, 5/6, and 2/3, respectively. While neither of the three patterns is preferred over another, they dominate the remaining patterns $\langle a\rangle$, $\langle b\rangle$, $\langle a, b\rangle$, and $\langle b, c\rangle$. Of course, other customized preference types and different combinations thereof can be conceived as well, eg. using built-in types like `lexico`, `and`, and `or` (with their literal meanings).

5 Experiments

Given its exhibited ease of modeling, it is now interesting to examine the computational behavior of our ASP-based approach. To this end, we conducted experiments on simulated databases. First, we present time efficiency results comparing our approach with the CP-based one of CPSM [Negrevergne and Guns, 2015]. Then, we illustrate the effectiveness of preference handling to reduce the size of output pattern sets.

```
1 #preference(p1,more(cardinality)){
2     pos(X) & cover(T): slot(X), seq(T,1,E) }.
3 #preference(p2,aconf){ hasfreq(F) : freq(F);
4                        cover(T) :seq(T,1,E) }.
5 #preference(p3,pareto){ name(p1); name(p2) }.

7 #optimize(p3).
```

Listing 6: Preference declaration

5.1 Computing time

The usage of simulated databases allows us to control and analyze the most important characteristics of data wrt the resulting time and memory demands. To be more precise, we generated databases[8] using a "retro-engineering" process: 1) a set of random patterns is generated, 2) occurrences of patterns are assigned to 10% of 500 database sequences, and 3) each sequence of items is randomly generated according to the patterns it must contain and a mean length. In our experiments, we vary the mean length from 10 to 40, and contained items are randomly generated according to a Gaussian law (some items are more frequent than others) over a vocabulary of 50 items. For each data point, we then give average computing times over six similar databases.

Figure 1 compares computing times of ASP-based sequence mining (using the ASP system *clingo*) and CPSM (based on the CP solver *gecode*). For fairness, we ran a CPSM version using constraints similar to those in our encodings.[9] The timeout was set to 20 minutes. We do not compare our approach with dedicated algorithms, eg. [Zaki, 2001], which are known to be more efficient than declarative mining approaches (see [Negrevergne and Guns, 2015]).

The results show that the computing times with ASP are comparable to CPSM for short sequences. When sequences become larger (mean length 40), the efficiency of our encoding somewhat decreases. This is due to our purely declarative approach. In fact, the check for embeddings is a heavy task for the solver, and the larger the sequences, the more complex it becomes (also in view of an increasing ground instantiation size). Unlike this, CPSM uses dedicated propagators for space-efficient embedding computation.

Although it is beyond the focus of this paper, we mention that computing times can be greatly improved by using maxgap constraints on embeddings, a popular means of analysis since distant events in a sequence often have less significance.

5.2 Pattern set reduction

This experiment aims at comparing different pattern mining tasks: skypatterns (with *area* and *aconf* measures) as well as

[7] For brevity, Listing 5 does not show an analogous rule replacing "`> 0`" by "`>= 0`" for defining `bettereq(p)`, used additionally for aggregation via Pareto efficiency.

[8] The databases used in our experiments are available at https://sites.google.com/site/aspseqmining.

[9] We are grateful to the developers of CPSM for this suggestion.

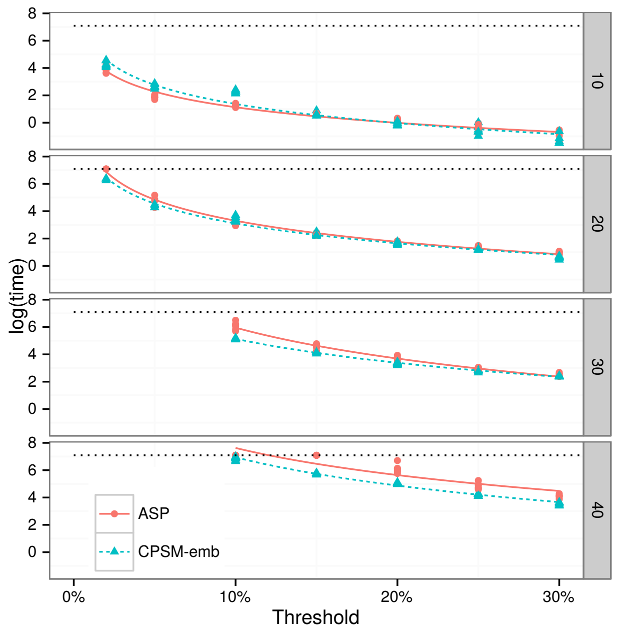

Figure 1: Mean times for computing all frequent patterns wrt different thresholds and mean sequence lengths (10, 20, 30, and 40), using ASP (solid) and CPSM (dashed). Dashed horizontal lines indicate the timeout limit (1200 seconds).

(BW/$\sqsubseteq_b$) closed and maximal frequent patterns. The objectives are to evaluate the effectiveness of our encodings on real databases and to illustrate how focusing on skypatterns drastically reduces the number of extracted patterns. We ran our respective encodings on two classical databases of the UCI collection [Lichman, 2013]: `jmlr` (a natural language processing database; each transaction is an abstract of a paper from the Journal of Machine Learning Research) and `Unix user` (each transaction is a series of shell commands executed by a user during one session).

Figure 2 displays numbers of frequent patterns obtained wrt different thresholds. For the basic as well as condensed frequent pattern mining tasks, we see that the lower the threshold, the greater the number of patterns. Despite the exponential growth (or decrease, respectively) of candidate pattern sets, the number of skypatterns remains very low (at most five). In addition, we observed that the times for computing skypatterns are comparable to those for computing all frequent patterns. In fact, the number of models to compare remained quite low although the search space is the same.

6 Discussion

We presented the first uniform approach dealing with several dimensions of sequence analysis. Our framework relies on off-the-shelf ASP technology and high-level specifications expressed in ASP. In turn, we have shown how effortlessly the basic approach can be extended to condensed pattern extraction and illustrated how ASP's preference handling capacities can be used for extracting patterns of interest. In doing so, we demonstrated the ease of incorporating knowledge into

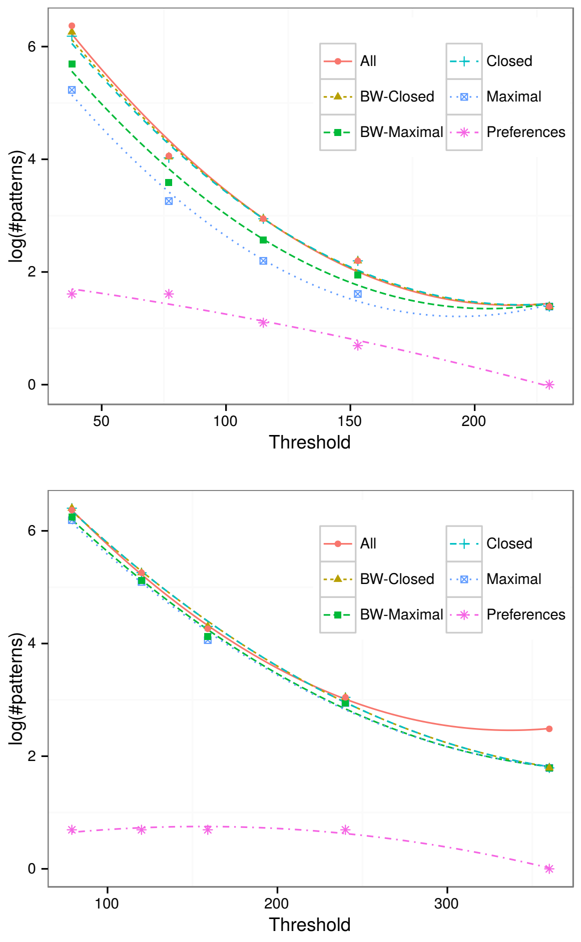

Figure 2: Numbers of frequent patterns wrt different thresholds. Top: `Unix user` database; bottom: `jmlr` database.

the ASP-based mining process. Although our declarative encodings cannot fully compete with the low-level efficiency of dedicated data mining systems on basic sequence mining tasks, our approach has an edge over them as regards expressiveness and extensibility. Nonetheless, our empirical analysis revealed that its performance matches that of a closely related CP-based approach [Negrevergne and Guns, 2015]. Beyond that, we have seen that the incorporation of preferences can lead to a significant reduction of pattern sets. All in all, our approach thus aims at processing mid-size databases and the enablement of analysts to extract patterns that they are really interested in.

Our approach follows the encoding principle of [Järvisalo, 2011], who used ASP for itemset mining. Unlike itemset mining, however, sequence mining is more demanding since we need to take the order of items in a pattern into account. The approach in [Guyet *et al.*, 2014] tackles another sequential pattern mining task, viz. episode mining, enumerating multiple embeddings within a single sequence. This task has also been addressed by means of Boolean Satisfiability (SAT) [Coquery *et al.*, 2012]. However, these approaches model sequential patterns with non-contiguous items by wildcards. In such a case, detecting embeddings is closer to string matching than subsequence finding. Unlike that, our encodings leave particular embeddings of a pattern implicit and focus on sequences for which some embedding exists. More generally, SAT lacks a modeling language, which makes it hard for any SAT-based approach to achieve a similar level of extensibility. The CPSM system [Negrevergne and Guns, 2015] pursues a CP-based approach and tackles sequential pattern mining tasks similar to ours. While we rely on fully declarative encodings, CPSM uses dedicated propagators for embedding computation. Although such propagators can be implemented by space-efficient special-purpose algorithms, they cannot easily be extended to accommodate further constraints. On the other hand, it will be interesting future work to equip ASP systems with similar constructs and to find a trade-off between declarativeness and effectiveness.

An original aspect of our approach is the incorporation of preferences by using the *asprin* system, as illustrated by the simple "implementation" of skypatterns on top of any underlying encoding of frequent patterns. This highlights another advantage of ASP for declarative pattern mining: It is not only effortless to add new knowledge, but also to incorporate new forms of reasoning into the main mining task, and thus, to limit the effort of implementing new tasks. Unlike this, [Ugarte *et al.*,] use dynamic constraint satisfaction problems to address skypatterns. The idea is to add constraints for avoiding solutions dominated by already extracted ones, and to post-process patterns for filtering optimal ones. The *asprin* system works in a similar yet more transparent way, as a user only needs to specify her preferences — and nothing else.

Generally speaking, we have demonstrated that the modeling capacities of ASP are appropriate to express complex sequence mining tasks, where choice rules define the search space, integrity constraints eliminate invalid candidate patterns, and preferences distinguish optimal outcomes. In view of the data-intense domain at hand, some efforts had to be taken to develop economic encodings of generic sequence mining tasks. Experts can now benefit from our work and take advantage of it by stating further constraints or preferences, respectively, for extracting the patterns of their interest.

Acknowledgments This work was partially funded by DFG grant SCHA 550/9.

References

[Brewka *et al.*, 2015] G. Brewka, J. Delgrande, J. Romero, and T. Schaub. asprin: Customizing answer set preferences without a headache. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 1467–1474. AAAI Press, 2015.

[Cho *et al.*, 2005] M. Cho, J. Pei, H. Wang, and W. Wang. Preference-based frequent pattern mining. *International Journal of Data Warehousing and Mining*, 1(4):56–77, 2005.

[Coquery *et al.*, 2012] E. Coquery, S. Jabbour, L. Saïs, and Y. Salhi. A SAT-based approach for discovering frequent, closed and maximal patterns in a sequence. In L. De Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. Lucas, editors, *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI'12)*, pages 258–263. IOS Press, 2012.

[De Bie, 2011] T. De Bie. Maximum entropy models and subjective interestingness: An application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23(3):407–446, 2011.

[Flouvat *et al.*, 2014] F. Flouvat, J. Sanhes, C. Pasquier, N. Selmaoui-Folcher, and J. Boulicaut. Improving pattern discovery relevancy by deriving constraints from expert models. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI'14)*, pages 327–332. IOS Press, 2014.

[Gebser *et al.*, 2014] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Clingo* = ASP + control: Preliminary report. In M. Leuschel and T. Schrijvers, editors, *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, *Theory and Practice of Logic Programming*, 14(4-5):Online Supplement, 2014. Available at http://arxiv.org/abs/1405.3694.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Guns *et al.*, 2011] T. Guns, S. Nijssen, and L. De Raedt. Itemset mining: A constraint programming perspective. *Artificial Intelligence*, 175(12-13):1951–1983, 2011.

[Guyet *et al.*, 2014] T. Guyet, Y. Moinard, and R. Quiniou. Using answer set programming for pattern mining. In *Actes des Huitièmes Journées d'Intelligence Artificielle Fondamentale (JIAF'14)*, 2014. Available at http://arxiv.org/abs/1409.7777.

[Järvisalo, 2011] M. Järvisalo. Itemset mining as a challenge application for answer set enumeration. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 304–310. Springer-Verlag, 2011.

[Kemmar *et al.*, 2015] A. Kemmar, S. Loudni, Y. Lebbah, P. Boizumault, and T. Charnois. PREFIX-PROJECTION global constraint for sequential pattern mining. In G. Pesant, editor, *Proceedings of the Twenty-first International Conference on Principles and Practice of Constraint Programming (CP'15)*, volume 9255 of *Lecture Notes in Computer Science*, pages 226–243. Springer-Verlag, 2015.

[Lichman, 2013] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[Lifschitz, 2008] V. Lifschitz. What is answer set programming? In D. Fox and C. Gomes, editors, *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, pages 1594–1597. AAAI Press, 2008.

[Muggleton and De Raedt, 1994] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[Negrevergne and Guns, 2015] B. Negrevergne and T. Guns. Constraint-based sequence mining using constraint programming. In L. Michel, editor, *Proceedings of the Twelfth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'15)*, volume 9075 of *Lecture Notes in Computer Science*, pages 288–305. Springer-Verlag, 2015.

[Padmanabhan and Tuzhilin, 1998] B. Padmanabhan and A. Tuzhilin. A belief-driven method for discovering unexpected patterns. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 94–100. AAAI Press, 1998.

[Shen *et al.*, 2014] W. Shen, J. Wang, and J. Han. Sequential pattern mining. In C. Aggarwal and J. Han, editors, *Frequent Pattern Mining*, pages 261–282. Springer-Verlag, 2014.

[Simons *et al.*, 2002] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[Ugarte *et al.*,] W. Ugarte, P. Boizumault, B. Crémilleux, A. Lepailleur, S. Loudni, M. Plantevit, C. Raïssi, and A. Soulet. Skypattern mining: From pattern condensed representations to dynamic constraint satisfaction problems. *Artificial Intelligence*. To appear.

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