Hybrid ASP-based Approach to Pattern Mining

SERGEY PARAMONOV
KU Leuven
(e-mail: sergey.paramonov@kuleuven.be)

DARIA STEPANOVA and PAULI MIETTINEN
Max Planck Institute for Informatics
(e-mail: {dstepano,pmiettin}@mpi-inf.mpg.de)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

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, sequence and graph mining which exploits dedicated highly optimized mining systems to detect frequent patterns and then filters the results using declarative ASP. To further demonstrate the generic nature of our hybrid framework we apply it to a problem of approximately tiling a database. Experiments on real-world datasets show the effectiveness of the proposed method and computational gains for itemset, sequence and graph mining, as well as approximate tiling.

KEYWORDS: answer set programming, pattern mining, structured mining, sequence mining, itemset mining, graph 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 criteria of interest. 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) (Négrevergne and Guns 2015; Guns et al. 2017) has been proposed as a general approach for (sequential) mining of frequent patterns (Agrawal et al. 1996), and Answer Set Programming (ASP) (Gelfond and Lifschitz 1988) 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., (Järvisalo 2011; Gebser et al. 2016; Guyet et al. 2014) 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 constraint-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 were proposed by Järvisalo (2011). In this approach, every item's inclusion into the candidate itemset is guessed at first, 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 in the context of a collection of other itemsets (Bonchi and Lucchese 2006). 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 the approach of Järvisalo (2011).

This shortcoming has been addressed in the recent work (Gebser et al. 2016) 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 Gebser et al. (2016) and Järvisalo (2011) 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, sequences and graphs 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 extendable 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.
- In addition to the classical pattern mining problems, we demonstrate the generic nature of our framework by applying it to a problem of approximately tiling a database.
- We introduce a feature comparison, such as closedness under solutions, between different

Datatype	**Task**	Järvisalo 2011	Gebser et al. 2016	Négrevergne et al. 2013	**Our work**
Itemset	frequent pattern mining	✓	–	✓	✓
	condensed (closed, max, etc)	✓*	–	✓	✓
	condensed under constraints	–	–	✓	✓
Sequence	frequent pattern mining	–	✓	–	✓
	condensed (closed, max, etc)	–	✓	–	✓
	condensed under constraints	–	✓	–	✓
Graphs	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)

ASP mining models and dominance programming, which is a generic itemset mining language and solver.

- We demonstrate the feasibility of our approach with an experimental evaluation across multiple itemset and sequence datasets using state-of-the-art ASP solvers.

Structure. After providing necessary background in Sec. 2 we introduce our approach in Sec. 3, and discuss approximate pattern mining in Sec. 4. Experimenal results are descibed in Sec. 5, while related work and final remarks are provided in Sec. 6 and Sec. 7 respectively.

2 Preliminaries

In this section we briefly recap the necessary background both from the fields of pattern mining and Answer Set Programming (ASP).

Let D be a dataset, $\mathscr{L}$ 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 $Th(\mathscr{L}, D, q) = \{\phi \in \mathscr{L} \mid q(D,\phi) \text{ is true}\}$, that is, to find all patterns $\phi \in \mathscr{L}$ that are selected by q (see, e.g, the seminal work of Mannila and Toivonen (1997)).

Pattern mining has been mainly studied for itemsets, sequences, graphs and tilings. These settings are determined by the language of $\mathscr{L}$. In this work we discuss all of these pattern types.

2.1 Patterns

2.1.1 Itemsets

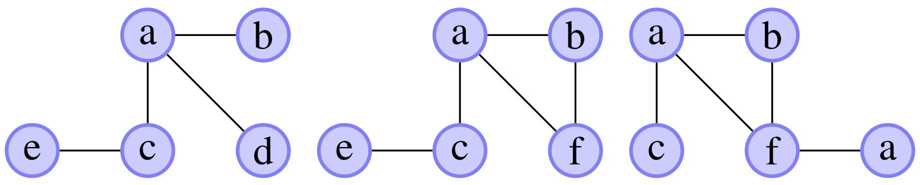
Itemsets represent the most simple setting of frequent pattern mining. Let $\mathscr{I}$ be a set of items $\{o_1, o_2, \ldots, o_n\}$. A nonempty subset of $\mathscr{I}$ is called an *itemset*. A *transaction dataset* D is a collection of itemsets, $D = \{t_1, \ldots, t_m\}$, where $t_i \subseteq \mathscr{I}$. For any itemset α, we denote the set of transactions that contain α as $D_\alpha = \{i \mid \alpha \subseteq t_i, t_i \in D\}$; we refer to D_α as the *cover* of an itemset

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

Table 2: Transaction database

ID	Sequence
1	$\langle abcdaeb\rangle$
2	$\langle bceb\rangle$
3	$\langle aae\rangle$

Table 3: Sequence database



(a) Graph G_1 (b) Graph G_2 (c) Graph G_3

Figure 1: Graph examples

α and to $|D_\alpha|$ as the *support (frequency)* of α in D, written $sup(\alpha)$. The *relative frequency* of α in D refers to the ratio between $sup(\alpha)$ and $|D|$. The *cardinality* (or *size*) $|\alpha|$ of an itemset α is the number of items contained in it.

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

We illustrate the introduced notions by the following example.

Example 1. *Consider a transaction dataset D from Tab. 2. We have $\mathscr{I} = \{a,b,c,d,e\}$ and $|D| = 3$. For $\sigma = 2$, the following itemsets are frequent: $\alpha_1=\{a\}$, $\alpha_2=\{b\}$, $\alpha_3=\{e\}$, $\alpha_4=\{a,e\}$ and $\alpha_5=\{b,e\}$. Moreover, it holds that $D_{\alpha_4} = \{1,3\}, D_{\alpha_5} = \{1,2\}$, and the coverage for the rest of the itemsets can be analogously found.* □

2.1.2 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 $e_1 < \ldots < e_m$ and for any $i \in \{1,\dots,m\}$ it holds that $s_i = s'_{e_i}$.*

Example 2. *For a dataset in Tab. 3 we have that $\langle bceb\rangle \sqsubseteq_{e_1} \langle abcdaeb\rangle$ for $e_1 = (2,3,6,7)$ and analogously, $\langle aae\rangle \sqsubseteq_{e_2} \langle abcdaeb\rangle$ with $e_2 = (1,5,6)$.* □

[1] In *frequent pattern mining*, often, a *relative threshold*, i.e., $\sigma/|D|$ is specified by the user.

We are now ready to define an 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 size m and n, respectively, 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.*

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

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

For a given sequence S and a sequential dataset $D = \{S_1, \dots, S_n\}$ we denote by D_S the subset of D s.t. $S \sqsubseteq S'$ for all $S' \in D_S$. The support of S is $sup(S) = |D_S|$. Frequent sequences are defined analogously to frequent itemsets.

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

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 $s_1 \not\sqsubset s_2$.

2.1.3 Graphs

A *graph G* is a triple $\langle V, E, l \rangle$ where V is a set of vertices, E is a set of edges and l_G is a labeling function that maps each edge and each vertex to a label.

Definition 5 (Graph Isomorphism). *Given two graphs $G = \langle V, E, l \rangle$ and $G' = \langle V', E', l' \rangle$, we say that G is isomorphic to G' iff there is an injective function f such that*

- *if $v \in V$ then $f(v) \in V'$ and $l_G(v) = l_{G'}(f(v))$*
- *if $e \in E$ then $f(e) \in E'$ and $l_G(e) = l_{G'}(f(e))$.*

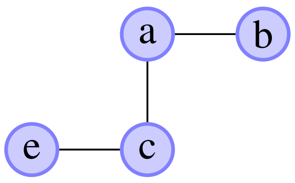
In this work we consider undirected graphs. Moreover, we primarily focus on two settings: a general one, where labels of the graph are not necessarily unique and a more restricted one, where unique labels are ensured.

Example 5. *The graphs in Fig. 1a and 1b are unique-labeled ones, while the graph from Fig. 1c is clearly not.* □

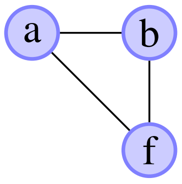
In the general setting, the problem of deciding whether isomorphism exists among two graphs is NP-complete (Cook 1971). However, several restricted settings have been identified, for which the problem can be solved in polynomial time, e.g., unique-labelled undirected graphs.

Definition 6 (Graph Inclusion). *Given two graphs $G = \langle V, E \rangle$ and $H = \langle U, F \rangle$, such that $|V| \leq |F|$, we say that G is an* (isomorphic) subgraph *of H denoted by $G \sqsubseteq H$ iff there exists a subgraph $H' = \langle U' \subseteq U, F' \subseteq F \rangle$ such that G is isomorphic to H'.*

Example 6. *Graph G'_1 of Fig. 2 is an isomorphic subgraph of graphs G_1 and G_2 of Fig. 1; it is not isomorphic to graph G_3. Graph G'_2 is isomorphic to graphs G_2 and G_3, but it is not isomorphic to graph G_1.* □



(a) Graph G_1'



(b) Graph G_2'

Figure 2: Isomorphic subgraphs for graphs in Fig. 1

2.2 Condensed Pattern Representations under Constraints

In data mining, constraints are typically specified by the user to encode domain background knowledge. Négrevergne and Guns (2015) distinguish four types of constraints: 1) constraints over the pattern (e.g., restriction on its size), 2) constraints over the cover set (e.g., minimal frequency), 3) constraints over the inclusion relation (e.g., maximal allowed gap in sequential patterns) and 4) constraints 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 size 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 (Bonchi and Lucchese 2006). Following Bonchi and Lucchese (2006), in this work we apply global constraints only after local ones.

We now present the notions required in our pattern mining framework. Here, the definitions are given for itemsets; for sequences and graphs 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 validity.

Definition 7 (Valid pattern under constraints)**.** *Let C be a constraint function from $\mathscr{L}$ to $\{\top, \bot\}$ and let p be a pattern in $\mathscr{L}$. Then the pattern p is called* valid *iff $C(p) = \top$; otherwise it is referred as* invalid.

Example 7. *Let C be a constraint function checking whether a given pattern is of size at least 2. Then in Ex. 1, we have $C(\alpha_i) = \bot$, $i \in \{1,2,3\}$, and $C(\alpha_j) = \top$, $j \in \{4,5\}$.* □

For detecting patterns that satisfy a given global constraint, the notion of *dominance* is of crucial importance. Intuitively, a dominance relation reflects pairwise preference ($<^*$) between patterns and it is specific for each mining setting. In this work we primarily focus on global constraints related to maximal, closed, free and skyline condensed representations, for which $<^*$ is defined as follows:

(i) **Maximal.** For itemsets p and q, $p <^* q$ holds iff $p \subset q$
(ii) **Closed.** For itemsets p and q, $p <^* q$ holds iff $p \subset q$ and $sup(p) = sup(q)$
(iii) **Free.** For itemsets p and q, $p <^* q$ holds iff $q \subset p$ and $sup(p) = sup(q)$
(iv) **Skyline.** For itemsets p and q, $p <^* q$ holds iff

 (a) $sup(p) \leq sup(q)$ and $size(p) < size(q)$ or
 (b) $sup(p) < sup(q)$ and $size(p) \leq size(q)$

Dominated patterns under constraints are now formally defined.

Definition 8 (Dominated pattern under constraints)**.** *Let C be a constraint function, and let p be a pattern, then p is called* dominated *iff there exists a pattern $p' \in \mathscr{L}$ such that $p <^* p'$ and p' is valid under C.*

Example 8. *In Ex. 1 for the maximality constraint we have that α_1 is dominated by α_4, α_2 by α_5, while α_3 both by α_4 and α_5.* □

Exploiting the above definitions we obtain condensed patterns under constraints.

Definition 9 (Condensed pattern under constraints)**.** *Let p be a pattern from $\mathscr{L}$, 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.*

Example 9. *For the constraint function selecting maximal itemsets of size at most 2 and size at least 2, α_4 and α_5 from Ex. 1 are condensed patterns.* □

2.3 Answer Set Programming

Answer Set Programming (ASP) (Gelfond and Lifschitz 1988) is a declarative problem solving paradigm oriented towards difficult search problems. ASP has its roots in Logic Programming and Nonmonotonic Reasoning. An *ASP program* Π is a set of rules of the form

```
a_0 :- b_1, ..., b_k, not b_k+1, ..., not b_m,                (2)
```

where $1 \leq k \leq m$, and `a_0, b_1, ..., b_m` are classical literals, and *not* is default negation. The right-hand side of r is its body, $Body(r)$, while the left-hand side is the head, $Head(r)$. $Body^+(r)$ and $Body^-(r)$ stand for the positive and negative parts of $Body(r)$ respectively. A rule r of the form (2) is a *fact* if $m = 0$. We omit the symbol `:-` when referring to facts. A rule without head literals is a *constraint*. Moreover, 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 subset-minimal Herbrand interpretation (*model*) of a ground positive program Π.

The semantics of an ASP program is given in terms of its answer sets. An interpretation A of Π is an *answer set* (or *stable model*) of Π iff $A = MM(\Pi^A)$, where Π^A is the *Gelfond–Lifschitz (GL) reduct* (Gelfond and Lifschitz 1988) of Π, obtained from $Gr(\Pi)$ by removing (i) each rule r such that $Body^-(r) \cap A \neq \emptyset$, and (ii) all the negative atoms from the remaining rules. The set of answer sets of a program Π is denoted by $AS(\Pi)$.

Example 10. *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(I,J):-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` with `1,2` and `V` with `b` resp.

The GL-reduct $\Pi^{A'}(\Pi)$ for the interpretation A' containing facts of Π and `not_subset(1,2)` differs from $Gr(\Pi)$ only in that `not item(2,b)` is not in the body of the rule. A' is the minimal model of $\Pi^{A'}(\Pi)$, and thus it is in $AS(\Pi)$. □

Other relevant language constructs include conditional literals and cardinality constraints (Simons et al. 2002). The former are of the form `a:b_1,...,b_m`, the latter can be written as `l{c_1,...,c_n}t`, where `a` and `b_i` are possibly default negated literals and each `c_j` is a conditional literal; `l` 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 instances of `a(X)` (subject to `b(X)`) are true. Furthermore, aggregates are of the form `#sum{K:cost(I,K)}>N`. This atom is true, whenever the sum of all `K`, such that `cost(I,K)` is true, exceeds `N`.

3 Hybrid ASP-based Mining Approach

In this section we present our hybrid method for 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. Here, we mainly focus on itemsets, sequence and graph mining.

Given a frequency threshold σ, a (sequential) dataset D and a set of constraints $\mathscr{C} = \mathscr{C}_l \cup \mathscr{C}_g$, where $\mathscr{C}_l$ and $\mathscr{C}_g$ 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. We use Eclat (Zaki et al. 1997) for itemsets, PPIC (Aoga et al. 2016) for sequences and gSpan (Yan and Han 2002) for graphs.

Step 2. Second, the computed patterns are post-processed using the declarative means to select a set of *valid* patterns (i.e., those satisfying constraints in $\mathscr{C}_l$). 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. Analogously, for graphs we have `graph(i,j,k)`, where `i` is the pattern's ID, `j` and `k` are node labels and the intentional meaning of this fact is that in the graph `i` the node `j` is connected to `k`. The local constraints in $\mathscr{C}_l$ are represented as ASP rules, which collect IDs of patterns satisfying constraints from $\mathscr{C}_l$ into the dedicated predicate `valid`, while the rest of the IDs are put 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 $\mathscr{C}_g$. Following the principle of (Järvisalo 2011), for itemsets and sequences in our work every answer set represents a single desired pattern, which satisfies both local and global constraints (for graphs slight variations apply, as we discuss later in this section). The set of all such patterns forms a condensed representation. In what follows we present 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 given in Def. 7. For every constraint in $\mathscr{C}_l$ we have a set of dedicated rules, stating

```
time(1..5).
% people born in Germany or France are Europeans
eu(I) :- seq(I,bG,P).
eu(I) :- seq(I,bF,P).
% collect those who moved to France before P
moved_before(X,P) :- seq(X,mF,P1), P > P1, time(P), time(P1).
% collect those who moved to France after P and before masters
moved_after(X,P) :- seq(X,mF,P1), seq(X,ma,P2), P < P1,
                    P1 < P2, time(P), time(P1), time(P2).
% keep Europeans who moved to Germany straight before masters
keep(X) :- seq(X,ma,P+1), seq(X,mG,P), eu(X).
% keep Germans who did not move before masters
keep(X) :- seq(X,bG,P1), seq(X,ma,P), not moved_before(X,P).
% keep Europeans whose last move before masters was to Germany
keep(X) :- seq(X,mG,P1), seq(X,ma,P2), P1 < P2,
           eu(X), not moved_after(X,P1).
% a pattern is not valid, if it should not be kept
not_valid(X) :- pattern(X), not keep(X)

```

Listing 1: Moving habits of people during studies

when a pattern is not valid. For instance, a constraint checking whether the cost of items in a pattern exceeds a given threshold N is encoded as

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

A similar rule for sequences can be defined as follows:

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

Analogously, one can specify arbitrary domain constraints on patterns.

Example 11. *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* 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, respectively. Suppose, we are only interested in moving habits of Europeans, who got their 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 mF in between or mG should precede ma. These constraints are encoded in the program in List. 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 $\mathscr{C}_l$ we add to a 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 analyzed to construct a condensed representation based on global constraints from $\mathscr{C}_g$.

```
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 a subset of J
5 dominated :- selected(I), valid(J),
6              I != J, not not_subset(J).
```

Listing 2: Maximal itemsets encoding

3.2 Encoding Global Constraints

The key for encoding global constraints is the declarative formalization of the dominance relation (Def. 8). 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 $\mathscr{L}$ and a constraint from $\mathscr{C}_g$ 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.
```

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 the `dominated/0` predicate.

Itemset Mining. We first provide an encoding for maximal itemset mining in List 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. 2 is required. An itemset is *closed* if none of its supersets has the same support. Thus to both of the rules from List. 2 we need to add atoms `support(I,X)`, `support(J,X)`, which store the support sets of `I` and `J` respectively (extracted from the output of Step 1).

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

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

Finally, the skyline itemset/sequence encoding is given in List. 3, where the first two rules specify the conditions (a) and (b) for skyline itemsets as specified in 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 embedding e exists, such that $S \sqsubseteq_e S'$.

In List. 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

```
 1 % support and size comparison among patterns
 2 g_size_geq_fr(J) :- selected(I), valid(J), support(I,X),
 3                     support(J,Y), size(I,Si), size(J,Sj),
 4                     Si <  Sj, X <= Y.
 5 geq_size_g_fr(J) :- selected(I), valid(J), support(I,X),
 6                     support(J,Y), size(I,Si), size(J,Sj),
 7                     Si <= Sj, X < Y.
 8 % derivation of the domination condition
 9 dominated :- valid(J), g_size_geq_fr(J).
10 dominated :- valid(J), geq_size_g_fr(J).
```

Listing 3: Skyline itemsets encoding

```
 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 4: Maximal sequence encoding

superpatterns of a selected sequence. First, J is not a superpattern of I if I is not a subset of J (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 neighbor 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. 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).
```

```
% J is not a subgraph of I if I has an edge (X,Y) that J does not have
not_subgraph(J) :- selected(I), edge(I,X,Y), valid(J),
                   not edge(J,X,Y), not edge(J,Y,X), I != J.
% derive dominated, whenever I is a subgraph of J
dominated :- selected(I), valid(J), not not_subgraph(J), I != J.
```

Listing 5: Maximal graph encoding (unique labeled)

Finally, the encoding for mining skyline sequences coincides with the skyline itemsets encoding, which is provided in List. 3.

Graph Mining. Graphs represent the most complex pattern type. The subpattern relation between graphs amounts to testing subgraph isomorphism between them. In general, this problem is NP-complete; however, for some restricted graph types, it is solvable in polynomial time. One of such graph types are undirected graphs, where every node has a unique label (and consequently, every edge (u,v) is labelled uniquely as $(l(u),l(v))$). For instance, the first two graphs in Fig. 1 fall into this category, while the third one does not. For this restricted graph type, we have that G is subgraph isomorphic to G' iff every edge in G is also present in G'. Therefore, the subgraph isomorphism test for these restricted graphs essentially boils down to subpattern test for itemsets (Neumann and Miettinen 2017).

We treat every edge in a graph as an item. Since the graph is allowed to have only unique labels, there cannot be any repetitions of the edge labels. The encoding of maximal frequent graph patterns is presented in List. 5. A selected graph pattern is not maximal if at least one of its frequent supergraphs is valid. Similar to the case of itemsets and sequences we rule out the patterns that are guaranteed to be not maximal. A pattern `J` is not a superpattern of `I` if `I` contains an edge that `J` does not have.

The encoding for closed frequent graph patterns differs from the one for maximal graph patterns only in that in the second rule in Fig. 5 the atoms `support(I,X), support(J,X)` are added. Encodings for other condensed representations are analogous.

Example 12. *The edges of the graph G_1 in Fig. 1 are represented with the following facts `edge(g1,e,c), edge(g1,c,a)`, etc. Given the unique-labeled graphs G_1 and G_2 and the frequency threshold $\sigma = 2$, we have that both $\{$`(a,b)`$\}$ and $\{$`(e,c), (c,a), (a,b)`$\}$ are frequent subgraphs. However, only the latter graph pattern is maximal.* □

The encoding for maximal (closed, etc) graph mining problem in the general case is slightly more complicated. For the maximal constraint we depict the encoding in List. 6 (the rest of the constraints are treated analogously). Since the subgraph isomorphism check is an NP-complete problem, after obtaining a set F of frequent candidate graph patterns using a dedicated algorithm, we perform a dominance check using the ASP program in List. 6 for each pattern separately (unlike in earlier presented encodings, where such a check was done for all frequent patterns jointly within a single ASP program).

More specifically, the solver receives as input a selected graph candidate G and the rest of the frequent patterns in F excluding G. The graph G is represented using two predicates: `selected_node(v,lv)` reflecting labelled vertices of G and `selected_edge(v,w,le)`, where `v,w` are nodes and the edge `(v,w)` is labeled with `le` in G. First, in (2) of List. 6 the guess is performed on a graph in the set F of frequent patterns, to which G can be mapped, i.e., a dominating candidate pattern is guessed. Second, in (4) the guess on a mapping from G to the

```
1 % pick one graph to map to
2 1 { mapped_to(X) : graph(X)} 1.
3 % guess the mapping to the selected graph
4 1 { map(X,N) : node(_,N,_) } 1 :- selected_node(X,_).
5 % check mapping validity on the edges with labels
6 :- mapped_to(I), map(X1,V1), map(X2,V2),
7    selected_edge(X1,X2,L),  not edge(I,V1,V2,L).
8 % check validity on the nodes
9 :- mapped_to(I), map(X,V), selected_node(X,L), not node(I,V,L).
```

Listing 6: Maximal graph encoding (core); general case

dominating candidate graph is done. Finally, in (6)-(9) the mapping is validated by two integrity constraints. If the solver returns an answer set, then the selected graph G is removed from the dataset as a dominated pattern, i.e., it is guaranteed to be not maximal.

4 Covering with Approximate Patterns

So far, we have concentrated on exact patterns, that is, patterns that are present exactly in the data; for example, an itemset is present only in transactions that contain every item in the itemset. This is a standard assumption in pattern mining, and it is heavily utilized by the pattern mining algorithms. But it also means that the patterns are not robust against noise: a single item missing from a single transaction can turn a frequent itemset into an infrequent one. To make the patterns more robust to noise, we can consider approximate patterns, that is, patterns that are contained in transactions even if the transaction does not contain all of the items in the pattern. In this section, we demonstrate how our hybrid approach can be used also with approximate patterns.

The approximate patterns we are interested in are based on the concept of *tiles* (Geerts et al. 2004). Let α be an itemset and D be a database. The *tile* corresponding to α is defined as $\tau(\alpha, D) = \{(t_{id}, i) \mid t_{id} \in D_\alpha, i \in I\}$. When D is clear from the context, we write $\tau(\alpha)$. The area of $\tau(\alpha)$ is equal to its cardinality.

Example 13. *The tile corresponding to the itemset $\{b,e\}$ from Tab. 2 is $\{(1,b),(2,b),(1,e),(2,e)\}$.*

A *tiling* $T = \{\tau(\alpha_1), \ldots, \tau(\alpha_k)\}$ consists of a finite number of tiles. Its area is $area(T,D) = |\tau(\alpha_1) \cup, \ldots, \cup \tau(\alpha_k)|$. In *Maximum k-Tiling* the goal is to find a tiling of k tiles with the maximum area, whereas in the *Minimum Tiling* the goal is to find the least number of tiles such that every (t_{id}, i) pair is included in at least one tile (Geerts et al. 2004).

Another way to define tiles is via binary matrices. A transaction database D can be regarded as a binary matrix $\mathbf{D}$ of size $m \times n$, where m is the number of transactions in D and n is the number of items in I. The (i, j)th element in the binary matrix corresponding to D is equal to 1 if the ith transaction contains item j, and it is equal to 0 otherwise. Hence, the size of D is equal to the number of ones in this matrix. In this framework, a tile is a *rank-1* matrix $\mathbf{T}$ that is *dominated* by the data matrix $\mathbf{D}$. Matrix $\mathbf{T}$ is rank-1 if it is an outer product of two binary vectors, that is, $\mathbf{T} = \mathbf{u}\mathbf{v}^T$ where $\mathbf{u} \in \{0,1\}^m$ and $\mathbf{v} \in \{0,1\}^n$. Matrix $\mathbf{T} = (t_{ij})$ is dominated by matrix $\mathbf{D} = (d_{ij})$ if $t_{ij} \leq d_{ij}$ for all i and j.

Example 14. *The transaction database in Tab. 2 corresponds to the following binary matrix:*

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} ,$$

and the tile from Example 13 corresponds to the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \end{pmatrix} .$$

□

A tiling is an element-wise logical disjunction of the matrices corresponding to the tiles,

$$\mathbf{S} = \mathbf{T}_1 \vee \mathbf{T}_2 \vee \cdots \vee \mathbf{T}_k .$$

Assuming that $\mathbf{T}_i = \mathbf{u}_i \mathbf{v}_i^T$, we can write each element of $\mathbf{S} = (s_{ij})$ as

$$s_{ij} = \bigvee_{\ell=1}^{k} u_{i\ell} v_{j\ell} ,$$

that is, $\mathbf{S}$ is the *Boolean matrix product* of matrices $\mathbf{U} = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_k]$ and $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_k]$ (see Miettinen (2009) for more discussion on the connections between pattern mining and Boolean matrix factorization).

The formulation of tiles as rank-1 binary matrices immediately suggests the concept of *approximate tiles* as non-dominated rank-1 matrices and *approximate tiling* as approximate Boolean matrix factorization. We will call both exact and approximate tiles simply as tiles from now on.

Example 15. *Consider the following boolean matrix:*

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

The areas with the red, blue and black border highlight the respective three approximate tiles of the matrix. □

A common approach to calculate an approximate Boolean matrix factorization (or tiling) is to first calculate a set of *candidate tiles* (or rank-1 matrices) and then select the final set from these (Miettinen et al. 2008; Tyukin et al. 2014). In what follows, we concentrate on a version of this problem where the user gives the target quality, and we try to find a set of tiles that obtains this quality.

Definition 10 (Approximate Tile Selection Problem)**.** *Given a binary matrix $\mathbf{D} \in \{0,1\}^{m \times n}$, a set T of tiles (rank-1 matrices) and an integer threshold σ, find a subset $T' = \{\mathbf{T}^{(1)}, \mathbf{T}^{(2)}, \ldots, \mathbf{T}^{(k)}\} \subseteq T$, such that when we write $\mathbf{S} = (s_{ij})$ with $s_{ij} = t_{ij}^{(1)} \vee t_{ij}^{(2)} \vee \cdots \vee t_{ij}^{(k)}$, the error*

$$\left|\{(i,j) \mid d_{ij} = 1 \wedge s_{ij} = 0\}\right| + \left|\{(i,j) \mid d_{ij} = 0 \wedge s_{ij} = 1\}\right| \tag{3}$$

(i.e., the number of ones outside of the tiling plus the number of zeros inside the tiling) is smaller then the threshold σ.

```
% for every tile store its ID and coordinates of 1s that it encodes
tileid(X) :- tile(X,Y,Z).
one(Y,Z)  :- tile(X,Y,Z).
% for every tile guess whether to include it into the tiling
1 { intile(X) : tileid(X) }.
% collect and count ones that are outside the constructed tiling
incell(Y,Z) :- intiling(X), tile(X,Y,Z).
outsideone(Y,Z) :- tile(X,Y,Z), not incell(Y,Z).
numberofoutsideones(N) :- N = #count{Y, Z : outsideone(Y,Z)}.
% collect and count zeros that are inside the constructed tiling
insidezero(Y,Z) :- intiling(X),       relitem(X,Y),
                   reltrans(X,Z), not tile(X,Y,Z).
numberofinsidezeros(N) :- N = #count{Y, Z : insidezero(Y,Z)}.
% compute the total error
er(0,K) :- numberofzerosinside(K).
er(1,K) :- numberofonesoutside(K).
totalerror(N) :- N = #sum{X, Y : er(Y,X)}.
% ensure that the total error does not exceed the threshold
:- totalerror(N), threshold(K), N > K.
```

Listing 7: Encoding for the approximate tiling problem from Def. 10

It is worth noticing that (3) corresponds to the Hamming distance between **D** and **S**. The decision version of the Approximate Tile Selection problem is NP-hard (Miettinen 2015), and the optimization version (finding the smallest possible error) is NP-hard to approximate to within $\Omega(2^{\log^{1-\varepsilon}|\mathbf{D}|})$ for any $\varepsilon > 0$, where $|\mathbf{D}|$ is the number of 1s in **D** (Miettinen 2015). The problem has two alternative characterizations, either as a variant of the famous Set Cover problem called *Positive-Negative Partial Set Cover* (Miettinen 2008) or as a form of *Boolean linear programming*: given a binary design matrix **A**, and a binary target vector **b**, find a binary vector **x** that minimizes the Hamming distance between **b** and **Ax**, where the matrix-vector product is over Boolean algebra.

For computing a set of candidate tiles T to choose from, effective implementations exist (see, e.g., Tyukin et al. (2014)). From these candidates a subset needs to be selected, which will be a solution to the problem from Def. 10. In Fig. 7 we provide an ASP program, whose answer sets exactly correspond to solutions to the above problem.

In the input the ASP program gets a set of tile candidates to choose from. They all are of the form `tile(id,i,t)`, where `id` is a unique ID of the given tile, while `i` and `t` are the IDs of the items and transactions respectively, such that the given tile has 1 in the intersection of the column `i` and the row `t`. Along with that, we are also provided the facts `relitem(id,i)` and `reltrans(id,t)`, which respectively encode the relevant items and transactions for the tile `id`.

Example 16. *Provided that the red tile from Ex. 15 has ID 1, its encoding is given by the following facts: `tile(1,1,1)`, `tile(1,2,1)`, `tile(1,1,2)`, `tile(1,1,3)`, `relitem(1..2)`, `reltrans(1..3)`. Observe that it is essential to specify relevant items and transactions, since without them different ways of interpreting a tile exist. For instance, the facts `tile(1,1,1)`, `tile(1,4,1)`, `tile(1,1,3)`, `tile(1,4,3)` can correspond to several tiles including:*

$$\begin{pmatrix} \boxed{1\ 0\ 0\ 1} \\ 0\ 0\ 0\ 0 \\ \boxed{1\ 0\ 0\ 1} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

□

In (1)–(3) of List. 7 we collect tiles' IDs and locations of ones using the dedicated predicates `tileid` and `one` respectively. Than in (4)–(5) for every tile we perform a guess on whether to include it into the tiling. In lines (6)–(9), we count the number of 1s that are outside of the tiling, while in lines (10)–(13), the number of 0s inside the tiling is computed. Finally, the total error is determined in lines (14)–(17) and its admissibility based on the given threshold is checked in lines (18)–(19).

This ASP program can be exploited to solve Boolean matrix factorization, when used together with existing candidate generation approaches, or to solve Boolean linear programming instances. In fact, in Boolean linear programming, the design matrix **A** is given as a part of the problem instance, hence for that even a purely declarative programming solution will suffice.

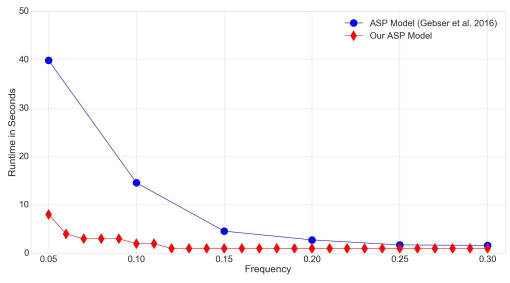
5 Evaluation

In this section we evaluate the proposed hybrid approach by comparing it to the existing declarative pattern mining methods: ASP model for sequences (Gebser et al. 2016) and Dominance Programming (DP) (Négrevergne et al. 2013). We do not consider the itemset mining ASP model (Järvisalo 2011), since it focuses only on frequent itemset mining and is not applicable to the construction of condensed representations under constraints as explained and addressed in (Négrevergne et al. 2013). Moreover, we do not perform comparison with dedicated algorithms designed for a specific problem type; these are known to be more efficient than declarative mining approaches (Négrevergne and Guns 2015), yet obviously less flexible.

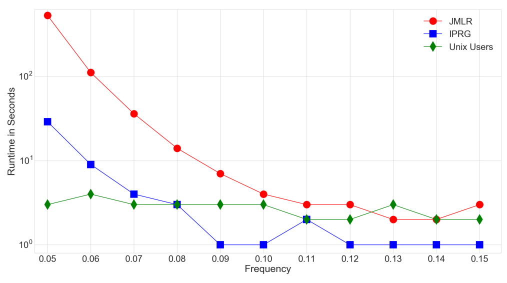
More specifically, we investigate the following experimental questions.

- $\mathbf{Q_1}$: How does the runtime of our method compare to the existing ASP-based sequence mining models?
- $\mathbf{Q_2}$: What is the runtime gap between the specialized mining languages such as dominance programming and our method?
- $\mathbf{Q_3}$: What is the influence of local constraints on the runtime of our method?
- $\mathbf{Q_4}$: How does the choice of an ASP solver influence the overall performance of the system?
- $\mathbf{Q_5}$: How does the hybrid system perform on highly structured graph mining task in comparison to other logic-based mining systems?
- $\mathbf{Q_6}$: How does the hybrid method perform on the approximate tiling problem?

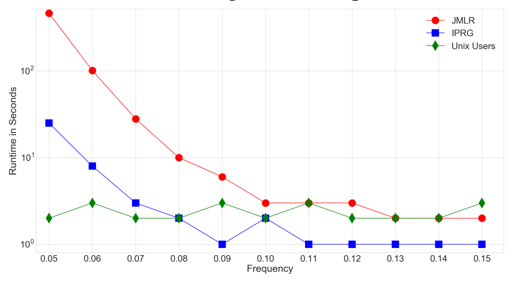
For $\mathbf{Q_1}$ we compare our work with the ASP-based model by Gebser et al. (2016). For $\mathbf{Q_2}$ we measure the runtime difference between specialized itemset mining languages (Négrevergne et al. 2013) and our ASP-based model. To address $\mathbf{Q_3}$ we estimate the runtime effect of adding local constraints. Moreover, for $\mathbf{Q_4}$ we compare the performance of our system with two state-of-the-art ASP solvers: Clasp and WASP. Regarding $\mathbf{Q_5}$, we perform the comparison of our hybrid mining system against the existing ILP-based method for graph mining. Finally, to tackle $\mathbf{Q_6}$ as a proof of concept we test the effectiveness of our hybrid method for solving the problem from Def. 10.



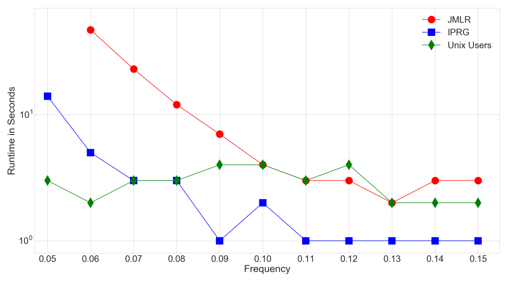
(a) Comparing with ASP sequence model (Gebser et al. 2016) on the 200 generated sequences (closed)



(b) Maximal sequence patterns



(c) Closed sequence patterns



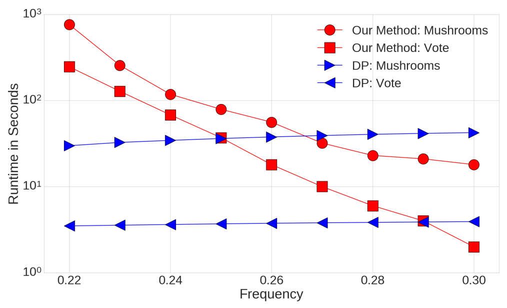
(d) Skyline sequence patterns

Figure 3: Investigating $\mathbf{Q_1}$: comparison with pure ASP model (3a) and maximal (3b), closed (3c), and skyline (3d) sequence mining on JMLR, Unix Users, and iPRG datasets.

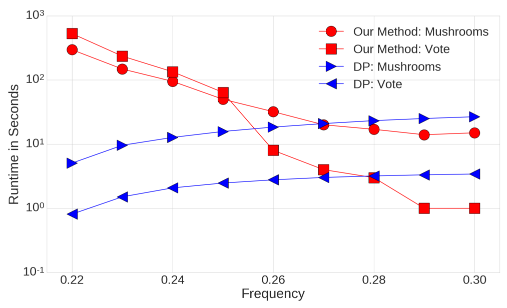
We report our evaluation results on 2 transaction datasets[2]: *Mushrooms* (8124 transactions/119 items) and *Vote* (435/48), 3 sequence datasets (full)[3]: *JMLR* (788 sequences/3847 distinct symbols), *Unix Users* (9099/4093), and *iPRG* (8628/21), 3 graph datasets[4]: *Yoshida* (265 graphs/20 avg. vertices/23 avg. edges/9 distinct labels), *Nctrer* (232/19/20/9) and *Bloodbarr*(413/21/23/9), and 2 datasets for the tiling problem: *Divorce*[5] (50 transactions/9 items) and *Glass*[6] (214/48). 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[7] C++14 for the itemset/sequence wrapper and python 2.7 for the graph wrapper. In the evaluation we used the latest available version 2 of WASP[8]. The timeout was set to one hour. Free pattern mining demonstrates the same runtime behavior as closed, due to the symmetric encoding, and is thus omitted.

To investigate $\mathbf{Q_1}$, in Fig. 3a, we compare the ASP model (Gebser et al. 2016) with our method on the default 200 sequence sample, generated by the tool[9] of Gebser et al. (2016). We performed the comparison on the synthetic data, as the sequence-mining model (Gebser et al. 2016) failed to compute condensed representations on any of the standard sequence datasets for any support threshold value within the timeout. One can observe that our method consistently outperforms the

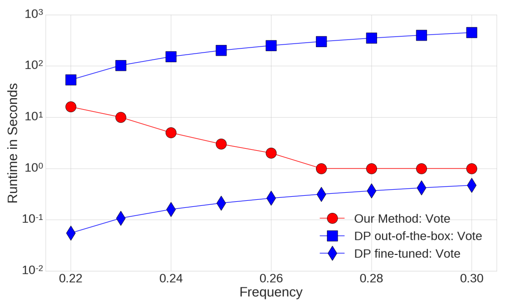
[2] From `https://dtai.cs.kuleuven.be/CP4IM/datasets/`.
[3] From `https://dtai.cs.kuleuven.be/CP4IM/cpsm/datasets.html`.
[4] From `https://github.com/amaunz/ofsdata`
[5] `https://sparse.tamu.edu/Pajek/divorce`
[6] `http://cgi.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html`
[7] `http://potassco.sourceforge.net`
[8] `https://github.com/alviano/wasp`
[9] `https://sites.google.com/site/aspseqmining`



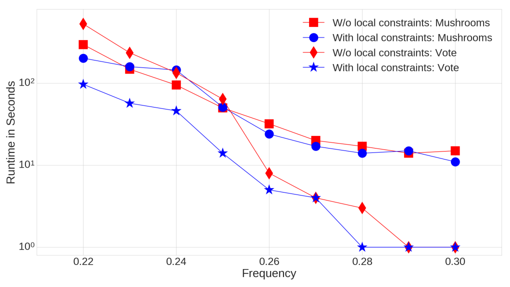
(a) Maximal itemset mining: comparing with DP on Vote and Mushrooms



(b) Closed itemset mining: comparing with DP on Vote and Mushrooms



(c) Skyline itemset mining: comparing with out-of-the-box and fine-tuned DP on Vote



(d) Closed itemset mining: our method with (w/o) local constraints on Vote and Mushrooms

purely declarative approach of Gebser et al. (2016) and the advantage naturally becomes more apparent for smaller frequency threshold values.

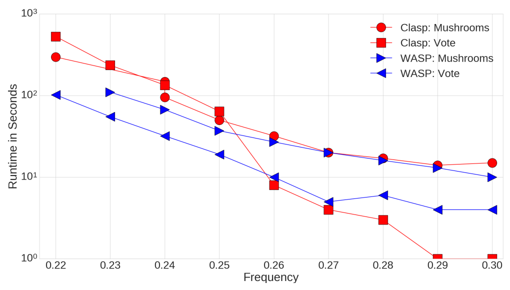
In Figs. 3b, 3c and 3d (the point 0.05 for JMLR is a timeout), we present the runtimes of our method for *maximal, closed* and *skyline* sequential pattern mining settings on JMRL, Unix Users and iPRG datasets. In contrast to Gebser et al. (2016), our method managed to produce results on all of these datasets for reasonable threshold values within a couple of minutes.

To investigate $\mathbf{Q_2}$, we compare out-of-the-box performance of DP (Négrevergne et al. 2013) with our approach on maximal, closed and skyline itemset mining problems using standard datasets Vote and Mushrooms. As we see in Figs. 4a and 4b, on average, DP is one-to-two orders of magnitude faster; this gap is, however, diminishing as the minimum frequency increases. Surprisingly, our approach is significantly faster than DP out-of-the-box for skyline patterns (Fig. 4c); this holds also for the Mushrooms dataset, not presented here.

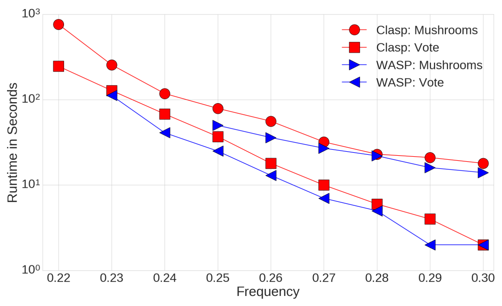
Fine-tuning parameters of DP by changing the order in which operators are applied within the system (skyline+ option) allowed to close this gap. With this adaptation DP demonstrates one-to-two orders of magnitude better performance, as can be seen in Fig. 4c. However, fine-tuning such a system requires the understanding of its inner mechanisms or exhaustive application of 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.

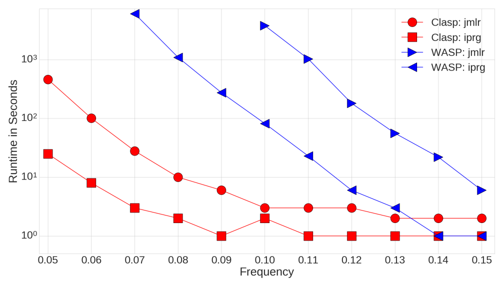
In Fig. 4d, we present the results for closed itemset mining with and without local constraints (experiments with other global constraints demonstrate a similar runtime pattern). Local constraints ensure better propagation and speed up the search. One of the key design features of our encoding



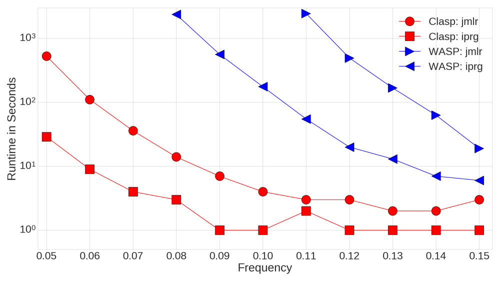
(a) Itemset mining: closed patterns



(b) Itemset mining: maximal patterns



(c) Sequence mining: closed patterns



(d) Sequence mining: maximal patterns

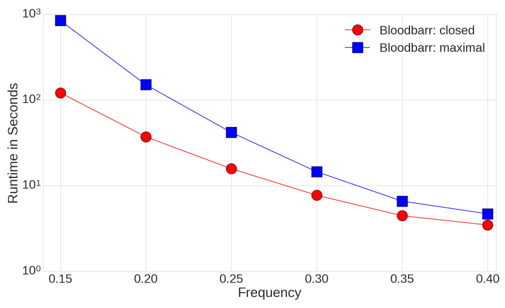
Figure 5: $\mathbf{Q_4}$: comparing ASP solvers performance – WASP and Clasp on itemset and sequence mining tasks

is the filtering technique used to select candidate patterns among only valid patterns. Its effect can be clearly seen, e.g., for the Vote dataset in Fig. 4d, where for certain frequencies the runtime gap is close to an order of magnitude.

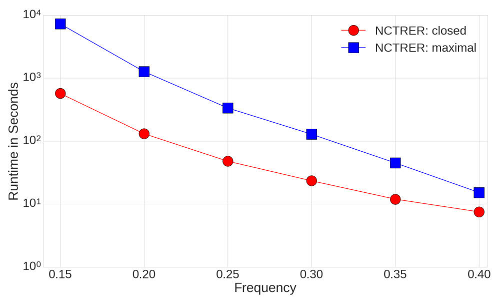
To analyze $\mathbf{Q_4}$, we have replaced Clasp solver with WASP (Alviano et al. 2013) in our system. As we see in Fig. 5a, for mining closed itemsets two systems perform on par except for a timeout of WASP on mushrooms with frequency 0.22. However, we already can observe significant difference in performance on maximal patterns for both itemsets and sequences. A similar behavior can be seen in the setting of closed sequences mining: there is at least an order of magnitude difference in performance. The runtime gap cannot be explained by differences in grounding, since for both tasks Gringo has been used. However, we have noticed the difference in memory management: Clasp seems to be able to reason and keep track of significantly larger sets of patterns, while being economic and provident with memory. Since the task naturally allows one to generate problem instances of practically any size, we presume that our setting might be well-suited as one of the tests for ASP solvers' performance.

To address $\mathbf{Q_5}$, we consider the standard graph mining datasets in Figs. 6a, 6b and 6c. One can observe, that our system is indeed able to handle real-sized datasets and process hundreds or even thousands of graph patterns within the time-limit. Furthermore, the comparison with the purely declarative framework presented in Fig. 6d demonstrates the two orders of magnitude speedup in runtime. In addition, note that our system is able to enumerate all condensed graph patterns, while the purely declarative approach is not capable of doing that within the timeout.

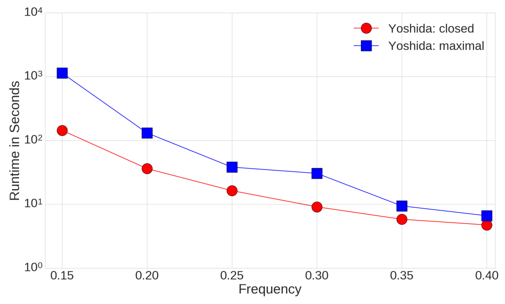
To investigate $\mathbf{Q_6}$, we tested our hybrid approach on the approximate tile selection problem from Def. 10, exploiting Clasp reasoner for the ASP part of the algorithm. Since the considered problem is very computationally demanding, we did not set any timeout in this experimental



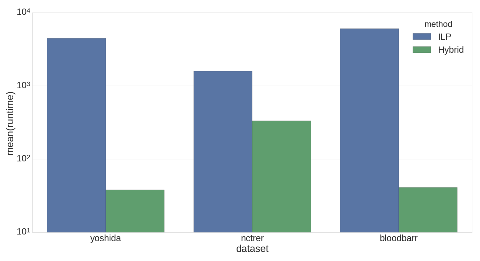
(a) Bloodbarr dataset: maximal and closed graph patterns



(b) Nctrer dataset: maximal and closed graph patterns



(c) Yoshida dataset: maximal and closed graph patterns



(d) Comparison with logic programming ILP framework (Paramonov et al. 2015). For the ILP framework time to find *a maximal solution* is indicated, while for our system time to enumerate all maximal graph patterns (frequency 0.2) is shown.

Figure 6: Investigating $\mathbf{Q_5}$: evaluation on standard graph datasets (a,b,c) and comparison to the declarative ILP system for graph mining (d). Log-scale on all plots

setting. In Tab. 4 and Tab. 5 we report the results for the Divorce and Glass datasets respectively. We compute the candidate tiles using an approach based on association confidences (see Miettinen et al. 2008). Given the set of candidate tiles, we use our ASP encoding from List. 7 to find all tilings whose overall error is below a specified threshold. We compare this approach to a greedy method for finding *any* tiling that has error below the threshold. The greedy tiling approach will add the candidate tiles one-by-one until it has either found an admissible tiling, or it has exhausted all tiles. Note that, since the problem of finding the tiling is NP-hard, the greedy method is not guaranteed to find an admissible tiling even if one exists.

The first column in Tab. 4 (respectively Tab. 5) reports the number n of candidate tiles and the error threshold value σ. In the second column we present the details of the solution found by the dedicated method, i.e., the number k of tiles in the solution tiling and its overall error. The computation of the tilings by the greedy algorithm takes less than a second. In the third column, the results of our hybrid approach are provided. More specifically, we present the number k of tiles and the overall error for the first solution found by the solver together with the running time, as well as the details of the optimal tiling. To compute the optimal tiling we enforce the solver to find all models of the given ASP program (their total number is likewise reported).

The candidate tiles are computed by the dedicated algorithm within a fraction of a second, and the major computational efforts are done by the ASP solver to find the final tiling. Moreover,

n/σ	native		hybrid						
	k	error	first			k opt	error opt	time all	|all solutions|
			k	error	time				
3/59	1	51	1	59	38.961	1	51	38.962	5
4/54	1	51	2	51	38.127	1	51	38.135	8
5/60	2	51	2	58	50.616	2	47	50.617	15
6/60	2	51	2	58	50.601	2	47	50.600	30

Table 4: Addressing $\mathbf{Q_6}$: approximate tiling problem solution using specialized native algorithm for tile candidate generation and ASP encoding for selecting the resulting tiling (Divorce dataset)

n/σ	native		hybrid						
	k	error	first			k opt	error opt	time all	|all solutions|
			k	error	time				
3/214	1	184	1	196	294.172	3	88	294.173	7
4/168	3	142	2	154	1194.088	3	142	1194.093	2
5/222	3	180	3	192	2126.944	4	142	2126.948	11
6/224	3	182	3	194	2198.739	5	142	2198.743	25

Table 5: Addressing $\mathbf{Q_6}$: approximate tiling problem solution using specialized native algorithm for tile candidate generation and ASP encoding for selecting the resulting tiling (Glass dataset)

observe that since the grounding step is the most time-consuming, the time difference between the first found solution and all solutions including the optimal one is actually neglectable. Note that comparing the running time of the dedicated and our hybrid algorithms is not entirely fair, as the former is not complete (i.e., it might not find any tiling, satisfying the conditions imposed by the error threshold even if one exists), which is in contrast to our hybrid approach.

The native algorithm is capable of computing the optimal tiling only for small instances of the Divorce dataset. Starting from $n = 5$, our hybrid method outperforms the native one with respect to the quality of the found solutions. For the Glass dataset the benefit of our method is apparent even for smaller instances.

Summary. In all experiments, Step 1 of our method contributes to less than 5% of runtime. Overall, our approach 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 $\mathbf{Q_2}$. As demonstrated in $\mathbf{Q_3}$ various local constraints can be effectively incorporated into our encoding bringing additional performance benefits. The choice of an ASP solver plays a crucial role in the overall performance of the system, as discussed in $\mathbf{Q_4}$. In $\mathbf{Q_5}$, it has has been established that our approach leads to a significant speed up (of orders of magnitude) for the mining tasks with a complex structure, such as graph mining. Finally, the results of $\mathbf{Q_6}$ have proved the applicability of our hybrid framework for other computationally intensive data mining tasks, e.g., approximate tile selection problem, where our method is able to find solutions of higher quality.

6 Related Work

Pattern mining approaches, especially frequent (closed or maximal) itemset, sequence, and subgraph mining are amongst the foundational methods in data mining (see, e.g., Aggarwal (2015) for a recent textbook on the topic), and have been studied actively since mid-nineties (Agrawal et al. 1993; Agrawal et al. 1996). These problems are considered local and exhaustive, as the goal is always to enumerate all patterns that satisfy the (local) frequency constraint (together with some global constraints, such as closedness). In addition, the patterns have to be exact, that is, they have to be present in the data. The exactness was relaxed in later work (Pensa and Boulicaut 2005), while Geerts et al. (2004) considered the problem of summarizing the data using closed itemsests, that is, tiling. These two approaches, non-exact patterns and summarization using them, were combined by Miettinen et al. (2008) in their work on Boolean matrix factorization.

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., Pei and Han 2000). On the other hand, declarative methods based on Constraint Programming (Rojas et al. 2014; Négrevergne and Guns 2015; Métivier et al. 2013), SAT solving (Jabbour et al. 2015; Jabbour et al. 2013) and ASP (Järvisalo 2011; Gebser et al. 2016; Guyet et al. 2014) have been proposed.

Techniques from the last group are the closest to our work. However, in contrast to our method, they typically focus only on one particular pattern type and consider local constraints and condensed representations in isolation (Pei et al. 2000; Yan et al. 2003). Négrevergne et al. (2013) and Guns et al. (2017) focused on CP-based rather than ASP-based itemset mining and did not take into account sequences unlike we do. Gebser et al. (2016) studied declarative sequence mining with ASP, but in contrast to our approach, optimized algorithms for frequent pattern discovery are not exploited in their method. A theoretical framework for structured pattern mining was proposed by Guns et al. (2016), whose main goal was to formally define the core components of the main mining tasks and compare dedicated mining algorithms to their declarative versions. While generic, this work did not take into account local and global constraints and neither has it been implemented.

Järvisalo (2011) and Gebser et al. (2016) considered purely declarative ASP methods; unlike our approach, they do not admit integration of optimized mining algorithms and thus lack practicality. In fact, the need for such an integration in the context of complex structured mining was even explicitly stated by Paramonov et al. (2015) and by van der Hallen et al. (2016), which study formalizations of graph mining problems using logical means.

7 Conclusion

We have presented a hybrid approach for condensed itemset, sequence and graph 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 specialized methods and also integrated the dominance programming machinery in our implementation to allow for combining local and global constraints on a generic level. Moreover, using the example of the approximate tile selection problem, we have demonstrated that our hybrid method can be further generalized to other data mining tasks.

One of the possible future directions is to extend the proposed approach to an iterative technique, where dedicated data mining and declarative methods are interlinked and applied in an alternating fashion. More specifically, all constraints can be split into two parts: those that can be effectively handled using declarative means and those for which specialized algorithms are much more scalable. Answer set programs with external computations (Eiter et al. 2009) could be possibly exploited in this mining context.

Another promising but challenging research stream concerns the integration of data *decomposition* techniques into our approach. Here, one can divide a given dataset into several parts, such that the frequent patterns are identified in these parts separately, and then the results are effectively combined; such data decomposition is expected to yield further computational gains.

References

AGGARWAL, C. C. 2015. *Data Mining: The Textbook*. Springer, Cham.

AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. In *SIGMOD '93*. 207–216.

AGRAWAL, R., MANNILA, H., SRIKANT, R., TOIVONEN, H., AND VERKAMO, A. I. 1996. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 307–328.

ALVIANO, M., DODARO, C., FABER, W., LEONE, N., AND RICCA, F. 2013. Wasp: A native asp solver based on constraint learning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 54–66.

AOGA, J. O. R., GUNS, T., AND SCHAUS, P. 2016. An efficient algorithm for mining frequent sequence with constraint programming. In *ECML PKDD*. 315–330.

BONCHI, F. AND LUCCHESE, C. 2006. On condensed representations of constrained frequent patterns. *Knowl. Inf. Syst. 9,* 2, 180–201.

COOK, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, M. A. Harrison, R. B. Banerji, and J. D. Ullman, Eds. ACM, 151–158.

EITER, T., BREWKA, G., DAO-TRAN, M., FINK, M., IANNI, G., AND KRENNWALLNER, T. 2009. Combining nonmonotonic knowledge bases with external sources. In *Frontiers of Combining Systems, 7th International Symposium, FroCoS Italy, September 16-18*. 18–42.

GEBSER, M., GUYET, T., QUINIOU, R., ROMERO, J., AND SCHAUB, T. 2016. 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*.

GEERTS, F., GOETHALS, B., AND MIELIKÄINEN, T. 2004. Tiling databases. In *Discovery Science, 7th International Conference, DS 2004, Padova, Italy, October 2-5, 2004, Proceedings*. 278–289.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proc. of ICLP/SLP*. 1070–1080.

GUNS, T., DRIES, A., NIJSSEN, S., TACK, G., AND DE RAEDT, L. 2017. MiningZinc: A declarative framework for constraint-based mining. *Artif. Intell. 244*, 6–29.

GUNS, T., PARAMONOV, S., AND NÉGREVERGNE, B. 2016. On declarative modeling of structured pattern mining. In *Declarative Learning Based Programming, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016.*

GUYET, T., MOINARD, Y., AND QUINIOU, R. 2014. Using answer set programming for pattern mining. *CoRR abs/1409.7777.*

JABBOUR, S., SAIS, L., AND SALHI, Y. 2013. 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*. 649–658.

JABBOUR, S., SAIS, L., AND SALHI, Y. 2015. Decomposition based SAT encodings for itemset mining problems. In *PAKDD*. 662–674.

JÄRVISALO, M. 2011. 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*. 304–310.

MANNILA, H. AND TOIVONEN, H. 1997. Levelwise search and borders of theories in knowledge discovery. *Data Min. Knowl. Discov. 1,* 3, 241–258.

MÉTIVIER, J., LOUDNI, S., AND CHARNOIS, T. 2013. A constraint programming approach for mining sequential patterns in a sequence database. *CoRR abs/1311.6907*.

MIETTINEN, P. 2008. On the positive-negative partial set cover problem. *Inform. Process. Lett. 108,* 4, 219–221.

MIETTINEN, P. 2009. Matrix decomposition methods for data mining: Computational complexity and algorithms. Ph.D. thesis, Department of Computer Science, University of Helsinki.

MIETTINEN, P. 2015. Generalized matrix factorizations as a unifying framework for pattern set mining: Complexity beyond blocks. In *ECMLPKDD '15*. 36–52.

MIETTINEN, P., MIELIKÄINEN, T., GIONIS, A., DAS, G., AND MANNILA, H. 2008. The discrete basis problem. *IEEE Trans. Knowl. Data Eng. 20,* 10, 1348–1362.

NÉGREVERGNE, B., DRIES, A., GUNS, T., AND NIJSSEN, S. 2013. Dominance programming for itemset mining. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*. 557–566.

NÉGREVERGNE, B. AND GUNS, T. 2015. Constraint-based sequence mining using constraint programming. In *CPAIOR*. 288–305.

NEUMANN, S. AND MIETTINEN, P. 2017. Reductions for frequency-based data mining problems. *CoRR abs/1709.00900*.

PARAMONOV, S., VAN LEEUWEN, M., DENECKER, M., AND DE RAEDT, L. 2015. An exercise in declarative modeling for relational query mining. In *ILP*. 166–182.

PEI, J. AND HAN, J. 2000. Can we push more constraints into frequent pattern mining? In *ACM SIGKDD, Boston, MA, USA*. 350–354.

PEI, J., HAN, J., AND MAO, R. 2000. CLOSET: an efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, TX, USA*. 21–30.

PENSA, R. G. AND BOULICAUT, J.-F. 2005. Towards fault-tolerant formal concept analysis. In *AI*IA '05*. 212–223.

ROJAS, W. U., BOIZUMAULT, P., LOUDNI, S., CRÉMILLEUX, B., AND LEPAILLEUR, A. 2014. Mining (soft-) skypatterns using dynamic CSP. In *CPAIOR*. 71–87.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artif. Intell. 138,* 1-2, 181–234.

TYUKIN, A., KRAMER, S., AND WICKER, J. 2014. BMaD – A boolean matrix decomposition framework. In *ECML PKDD '14*, T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, Eds. 481–484.

VAN DER HALLEN, M., PARAMONOV, S., LEUSCHEL, M., AND JANSSENS, G. 2016. Knowledge representation analysis of graph mining. *CoRR abs/1608.08956*.

YAN, X. AND HAN, J. 2002. gspan: Graph-based substructure pattern mining. In *ICDM '02*.

YAN, X., HAN, J., AND AFSHAR, R. 2003. Clospan: Mining closed sequential patterns in large datasets. In *In SDM*. 166–177.

ZAKI, M. J., PARTHASARATHY, S., OGIHARA, M., AND LI, W. 1997. New algorithms for fast discovery of association rules. Tech. rep., Rochester, NY, USA.