Hybrid ASP-based Approach to Pattern Mining

Sergey Paramonov[a], Daria Stepanova[b], Pauli Miettinen[b]

[a]Department of Computer Science, KU Leuven, Belgium
[b]Max Planck Institute of Informatics, Saarbrücken, Germany
sergey.paramonov@kuleuven.be,{dstepano,pmiettin}@mpi-inf.mpg.de

Abstract. Detecting small sets of relevant patterns from a given dataset is a central challenge in data mining. The relevance of a pattern is based on user-provided criteria; typically, all patterns that satisfy certain criteria are considered relevant. Rule-based languages like Answer Set Programming (ASP) seem well-suited for specifying such criteria in a form of constraints. Although progress has been made, on the one hand, on solving individual mining problems and, on the other hand, developing generic mining systems, the existing methods either focus on scalability or on generality. In this paper we make steps towards combining local (frequency, size, cost) and global (various condensed representations like maximal, closed, skyline) constraints in a generic and efficient way. We present a hybrid approach for itemset and sequence mining which exploits dedicated highly optimized mining systems to detect frequent patterns and then filters the results using declarative ASP. Experiments on real-world datasets show the effectiveness of the proposed method and computational gains both for itemset and sequence mining.

Keywords: answer set programming, pattern mining, structured mining, sequence mining, itemset mining

1 Introduction

Motivation. Availability of vast amounts of data from different domains has led to an increasing interest in the development of scalable and flexible methods for data analysis. A key feature of flexible data analysis methods is their ability to incorporate users' background knowledge and different interestingness criteria. They are often provided in the form of *constraints* to the valid set of answers, the most common of which is the frequency threshold: a pattern is only considered interesting if it appears often enough. Mining all frequent (and otherwise interesting) patterns is a very general problem in data analysis, with applications in medical treatments, customer shopping sequences, Weblog click streams and text analysis, to name but a few examples.

Most data analysis methods consider only one (or few) types of constraints, limiting their applicability. Constraint Programming (CP) has been proposed as a general approach for (sequential) mining of frequent patterns [1], and Answer Set Programming (ASP) [6] has been proved to be well-suited for defining the constraints conveniently thanks to its expressive and intuitive modelling language and the availability of optimized ASP solvers (see e.g., [13,5,9] for existing approaches).

In general, all constraints can be classified into *local constraints*, that can be validated by the pattern candidate alone, and *global constraints*, that can only be validated via an exhaustive comparison of the pattern candidate against all other candidates. Combining local and global constraints in a generic way is an important and challenging problem, which has been widely acknowledged in the constrained-based mining community. Although progress has been made, on the one hand, on solving individual mining problems and, on the other hand, on developing generic mining systems, the existing methods either focus on scalability or on generality, but rarely address both of these aspects. This naturally limits the practical applicability of the existing approaches.

State of the art and its limitations. Purely declarative ASP encodings for frequent and maximal itemset mining was proposed in [13]. In this approach, first every item's inclusion into the candidate itemset is guessed, and the guessed candidate pattern is checked against frequency and maximality constraints. While natural, this encoding is not truly generic, as adding extra local constraints requires significant changes in it. Indeed, for a database, where all available items form a frequent (and hence maximal) itemset, the maximal ASP encoding has a single model. The latter is, however, eliminated once restriction on the length of allowed itemsets is added to the program. This is undesired, as being maximal is not a property of an itemset on its own, but rather its property in the context of a collection of other itemsets [3]. Thus, ideally one would be willing to first apply all local constraints and only afterwards construct a condensed representation of them, which is not possible in [13].

This shortcoming has been addressed in the recent work [5] on ASP-based sequential pattern mining, which exploits ASP preference-handling capacities to extract patterns of interest and supports the combination of local and global constraints. However, both [5] and [13] present purely declarative encodings, which suffer from scalability issues caused by the exhaustive exploration of the huge search space of candidate patterns. The subsequence check amounts to testing whether an embedding exists (matching of the individual symbols) between sequences. In sequence mining, a pattern of size m can be embedded into a sequence of size n in $O(n^m)$ different ways, therefore, clearly a direct pattern enumeration is unfeasible in practice.

While a number of individual methods tackling selective constraint-based mining tasks exist (see Tab. 1 for comparison) there is no uniform ASP-based framework that is capable of effectively combining constraints both on the global and local level and is suitable for itemsets and sequences alike.

Contributions. The goal of our work is to make steps towards building a generic framework that supports mining of condensed (sequential) patterns, which (1) effectively combines dedicated algorithms and declarative means for pattern mining and (2) is easily extendible to incorporation of various constraints. More specifically, the salient contributions of our work can be summarized as follows:

- We present a general extensible pattern mining framework for mining patterns of different types using ASP.
- We introduce a feature comparison, such as closedness under solutions, between different ASP mining models and dominance programming, which is a generic itemset mining language and solver.

Datatype	Task	[13]	[5]	[16]	Our work
Itemset	frequent pattern mining	✓	–	✓	✓
	condensed (closed, max, etc)	✓*	–	✓	✓
	condensed under constraints	–	–	✓	✓
Sequence	frequent pattern mining	–	✓	–	✓
	condensed (closed, max, etc)	–	✓	–	✓
	condensed under constraints	–	✓	–	✓

Table 1: Feature comparison between various ASP mining models and dominance programming ("–" : "not designed for this datatype", ✓* : only maximal is supported)

– We demonstrate the feasibility of our approach with an experimental evaluation across multiple itemset and sequence datasets.

2 Preliminaries

*** DS: to be contracted to 3 p. max ***

Let D be a dataset, $\mathcal{L}$ a language for expressing pattern properties or defining subgroups of the data, and q a selection predicate. The task of pattern mining is to find $Th(\mathcal{L}, \mathbf{r}, q) = \{\phi \in \mathcal{L} \mid q(\mathbf{r}, \phi) \text{ is true}\}$ (see the seminal work [14]).

Pattern mining has been mainly studied in three settings: itemsets, sequences and graphs. These settings are determined by the language of $\mathcal{L}$. We focus on the first two categories. In this section we briefly recap the necessary background both from the fields of pattern mining and Answer Set Programming (ASP).

2.1 Patterns

Sergey: relative frequency should be defined somewhere in this section **Itemsets**. *Itemsets* represent the most simple setting of frequent pattern mining. Let $\mathcal{I}$ be a set of items $\{o_1, o_2, \ldots, o_n\}$. Then a nonempty subset of $\mathcal{I}$ is called an *itemset*. A *transaction dataset* D is a collection of itemsets, $D = t_1, \ldots, t_m$, where $t_i \subseteq I$. For any itemset α, we denote the set of transactions that contain α as $D_\alpha = \{i \mid \alpha \subseteq t_i, t_i \in D\}$ and we refer to $|D_\alpha|$ as the *support* of α in D, written $s(\alpha)$. The *cardinality* (or *length*) of an itemset α is the number of items contained in it, i.e., $|\alpha| = |\{o_i \mid o_i \in \alpha\}|$.

Definition 1 (Frequent Itemset). *For a transaction dataset D and a frequency threshold $\sigma \geq 0$, an itemset α is* frequent *in D if $s(\alpha) \geq \sigma$.*

Example 1. Consider a transaction dataset D from Tab. 2. We have $\mathcal{I} = \{a, b, c, d, e\}$ and $|D| = 3$. For $\sigma = 2$, the following itemsets are frequent: $\alpha_1 = \{a\}$, $\alpha_2 = \{b\}$, $\alpha_3 = \{e\}$, $\alpha_4 = \{a, b\}$, $\alpha_5 = \{a, e\}$ and $\alpha_6 = \{b, e\}$. □

ID	a	b	c	d	e
1	✓	✓		✓	✓
2		✓	✓		✓
3	✓				✓

Table 2: Transaction database

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

Table 3: Sequence database

Sequences. A *sequence* is an ordered set of items $\langle s_1, \dots, s_n\rangle$. The setting of *sequence mining* includes two related yet different cases: frequent substrings and frequent subsequences. In this work we focus on the latter.

Definition 2 (Embedding in a Sequence). *Let $S = \langle s_1, \dots, s_m\rangle$ and $S' = \langle s'_1, \dots, s'_n\rangle$ be two sequences of size m and n respectively with $m \leq n$. The tuple of integers $e = (e_1, \dots, e_m)$ is an* embedding *of S in S' (denoted $S \sqsubseteq_e S'$) if and only if $e_1 < \dots < e_m$ and for any $i \in 1..m$ it holds that $s_i = s_{e_i}$.*

Example 2. For a dataset in Tab. 3 we have that $\langle b\,c\,e\,b\rangle \sqsubset_{e_1} \langle a\ b\ c\ d\ a\ e\ b\rangle$ for $e_1 = (2, 3, 6, 7)$ and analogousy, $\langle a\ a\ e\rangle \sqsubset_{e_2} \langle a\ b\ c\ d\ a\ e\ b\rangle$ with $e_2 = (1, 5, 6)$.

We are now ready to define inclusion relation for sequences.

Definition 3 (Sequence Inclusion). *Given two sequences $S = \langle s_1, \dots, s_n\rangle$ and $S' = \langle s'_1, \dots, s'_m\rangle$, of length m and n resp. with $n \leq m$, we say that S is included in S' or S is a* subsequence *of S' denoted by $S \sqsubseteq S'$ iff an embedding e of S in S' exists, i.e.*

$$S \sqsubseteq S' \leftrightarrow e_1 < \dots < e_m \textit{ and } \forall i \in 1 \dots m : s_i = s'_{e_i}. \tag{1}$$

Example 3. In Ex. 2 we have $\langle b\,c\,e\,b\rangle \sqsubset \langle a\ b\ c\ d\ a\ e\ b\rangle$ but $\langle a\ a\ e\rangle \not\sqsubseteq \langle b\ c\ e\ b\rangle$. □

Let D_S be the subset of D such that $S \sqsubseteq S'$ for all $S' \in D_S$ for a given sequential dataset $D = \{S_1, \dots, S_n\}$ and a sequence S.

Definition 4 (Frequent Sequence). *For a sequential dataset $D = \{S_1, \dots, S_n\}$ and a frequency threshold $\sigma \geq 0$, a sequence S is* frequent *in D if $|D_S| \geq \sigma$.*

Example 4. For a dataset in Tab. 3 and $\sigma = 2$, it holds that $\langle b\ c\ e\ b\rangle$ and $\langle a\,a\,e\rangle$ are frequent, while $\langle b\ d\ b\rangle$ is not. □

Note that $\sqsubseteq$ and $\subseteq$ are incomparable relations. Indeed, consider two sequences $s_1 = \langle a\,b\rangle$ and $s_2 = \langle b\,a\,a\rangle$. While $s_1 \subset s_2$, we clearly have that $s_1 \not\sqsubseteq s_2$.

2.2 Condensed Pattern Representations under Constraints

In data mining, constraints are typically specified by the user to encode domain background knowledge. In [17] four types of constraints are distinguished: constraints 1) over the pattern (e.g., restriction on its length), 2) over the cover set (e.g., minimal frequency), 3) over the inclusion relation (e.g., maximal allowed gap in sequential patterns) and 4) over the solution set (e.g., condensed representations).

Orthogonally, constraints can be classified into *local* and *global* ones. A constraint is *local* if deciding whether a given pattern satisfies it is possible without looking at other patterns. For example, minimal frequency or maximal pattern length are local constraints. On the contrary, deciding whether a pattern satisfies a *global* constraint requires comparing it to other patterns. All constraints from the 4th group are global ones. We are interested in global constraints related to condensed representations.

As argued in Sec. 1, the order in which constraints are applied influences the solution set [3]. In this work we take the same view as in [3] and aim at applying global constraints only after the local ones.

We now present the notions required in our pattern mining framework. Here, the definitions are given for itemsets; for sequences they are identical up to substitution of $\subset$ with $\sqsubset$ (subsequence relation). First, to rule out patterns that do not satisfy some of the local constraints, we introduce the notion of valididy.

Definition 5 (Valid pattern under constraints). *Let C be a constraint function from $\mathcal{L}$ to $\{\top, \bot\}$ and p be a pattern in $\mathcal{L}$, then a pattern is called* valid *iff $C(p) = \top$, otherwise it is referred as* invalid*.*

Example 5. Let C be a constraint function checking whether a given pattern has length at least 2. Then for Ex. 1, we have $C(\alpha_i) = \bot$, $i = 1..3$ and $C(\alpha_j) = \top$, $j = 4..6$. □

For detecting patterns that satisfy a given global constraint, the notion of *dominance* is of crucial importance.

Definition 6 (Dominated pattern under constraints). *Let C be a constraint function, and p be a pattern, then p is called* dominated *iff there exists a pattern $p' \in \mathcal{L}$ such that $p <^* p'$ and p' is valid under C.*

Example 6. In Ex. 1 for the closedness constraints we have that α_1 is dominated by α_4 and α_5. Analogously, α_2 is dominated by α_4 and α_6, while α_3 by α_5 and α_6. □

Definition 7 (Condensed pattern under constraints). *Let p be a pattern from $\mathcal{L}$, and let C be a constraint function, then a pattern p is called* condensed *under constraints iff it is valid and not dominated under C.*

In this work we primarily focus on the following condensed representations:

(i) **Maximal.** For itemsets p, q, $p <^* q$ holds iff $p \subset q$
(ii) **Closed.** For itemsets p, q, $p <^* q$ holds iff $p \subset q \wedge \mathit{support}(p) = \mathit{support}(q)$
(iii) **Free.** For itemsets p, q, $p <^* q$ holds iff $q \subset p \wedge \mathit{support}(p) = \mathit{support}(q)$
(iv) **Skyline.** For itemsets p, q, $p <^* q$ holds iff
  (a) $\mathit{support}(p) \leq \mathit{support}(q)$ and $\mathit{size}(p) < \mathit{size}(q)$ or
  (b) $\mathit{support}(p) < \mathit{support}(q)$ and $\mathit{size}(p) \leq \mathit{size}(q)$

For brevity we refer to the above constraints as max, cl, sky, and free.

2.3 Answer Set Programming

Answer Set Programming (ASP) [6] is a declarative problem solving paradigm oriented towards difficult search problems. ASP has its roots in Logic Programming and Non-monotonic Reasoning. An *ASP program Π* is a set of rules of the form

$$\texttt{a\_0 :- b\_1, ..., b\_k, not b\_k+1, ..., not b\_m,} \tag{2}$$

where $1 \leq k \leq m$, `a_0` is a classical literal, and not denotes default negation. A rule r of the form (2) is called a *fact* if $m = 0$. We omit the symbol `:-` when referring to facts. A rule without head literals is a *constraint*. A rule is *positive* if $k = m$.

An ASP program Π is *ground* if it consists of only ground rules, i.e. rules without variables. Ground instantiation $Gr(\Pi)$ of a nonground program Π is obtained by substituting variables with constants in all possible ways. The *Herbrand universe* $HU(\Pi)$ (resp. *Herbrand base* $HB(\Pi)$) of Π, is the set of all constants occurring in Π, (resp. the set of all possible ground atoms that can be formed with predicates and constants occurring in Π). Any subset of $HB(P)$ is a *Herbrand interpretation*. $MM(\Pi)$ denotes the set-inclusion minimal Herbrand interpretation of a ground positive program Π.

The semantics of ASP programs is given in terms of its answer sets. An interpretation I of Π is an *answer set* (or *stable model*) of Π iff $I \in MM(\Pi^I)$, where Π^I is the *Gelfond-Lifschitz (GL) reduct* [6] of Π, obtained from $Gr(\Pi)$ by removing (i) each rule r such that $Body^-(r) \cap I \neq \emptyset$, and (ii) all the negative atoms from the remaining rules. The set of answer sets of a program Π is denoted by $AS(\Pi)$.

Other relevant language constructs include conditional literals and cardinality constraints [22]. The former are of the form `a:b_1, ..., b_m`, the latter can be written as `s{c_1, ..., c_n }t`, where `a` and `b_i` are possibly default negated literals and each `c_j` is a conditional literal; `s` and `t` provide lower and upper bounds on the number of satisfied literals in a cardinality constraint. For instance, `1{a(X):b(X)}3` holds, whenever between 1 and 3 instance of `a(X)` (subject to `b(X)`) are true. Furthermore, aggregates are of the form `#count{J: item(I,J)}<N`. This atom is true, whenever for every `I`, s.t. `item(I,J)` is true, the number of `J` does not exceed `N`.

Example 7. Consider the program Π given as follows:

```
(1) pattern(1); (2) pattern(2); (3) item(1,a);
(4) item(1,b); (5) item(2,a);
(6) not_subset(J,I):-pattern(I), item(I,V), I != J,
                     pattern(J), not item(J,V).
```

The grounding $Gr(\Pi)$ of Π is obtained from Π by substituting I, J, V with `1,2` and `b` resp. The GL-reduct $\Pi^{I'}(\Pi)$ for I' containing facts of Π and `not_subset(2,1)` differs from $Gr(\Pi)$ only in that `not item(2,b)` is not in the body of the rule. Since I' is a minimal model of $\Pi^{I'}(\Pi)$, it is also an answer set of Π. □

3 Hybrid ASP-based Mining Approach

In this section we present our hybrid method to frequent pattern mining. Unlike previous ASP-based mining methods, our approach combines highly optimized algorithms for frequent pattern discovery with the declarative ASP means for their convenient post-processing. In this work we focus on itemset and sequence mining; however our approach can be also applied to subgraph discovery (details are left for future work).

Given a frequency threshold σ, a (sequential) dataset D and a set of constraints $\mathcal{C} = \mathcal{C}_l \cup \mathcal{C}_g$, where $\mathcal{C}_l$ and $\mathcal{C}_g$ are respectively local and global constraints, we proceed in two steps as follows.

Step 1. First, we launch a dedicated optimized algorithm to extract all (sequential) frequent patterns from a given dataset, satisfying the minimal frequency threshold σ. Here, any frequent pattern mining algorithm can be invoked, the ones we used: Eclat [24] for itemsets and PPIC [2] for sequences.

Step 2. Second, the computed patterns are postprocessed using the declarative means to select a set of *valid* patterns (i.e., those satisfying constraints in $\mathcal{C}_l$). For that the frequent patterns obtained in Step 1 are encoded as facts `item(i,j)` for itemsets and `seq(i,j,p)` for sequences where `i` is the pattern's ID, `j` is an item contained in it and `p` is its position. The local constraints in $\mathcal{C}_l$ are represented as ASP rules, which collect IDs of patterns satisfying constraints from $\mathcal{C}_l$ into the dedicated predicate `valid`, while the rest of the IDs into the `not_valid` predicate.

Finally, from all valid patterns a desired condensed representation is constructed by storing patterns `i` in the `selected` predicate if they are not `dominated` by other valid patterns based on constraints from $\mathcal{C}_g$. Following the principle of [13], in our work every answer set represents a single desired pattern, which satisfies both local and global constraints. The set of all such patterns forms a condensed representation. In what follows we discuss our encodings of local and global constraints in details.

3.1 Encoding Local Constraints

In our declarative program we specify local constraints by the predicate `valid`, which reflects the conditions stated in Def. 5. For every constraint in the given set $\mathcal{C}_l$ we have a set of dedicated rules, stating when a pattern is not valid. For example, a constraint checking whether the cost of items in a pattern exceeds a given threshold N is encoded using the following rule:

```
not_valid(I) :- #sum{C:item(I,J),cost(J,C)} > N,
                     pattern(I).
```

A similar rule for sequences will be as follows:

```
not_valid(I) :- #sum{C:seq(I,J,P),cost(P,C)} > N,
                     pattern(I).
```

Similarly, one can define arbitrary domain constraints on patterns.

Example 8. Consider a dataset storing moving habits of young people during their studies. Let the dedicated frequent sequence mining algorithm return the following patterns: $s_1 = \langle bG\ mF\ ba\ mG\ ma \rangle$; $s_2 = \langle bF\ mG\ ba\ mF\ ma \rangle$; $s_3 = \langle bA\ ba\ ma \rangle$, where

```
% people born in Germany or France are Europeans
seq(X,e,P) :- seq(I,bG,P)
seq(X,e,P) :- seq(I,bF,P)
% collect moving actions regardless of a European country
seq(I,m,P) :- seq(I,mG,P)
seq(I,m,P) :- seq(I,mF,P)
% keep patterns about Europeans who finished masters in Germany
keep(X) :- pattern(X), seq(X,ma,P'+1), seq(X,mG,P'), seq(X,e,P)
keep(X) :- pattern(X), seq(X,bG,P'), not seq(X,m,P), P>P'
% if a pattern should not be kept, it is not valid
not_valid(X) :- pattern(X), not keep(X)
```

Listing 1.1: Moving Habits of People during Studies

bG, bF, bA stand for born in Germany, France and America, ba, ma stand for bachelors and masters and the predicates mG, mF reflect that a person moved to Germany and France resp. Suppose, we are only interested in moving habits of those Europeans, who got masters degree from a German university. The local domain constraint expressing this would state that (1) bA should not be in the pattern, while (2) either both bG and ma should be in it without any $m_{\_}$ in between or mG should preceed ma. These constraints are encoded in the program in List. 1.1. From the answer set of this program we get that both s_2 and s_3 are not valid, while s_1 is. □

To combine all local constraints from $\mathcal{C}_l$ we add to our program a generic rule specifying that a pattern `I` is valid whenever `not_valid(I)` cannot be inferred.

```
valid(I) :- pattern(I), not not_valid(I)
```

Patterns `i`, for which `valid(i)` is deduced are then further analysed to construct a condensed representation based on global constraints from $\mathcal{C}_g$.

3.2 Encoding Global Constraints

The key for encoding global constraints is the declarative formalization of the dominance relation (Def. 6). For example, for itemsets the maximality constraint boils down to pairwise checking of subset inclusion between patterns. For sequences this requires a check of embedding existence between sequences.

Regardless of a pattern type from $\mathcal{L}$ and a constraint from $\mathcal{C}_g$ every encoding presented in this section is supplied with a rule, which guesses (`selected/1` predicate) a single valid pattern to be a candidate for inclusion in the condensed representation, and a constraint that rules out dominated patterns thus enforcing a different guess.

```
1 {selected(I) : valid(I)} 1.

:- dominated.
```

```
1 % I is not a subset of J if I has items that are not in J
2 not_subset(J) :- selected(I), item(I,V), not item(J,V),
3                  valid(J), I != J.
4 % derive dominated whenever I is subset of J
5 dominated :- selected(I), valid(J),
6              not not_subset(J), I != J.
```

Listing 1.2: Maximal Itemsets Encoding

```
% support and size comparison among patterns
g_in_size_and_geq_in_freq(J) :- selected(I), support(I,X),
                                support(J,Y), size(I,Si),
                                size(J, Sj), Si <  Sj, X <= Y.
geq_in_size_and_g_in_freq(J) :- selected(I), support(I,X),
                                support(J,Y), size(I,Si),
                                size(J, Sj), Si <= Sj, X <  Y.
% derivation of the domination condition
dominated :- valid(J), g_in_size_and_geq_in_freq(J).
dominated :- valid(J), geq_in_size_and_g_in_freq(J).
```

Listing 1.3: Skyline Itemsets Encoding

In what follows, we discuss concrete realizations of the dominance relation both for itemsets and sequences for various global constraints, i.e., we present specific rules related to the derivation of `dominated/0` predicate.

Itemset Mining. We first provide an enoding for maximal itemset mining in List 1.2. To recall, a pattern is *maximal* if none of its supersets is frequent. An itemset I is included in J iff for every item $i \in I$ we have $i \in J$. We encode the violation of this condition in lines (1)-(3). The second rule presents the dominance criteria.

For closed itemset mining a simple modification of List 1.2 is required. An itemset is *closed* if none of its supersets has the same support. Thus to both of the rules from List. 1.2 we need to add atoms `support(I,X), support(J,X)`, which store the support sets of `I` and `J` (output by the dedicated algorithms).

For free itemset mining the rules of the maximal encoding are changed as follows:

```
not_superset(J) :- selected(I), item(J,V), not item(I,V),
                   valid(J), I != J.
dominated :- selected(I), valid(J), support(I,X),
             I != J, not not_superset(J), support(J,X).
```

Finally, the skyline itemset encoding is given in List 1.3, where the first two rules specify the conditions (a) and (b) for skyline itemsets from Sec. 2.

Sequence Mining. The subpattern relation for sequences is slightly more involved, than for itemsets, as it preserves the order of elements in a pattern. To recall, a sequence S is included in S' iff an embeding e exists, such that $S \sqsubseteq_e S'$.

```
 1 % if V appears in a valid pattern I, derive in(V,I)
 2 in(V,I) :- seq(I,V,P), valid(I).
 3 % I is not a subset of J if I has V that J does not have
 4 not_subset(J) :- selected(I), valid(J), I != J,
 5                  seq(I,V,P), not in(V,J).
 6 % if for a subseq <V,W> in I there is V followed
 7 % by W in J then deduce domcand(V,J)
 8 domcand(V,J,P) :- selected(I), seq(I,V,P), seq(I,W,P+1), I != J
 9                   valid(J), seq(J,V,Q), seq(J,W,Q'), Q'>Q.
10 % if domcand(V,J) does not hold for some V in I
11 % and a pattern J then derive not_dominated_by(J)
12 not_dominated_by(J) :- selected(I), seq(I,V,P), seq(I,W,P+1),
13                        I != J, valid(J), not domcand(V,P,J).
14 % if neither not_dominated_by(J) nor not_subset_of(J)
15 % are derived for some J, then I is dominated
16 dominated :- selected(I), valid(J), I != J,
17              not not_subset_of(J), not not_dominated_by(J).
```

Listing 1.4: Maximal Sequence Encoding

In List. 1.4 we present the encoding for maximal sequence mining. A selected pattern is not maximal if it has at least one valid superpattern. We rule out patterns that are for sure not superpatterns of a selected sequence. First, obviously J is not a superpattern of I if it is not its superset (lines (4)-(5)), i.e., if not_subset(J) is derived, then J does not dominate I. If J is a superset of I then to ensure that I is not dominated by J, the embedding existence has to be checked (lines (6)-(9)). I is not dominated by J if an item exists in I, which together with its sequential neibourgh cannot be embedded in J. This condition is checked in lines (10)-(13), where domcand(V,J,P) is derived if for an item V at position P and its follower, embedding in J can be found.

The encoding for closed sequence mining is obtained from the maximal sequence encoding analogously as it is done for itemsets. The rules for free sequence mining are constructed by substituting lines (4)-(13) of List. 1.4 with the following ones:

```
 4 not_superset(J) :- selected(I), in(V,J),
 5                    not in(V,I), I != J.
 6 domcand(V,J) :- selected(I), seq(J,V,P), item(J,P+1,W),
 7                 item(I,V,Q), seq(I,W,Q'),Q'>Q, I != J.
 8 not_dominated_by(J) :- selected(I), valid(J), I != J,
 9                        seq(J,V,P), seq(J,W,P+1),
10                        not domcand(V,J).
```

Finally, the encoding for mining skyline sequences is given in List. 1.5. The conditions (a) and (b) from Sec. 2 are tested in lines (9)-(12), ensuring that neither (a) nor (b) hold for any valid pattern J.

```
% (1) sup(I) <= sup(J) and size(I) < size(J)
geq_sup_g_size(J) :- selected(I), valid(J), I != J,
                     support(I,X), support(J,Y), X<=Y,
                     size(I,X'), size(J,Y'), X'<Y'.
% (2) sup(I) < support(J) and size(I) <= size(J)
g_sup_geq_size(J) :- selected(I), valid(J), I != J,
                     sup(I,X), support(J,Y), X<Y,
                     size(I,X'), size(J,Y'), X'<=Y'.
% if neither (1) nor (2) holds for J, I is not dom. by J
not_dominated_by(J) :- selected(I), I != J, valid(J),
                       not geq_sup_g_size(J),
                       not g_sup_geq_size(J)
% otherwise it is
dominated :- selected(I), valid(J), I != J,
             not not_dominated_by(J).
```

Listing 1.5: Sky Sequence Encoding

4 Evaluation

In this section we evaluate the proposed hybrid approach. More specifically, we investigate the following experimental questions.

- **Q_1**: how does the runtime of our method compare to the existing ASP-based sequence mining models?
- **Q_2**: what is the runtime gap between the specialized mining languages such as dominance programming and our method?
- **Q_3**: what is the influence of local constraints on the runtime of our method?

In **Q_1** we compare with the ASP-based model from [5], in **Q_2** we investigate how big is the runtime difference between specialized mining languages for itemset mining and our ASP-based model. We do not compare our method with the existing ASP model [13] for itemset mining, since the latter focuses only on frequent itemset mining and is not applicable to the construction of condensed representations under constraints as in [16]. Finally, in **Q_3** we look at the effect on runtime caused by the addition of local constraints.

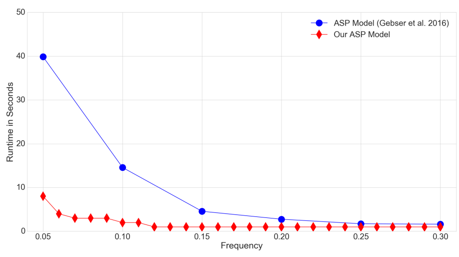
We evaluated our hybrid mining approach on various standard itemset and sequence datasets including *Mushrooms*, *Vote*[1], *JMLR*, *Unix Users* and *iPRG*[2]

All experiments have been performed on a desktop with Ubuntu 14.04 64-bit environment, Intel Core i5-3570 4xCPU 3.40GHz and 8GB memory using clingo 4.5.4 [3] and C++14 for the wrapper. The timeout is set to one hour. Free pattern mining demonstrates the same runtime patterns as closed, due to the symmetric encoding, and is omitted here. We do not perform comparison with algorithms dedicated to only one type

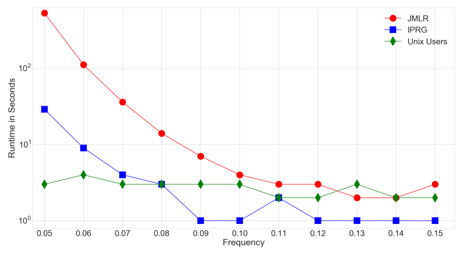
[1] https://dtai.cs.kuleuven.be/CP4IM/datasets/

[2] https://dtai.cs.kuleuven.be/CP4IM/cpsm/datasets.html

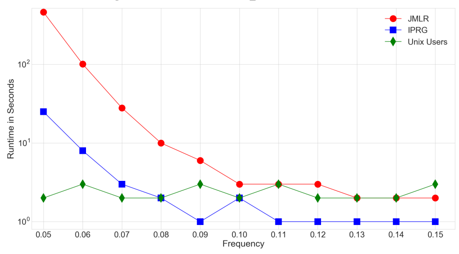
[3] http://potassco.sourceforge.net



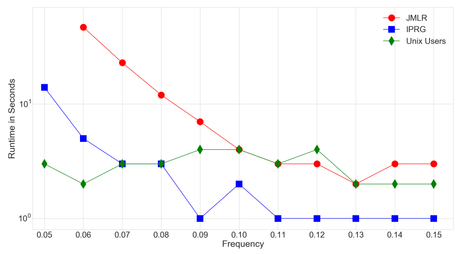
(a) Comparing with ASP sequence model [5] on the 200 generated sample (closed)



(b) Maximal sequence patterns



(c) Closed sequence patterns



(d) Skyline sequence patterns

Fig. 1: Investigating $\mathbf{Q_1}$: comparison with pure ASP model (1a) and maximal (1b), closed (1c), skyline (1d) sequence mining evaluation on datasets: JMLR, Unix Users, iPRG

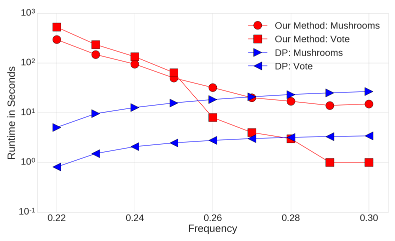
of problems, since they are known to be more efficient than general declarative mining approaches [17].

To investigate $\mathbf{Q_1}$, in Figure 1a, we compare the ASP model [5] with our method on the default 200 sequence sample, generated by the tool[4] from [5]. We performed the comparison on the synthetic data, as the sequence-mining model [5] failed to compute condensed representations on any of the standard sequence datasets for any support threshold value. One can observe that our method consistently outperforms the purely declarative approach from [5] and the advantage naturally becomes more apparent for smaller threshold values.

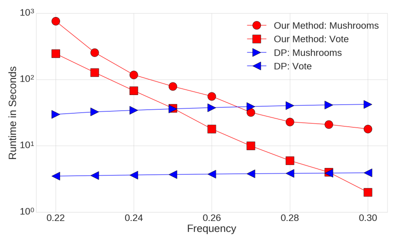
In Fig. 1b, 1c and 1d we present the runtimes of our method for *maximal, closed* and *skyline* pattern mining settings resp. on JMRL, Unix Users and iPRG datasets. In contrast to [5] our method managed to produce results on all of these datasets for reasonable threshold values within a couple of minutes (the runtime on Unix Users shows slight fluctuations within a couple of seconds Sergey: it is just runtime is low on that dataset, so 2 second fluctuations occur).

To investigate $\mathbf{Q_2}$, we compare out-of-the-box performance of Dominance Programming (DP) with our approach on closed, maximal and skyline itemset mining problems using standard datasets Vote and Mushrooms. As we see in Fig. 2a and 2b, on average, DP is one-two orders of magnitude faster, this gap is, however, diminishing with

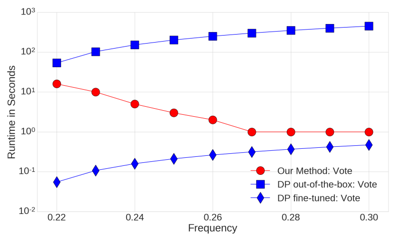
[4] https://sites.google.com/site/aspseqmining



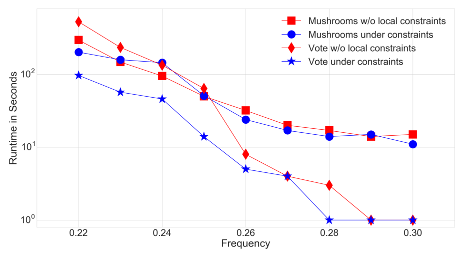
(a) Closed Itemset Mining: comparing with DP on Vote, Mushrooms datasets



(b) Maximal Itemset Mining: comparing with DP on Vote, Mushrooms datasets



(c) Skyline Itemset Mining: comparing on Vote dataset with out-of-the box and fine-tune DP



(d) Closed Itemset Mining: local constraint enable faster propagation and speedup the search

Fig. 2: Investigating $\mathbf{Q_2}$: comparison with Dominance Programming (2a, 2b, 2c); and $\mathbf{Q_3}$: the effect of local constraints on runtime (2d)

the growth of the frequency. Surprisingly, our approach is significantly faster than DP out-of-the-box for skyline patterns (see Fig. 2c). This holds also for other datasets, not depicted in the figure. It is possible to fine-tune DP to change its behaviour to close this gap, namely we analyzed all options of DP and found that besides of the default flag *skyline* for skyline patterns, it has the option *skyline+*, which computes the same patterns but applies operators in a different order. Then DP demonstrates one-two orders of magnitude better performance, as can be seen in Fig. 2c. However, fine-tuning such a system requires either understanding of the inner mechanisms of the system or trying all available options.

To address $\mathbf{Q_3}$ we introduced three simple local constraints for the itemset mining setting from $\mathbf{Q_2}$: two size constraints *size(I)* > 2 and *size(I)* < 7 and a cost constraint: each item gets weight equal to its value with the maximal budget of n, which is set to 20 in the experiments (Vote has 435 transactions and 48 binary items and Mushrooms has 8124 and 119 respectively). In Figure 2d, we present the results for closed itemset mining with and without local constraints (experiments with other local constraints demonstrate a similar runtime pattern and are not depicted here for space reasons). As we see local constraints do allow for better propagation and speedup the search. One of the key design features of our encoding is the filtering technique used to select candidate patterns among only valid patterns. This effect can be clearly seen, e.g., on the Vote dataset in Fig. 2d, where for certain frequencies the runtime gap is close to an order of magnitude.

In all experiments, the first step of the method contributes to less than five percent of runtime. Overall, our method can handle real world datasets for sequential pattern mining as demonstrated in $\mathbf{Q_1}$. In many cases its performance is close to the specialized mining languages, as shown in $\mathbf{Q_2}$. Finally, as demonstrated in $\mathbf{Q_3}$ various local constraints can be effectively incorporated into our encoding bringing additional performance benefits.

5 Related Work

The problem of enhancing pattern mining by injecting various user-specified constraints has recently gained increasing attention. On the one hand, optimized dedicated approaches exist, in which some of the constraints are deeply integrated into the mining algorithm, e.g., [19]. On the other hand, purely declarative methods based on Constraint Programming [21,17,15], SAT solving [12,11] and ASP [13,5,9] have been proposed.

Techniques from the last group are the closest to our work. However, unlike our method, they typically focus only on one particular pattern type and consider local constraints and condensed representations in isolation [20,23] (with exceptions [16,7] for CP-based itemset mining, [5] for ASP-based sequence mining, and a theoretical framework for structured pattern mining [8]).

In [13,5] in contrast to our work, purely declarative ASP methods have been proposed, which do not admit the integration of highly-optimized mining algorithm, which limits their practicability.

Paramonov et al. presents an initial formalization of graph mining using logic programming in [18] and Van der Hallen et al. [10] cover theoretical and practical problems of modeling graph mining using pure logical models indicating that higher order reasoning or a hybrid approach is needed to model complex structured mining problems in general case.

6 Conclusion

We have presented a hybrid approach for condensed itemset and sequence mining, which uses the optimized dedicated algorithms to determine the frequent patterns and post-filters them using a declarative ASP program. The idea of exploiting ASP for pattern mining is not new; it was studied for both itemsets and sequences. However, unlike previous methods we made steps towards optimizing the declarative techniques by making use of the existing highly optimized methods and also integrated the dominance programming machinery in our implementation to allow for combining local and global constraints on a generic level.

One of the possible future directions is to generalize the proposed approach into an iterative algorithm, where dedicated data mining and declarative methods are interlinked and applied in an iterative fashion. Indeed, in principle a set of constraints can be split into two parts: those that can be effectively applied using declarative means and those for which dedicated algorithms are expected to be more effective.

Another promising research stream concerns a common *decomposition* technique often exploited in databases. Here, one can decompose a given dataset into several parts, such that the frequent patterns can be identified in these parts separately, and then the results for each of the parts can be conveniently combined. Our hybrid constraint-based mining approach can greatly benefit from such decomposition techniques. In fact decomposition can be performed at different levels. First possibility is to split the original dataset into components and then apply our hybrid approach on each component separately. Second option would be perform data splitting on the pattern level, i.e., once the frequent patterns are computed, we could come up with their splitting and then perform post-processing on every pattern group considered individually. Yet combinations of the two strategies are possible.

Orthogonal to this, one can exploit HEX-programs [4], which effectively combine logic programming with external computations in the of hybrid mining.

Last but not least, materialization of the presented ideas on other pattern types including graphs and sequences of itemsets instead of sequences of individual symbols is a promising and interesting future direction.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI/MIT Press (1996)
2. Aoga, J.O.R., Guns, T., Schaus, P.: An efficient algorithm for mining frequent sequence with constraint programming. In: ECML PKDD. pp. 315–330 (2016)
3. Bonchi, F., Lucchese, C.: On condensed representations of constrained frequent patterns. Knowl. Inf. Syst. 9(2), 180–201 (2006)
4. Eiter, T., Brewka, G., Dao-Tran, M., Fink, M., Ianni, G., Krennwallner, T.: Combining nonmonotonic knowledge bases with external sources. In: Frontiers of Combining Systems, 7th International Symposium, FroCoS Italy, September 16-18. pp. 18–42 (2009)
5. Gebser, M., Guyet, T., Quiniou, R., Romero, J., Schaub, T.: Knowledge-based sequence mining with ASP. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016 (2016)
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. of ICLP/SLP. pp. 1070–1080 (1988)
7. Guns, T., Dries, A., Nijssen, S., Tack, G., Raedt, L.D.: Miningzinc: A declarative framework for constraint-based mining. Artif. Intell. 244, 6–29 (2017)
8. Guns, T., Paramonov, S., Négrevergne, B.: On declarative modeling of structured pattern mining. In: Declarative Learning Based Programming, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016. (2016)
9. Guyet, T., Moinard, Y., Quiniou, R.: Using answer set programming for pattern mining. CoRR abs/1409.7777 (2014)
10. van der Hallen, M., Paramonov, S., Leuschel, M., Janssens, G.: Knowledge representation analysis of graph mining. CoRR abs/1608.08956 (2016)
11. Jabbour, S., Sais, L., Salhi, Y.: Boolean satisfiability for sequence mining. In: 22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013. pp. 649–658 (2013)
12. Jabbour, S., Sais, L., Salhi, Y.: Decomposition based SAT encodings for itemset mining problems. In: PAKDD. pp. 662–674 (2015)

13. Järvisalo, M.: Itemset mining as a challenge application for answer set enumeration. In: Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings. pp. 304–310 (2011)
14. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Min. Knowl. Discov. 1(3), 241–258 (1997)
15. Métivier, J., Loudni, S., Charnois, T.: A constraint programming approach for mining sequential patterns in a sequence database. CoRR abs/1311.6907 (2013)
16. Négrevergne, B., Dries, A., Guns, T., Nijssen, S.: Dominance programming for itemset mining. In: 2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013. pp. 557–566 (2013)
17. Négrevergne, B., Guns, T.: Constraint-based sequence mining using constraint programming. In: CPAIOR. pp. 288–305 (2015)
18. Paramonov, S., van Leeuwen, M., Denecker, M., Raedt, L.D.: An exercise in declarative modeling for relational query mining. In: ILP. pp. 166–182 (2015)
19. Pei, J., Han, J.: Can we push more constraints into frequent pattern mining? In: ACM SIGKDD, Boston, MA, USA, August 20-23, 2000. pp. 350–354 (2000)
20. Pei, J., Han, J., Mao, R.: CLOSET: an efficient algorithm for mining frequent closed itemsets. In: 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, USA, May 14, 2000. pp. 21–30 (2000)
21. Rojas, W.U., Boizumault, P., Loudni, S., Crémilleux, B., Lepailleur, A.: Mining (soft-) skypatterns using dynamic CSP. In: CPAIOR. pp. 71–87 (2014)
22. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. 138(1-2), 181–234 (2002)
23. Yan, X., Han, J., Afshar, R.: Clospan: Mining closed sequential patterns in large datasets. In: In SDM. pp. 166–177 (2003)
24. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. Tech. rep., Rochester, NY, USA (1997)