TLP **19** (4): 505–535, 2019. © Cambridge University Press 2019
doi:10.1017/S1471068418000467 First published online 18 January 2019

*Hybrid ASP-based Approach to Pattern Mining**

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

DARIA STEPANOVA and PAULI MIETTINEN
Max Planck Institute for Informatics, Saarbrücken, Germany
(*e-mails:* dstepano@mpi-inf.mpg.de, pmiettin@mpi-inf.mpg.de)

submitted 29 November 2017; revised 23 August 2018; accepted 24 August 2018

Abstract

Detecting small sets of relevant patterns from a given data set 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 focus either on scalability or on generality. In this paper, we make steps toward combining local (frequency, size, and cost) and global (various condensed representations like maximal, closed, and 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 data sets show the effectiveness of the proposed method and computational gains for itemset, sequence, and graph mining, as well as approximate tiling.

Under consideration in Theory and Practice of Logic Programming.

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

* This work has been supported by the FWO and by the ERC-ADG-201 project 694980 SYNTH funded by the European Research Council. This is an extended version of a paper presented at the RuleML+RR 2017 conference, which has been invited for submission to TPLP. The authors acknowledge the assistance of the RuleML+RR 2017 Program Chairs Stefania Costantini, Enrico Franconi, Fariba Sadri and William van Woensel.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

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) type 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; Eiter *et al.* 2009b) has been proven to be well suited for defining the constraints conveniently [see, e.g. Järvisalo (2011), Gebser *et al.* (2016), and Guyet *et al.* (2014) for existing approaches on ASP-based frequent pattern mining] thanks to its expressive and intuitive modeling language and the availability of optimized ASP solvers such as Clasp (Gebser *et al.* 2007) and WASP (Alviano *et al.* 2013).

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. The existing methods have focused either on scalability of solving individual mining problems 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 on ASP-based sequential pattern mining (Gebser *et al.* 2016), 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 [existing solvers cannot yet take the full advantage of the stucture of the problem (van der Hallen *et al.* 2016), as specialized algorithms do to scale up and work with large data sets]. 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 Table 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.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Table 1. *Feature comparison between various ASP mining models and dominance programming*

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	–	–	–	✓

Notes: –: not designed for this datatype; ✓*: only maximal is supported.

Contributions. The goal of our work is to make steps toward building a generic framework that supports mining of condensed patterns, which (1) effectively combines dedicated algorithms and declarative means for pattern mining and (2) is easily extendable to incorporation of various constraints. We propose a two-step approach. In the first step, optimized algorithms are applied to discover a set of frequent patterns, and in the second step, the patterns are post-processed using declarative means. The key advantage of our approach stems from the fact that it preserves the generality of purely declarative methods with respect to the frequent pattern mining problems of the specified types, while providing an efficient system to develop prototypes which can run on real-world data sets where typically only specialized algorithms are deployed. This is especially beneficial in the setting where a user considers a new variation of a pattern mining problem and needs to prototype a system to run on the standard real-world pattern mining data sets. Typically, even standard pattern mining data sets are too large for purely declarative systems, and researchers have to experiment with the smallest data sets available [see, e.g. experimental sections of Gebser *et al.* (2016), Négrevergne *et al.* (2013), Négrevergne and Guns (2015), Guns *et al.* (2013), and Paramonov *et al.* (2015)]; contrary to this, developers of specialized algorithms, practically, have to rewrite the algorithms almost completely to model and solve new variations of a problem [e.g. see separate algorithms and papers for gSpan (Yan and Han 2002) and cloSpan (Yan *et al.* 2003)]. Our approach provides a middle ground between them, on the one hand, a researcher can model a new problem variation without changing the whole model and, on the other hand, she can experiment with the real-world data sets, which indeed makes the declarative approach more practical and appealing for applications.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

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 between different ASP mining models and dominance programming (a generic itemset mining language and solver).
- We demonstrate the feasibility of our approach with an experimental evaluation across multiple itemset, sequence, and graph data sets using state-of-the-art ASP solvers.

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

2 Preliminaries

In this section, we briefly recap the necessary background from the fields of both pattern mining and ASP.

Let D be a data set, $\mathcal{L}$ 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 $Th(\mathcal{L}, D, q) = \{\phi \in \mathcal{L} \mid q(D, \phi)\text{is true}\}$, that is, to find all patterns $\phi \in \mathcal{L}$ 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 $\mathcal{L}$. 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 $\mathcal{I}$ be a set of items $\{o_1, o_2, \ldots, o_n\}$. A nonempty subset of $\mathcal{I}$ is called an *itemset*. A *transaction data set* D is a collection of itemsets, $D = \{t_1, \ldots, t_m\}$, where $t_i \subseteq \mathcal{I}$. 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 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 data set 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.

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

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Table 2. *Transaction database*

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

Table 3. *Sequence database*

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

Example 1
Consider a transaction data set D from Table 2. We have $\mathcal{I} = \{a, b, c, d, e\}$ and $|D| = 3$. For $\sigma = 2$, the following itemsets are frequent: α_1=$\{a\}$, α_2=$\{b\}$, α_3=$\{e\}$, α_4=$\{a, e\}$, and α_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 $\langle s_1, \ldots, s_n\rangle$. 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 $S = \langle s_1, \ldots, s_m\rangle$ and $S' = \langle s'_1, \ldots, s'_n\rangle$ be two sequences of size m and n, respectively, with $m \leq n$. The tuple of integers $e = (e_1, \ldots, e_m)$ is an *embedding* of S in S' (denoted $S \sqsubseteq_e S'$) if and only if $e_1 < \ldots < e_m$ and for any $i \in \{1, \ldots, m\}$ it holds that $s_i = s'_{e_i}$.

Example 2
For the data set in Table 3, we have that $\langle b\,c\,e\,b\rangle \sqsubseteq_{e_1} \langle a\ b\ c\ d\ a\ e\ b\rangle$ for $e_1 = (2, 3, 6, 7)$ and analogously, $\langle a\ a\ e\rangle \sqsubseteq_{e_2} \langle a\ b\ c\ d\ a\ e\ b\rangle$ with $e_2 = (1, 5, 6)$. □

We are now ready to define an inclusion relation for sequences.

Definition 3 (*Sequence inclusion*)
Given two sequences $S = \langle s_1, \ldots, s_m\rangle$ and $S' = \langle s'_1, \ldots, s'_n\rangle$, of size m and n, respectively, with $m \leq n$, 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, that is.

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

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

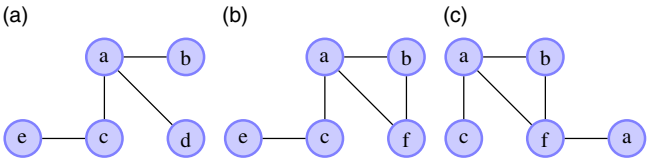


Fig. 1. Graph examples. (a) Graph G_1, (b) Graph G_2, and (c) Graph G_3.

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

For a given sequence S and a sequential data set $D = \{S_1, \ldots, S_n\}$ we denote by D_S the subset of D such that $S \sqsubseteq S'$ 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 data set $D = \{S_1, \ldots, S_n\}$ and a frequency threshold $\sigma \geq 0$, a sequence S is *frequent* in D if $sup(S) \geq \sigma$.

Example 4
For the data set in Table 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 $s_1 \not\sqsubseteq s_2$.

2.1.3 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 is a labeling function that maps each edge and each vertex to a label.

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

Uniquely labeled graphs. In this restricted setting, we assume that each node has a unique label within the graph, and no labels on the edges are provided. This restriction makes the sub-pattern check computationally easier and allows certain reductions to other pattern mining problems as indicated below.

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

General case of graph mining. In the general case, we consider graphs, whose nodes have labels, that are not necessarily unique. Here, a pattern is an arbitrary graph with labeled nodes and edges. This general case is computationally more demanding than the special case of uniquely labeled graphs.

In the further exposition of the results, the general case is assumed by default, unless explicitly stated otherwise.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

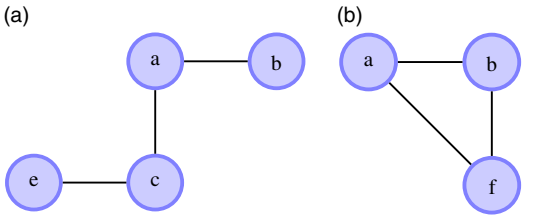


Fig. 2. Isomorphic subgraphs for graphs in Figure 1. (a) Graph G'_1 and (b) Graph G'_2.

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 a bijective function f such that

- $v \in V$ iff $f(v) \in V'$ and for all $v \in V$ it holds that $l(v) = l'(f(v))$,
- $e \in E$ iff $f(e) \in E'$ and for all $e \in E$ it holds that $l(e) = l'(f(e))$.

The graph isomorphism problem is claimed to be solvable in quasi-polynomial time (Babai 2015).

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'.

We now illustrate the introduce notions by the following example.

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

In the general setting, the problem of deciding whether a subgraph isomorphism exists (which is at the core of graph mining problems) is NP-complete (Cook 1971). However, several restricted settings have been identified, for which the problem can be solved in polynomial time, for example, unique-labeled undirected graphs (Kimelfeld and Kolaitis 2014).

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 fourth group are global ones.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

As argued in Section 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/subgraph 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 (local constraint) from $\mathcal{L}$ to $\{\top, \bot\}$ and let p be a pattern in $\mathcal{L}$. Then the pattern p is called *valid* iff $C(p) = \top$; otherwise it is referred to as *invalid*.

Example 7
Let C be a constraint function checking whether a given pattern is of size at least 2. Then in Example 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 $size(p) \leq size(q)$.

We are now ready to define dominated patterns under constraints.

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 $p' \in \mathcal{L}$ such that $p <^* p'$, and p' is valid under C.

Example 8
In Example 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 $\mathcal{L}$, 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.

Example 9
For the constraint function selecting maximal itemsets of size at least 2 and support at least 2, α_4 and α_5 from Example 1 are condensed patterns. The restriction on the pattern size rules out α_3. □

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Intuitively, a condensed representation is the smallest set of "good" patterns describing the data, that is, a set that does not contain any redundant or invalid patterns. By redundant here, we mean patterns dominated by others. More formally,

Definition 10 (*Condensed representation*)
A *condensed representation* is a set of all condensed patterns.

Condensed representations allow one to get insight about the data without analyzing all frequent patterns. This is advantageous, since typically there are orders of magnitude fewer condensed patterns than frequent ones. Moreover, many standard condensed representations (e.g. closed) allow a full reconstruction of the whole set of frequent patterns. In other words, from a substantially smaller set of condensed patterns, often the same knowledge can be extracted about the data, as from all frequent patterns. Note that in the literature condensed representations are also known under other names, for example, *dominating sets* (Négrevergne *et al.* 2013).

2.3 Answer set programming

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

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

where $0 \leq k \leq m$, `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 sets of atoms (respectively) that compose $body(r)$. A rule 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, that is, 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 appearing in Π). Any subset of $HB(P)$ is a *Herbrand interpretation*. $MM(\Pi)$ denotes the subset-minimal Herbrand interpretation that is a *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(Π).

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_superset(J,I):-pattern(I), item(I,V), I != J,
                       pattern(J), not item(J,V).
```

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Apart from the facts (1)–(5) the program Π contains a rule, which intuitively states that the pattern `J` is not a superset of the pattern `I` if `I` has an item `V` that J does not have. The grounding $Gr(\Pi)$ of Π is obtained from Π by substituting `I,J,V` with the constants `1,2,a,b` in all possible ways. Consider the interpretation A = {`pattern(1),pattern(2),item(1,a),item(1,b),item(2,a),not_superset(2,1)`}. The GL-reduct $\Pi^A(\Pi)$ for A contains the facts (1)–(5) and the rule (6) with only positive atoms in its body, and with `I,J,V` substituted by `2,1,b` respectively. A is the minimal model of $\Pi^A(\Pi)$, and thus it is in AS(Π).

Cardinality constraints. Cardinality constraints are extended literals (Simons *et al.* 2002). They are of the form

`l{b_1, ..., b_m}u,`

for $m \geq 1$, where `l, u` are lower and upper bounds on the cardinality of subsets of {`b_1, ..., b_m`} satisfied in an encompassing answer set. They can appear in the head or in the body of a rule. A cardinality constraint is satisfied in an answer set A, if the number of atoms from `b_1, ..., b_m` belonging to A is between `l` and `u`.

Example 11
For instance, `1 {a(X),b(X)} 3` is satisfied in A, whenever between 1 and 3 instances of `a(X),b(X)` are true in A. □

Other relevant language constructs include *conditional literals*. A *conditional literal* is an expression of the form

`{a:b_1,...,b_m},`

where `a` and `b_i` are possibly default negated literals. This expression denotes the set of atoms `a(X)` for which it holds that `b_1(X),...,b_m(X)` are true.

Example 12
For example, the cardinality atom: `k {in_subset(X) : in_set(X)} k` expresses the condition that a subset has exactly `k` elements when the predicate `in_set(X)` defines the elements that belong to the set and the predicate `in_subset(X)` defines the subset. □

Aggregate functions and aggregate atoms. An *aggregate function* is of the form `f(S)`, where `S` is a set and `f` is a function name among `#count, #min, #max, #sum,` and `#times`. An *aggregate atom* is

`Lg` $\prec_1$ `f(S)` $\prec_2$ `Rg,`

where `f(S)` is an *aggregate function*, $\prec_1, \prec_2 \in \{=, <, \leq, >, \geq\}$, and `Lg` and `Rg` (called left guard and right guard, respectively) are terms. One of `Lg`$\prec_1$ and $\prec_2$`Rg` can be omitted, in which case, "0" and "$+\infty$" are assumed, respectively. If both $\prec_1$ and $\prec_2$ are present, we assume for simplicity that $\prec_1 \in \{<, \leq\}$ if and only if $\prec_2 \in \{<, \leq\}$, and that both $\prec_1$ and $\prec_2$ are different from $=$ (Faber *et al.* 2008). We consider the standard semantics of aggregates:

- `#count(`X`)`, defined over a multiset of atoms X, is the number of atoms X that hold in an answer set (zero for the empty set),
- `#min(`X`)`, defined over a multiset of atoms X, is a minimal atom in X that holds in an answer set,

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

- `#max(X)`, defined over a multiset of atoms X, is a maximal atom in X that holds in an answer set,
- `#sum(X)`, defined over a multiset of atoms X, is the sum of atoms X (which are typically numbers) that hold in an answer set,
- `#time(X)`, defined over a multiset of atoms X, is the product of atoms X (which are typically numbers) that hold in an answer set (one for the empty set).

Encoding methodology and ASP solvers. In this paper we make use of two existing ASP systems Clasp (Gebser *et al*. 2007) and WASP (Alviano *et al*. 2013). We mostly focus on the former and use the latter to provide an experimental comparison on how two different systems perform on various pattern mining tasks with respect to runtime and memory usage.

Clasp extends the standard ASP theory described here with a number of features such as incremental grounding, preferences, satisfiability modulo theories, various solving parameters such as brave and cautious reasoning, parallel executions, heuristics, and meta-programming (Gebser *et al*. 2012).

A typical modeling approach in ASP follows the guess and check paradigm, where at first we define potential stable model candidates (typically through non-deterministic constructs) and then eliminate invalid candidates (typically through integrity constraints) (Gebser *et al*. 2007; Gebser *et al*. 2012; Lifschitz 2008), which in a nutshell allows us to write the following formula for ASP modeling:

$$\text{ASP program} = \text{Data} + \text{Generator} + \text{Tester} \ (+ \ \text{Optimizer}).^{2}$$

We follow this modeling paradigm throughout the paper. To demonstrate it, consider as an example, n-queen modeling problem, where one needs to put a number of queens on the board such that no queen attacks another. This can be modeled as in Listing 1. First we specify data as facts, in lines 2 and 3, then we define the set of possible answer sets using a choice rule in line 5 and finally we validate them in lines 7–9.

```
1 % Data specifying the board
2 col(1..k).
3 row(1..k).
4 % Choice rule to specify possible stable models
5 k { queen(Row,Col) : col(Col), row(Row) } k.
6 % Integrity constraint to validate the candidate models
7 :- queen(Rw,Cw), queen(Rb,Cb), Rw  = Rb, Cw != Cb.
8 :- queen(Rw,Cw), queen(Rb,Cb), Rw != Rb, Cw  = Cb.
9 :- queen(Rw,Cw), queen(Rb,Cb), Rw != Rb, |Rw - Rb| = |Cw - Cb|.
```

Listing 1. An ASP guess-and-check paradigm modeling example on the n-queen problem.

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 consists of two steps, where in the first step, we apply highly optimized algorithms for frequent pattern discovery and, in the second step,

[2] https://www.cs.uni-potsdam.de/~torsten/Potassco/Slides/asp.pdf.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

we use a declarative ASP solver for their convenient post-processing. Here, we mainly focus on itemsets, sequence, and graph mining.

Given a frequency threshold σ, a data set 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 frequent patterns from a given data set, 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 $\mathcal{C}_l$). 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. Analogously, we encode graphs with unique labels using facts `graph_u(i,j,k)`, where `i` is the pattern's ID, `j` and `k` are node labels, and the respective fact means that in the graph `i`, the node `j` is connected to `k`. In the general setting, we have `graph(i,j,k,l)`, where `i` is again the pattern's ID, `j` and `k` are node labels, and `l` is the label of an edge connecting `j` and `k` in `i`. 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 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 $\mathcal{C}_g$. 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 Definition 7. For every constraint in $\mathcal{C}_l$, we have a set of dedicated rules, stating 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,J:item(I,J),cost(J,C)} > N, pattern(I).
```

A similar rule for sequences can be defined as follows:

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

Analogously, one can specify arbitrary domain constraints on patterns.

Example 13

Consider a data set storing moving habits of young people during their studies. Let the dedicated frequent sequence mining algorithm return the following patterns: $S_1 = \langle bG\ mA\rangle\ ba\ mG\ ma$; $S_2 = \langle bA\rangle\ mG\ ba\ mA\ ma$; $S_3 = \langle bUS\ mA\ ba\ mUS\ ma\rangle$, where

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

bG, bA, bUS stand for born in Germany, Austria and US, ba, ma stand for bachelors and masters and the predicates mG, mA, mUS reflect that a person moved to Germany, Austria, and USA, respectively. Suppose, we are only interested in moving habits of German native speakers, who got their masters degree from a German university. The local domain constraint expressing this would state that (1) bUS should not be in the pattern, while (2) either both bG and ma should be in it without any mA or mUS in between or mG should directly precede ma. These constraints are encoded in the program in Listing 2. From the answer set of this program we get that both S_2 and S_3 are not valid, while S_1 is. □

```
time(1..5).
% people born in Germany or Austria are German-speaking
gs(I) :- seq(I,bG,P).
gs(I) :- seq(I,bA,P).
% collect those who moved to Austria or US before P
moved_before(X,P) :- seq(X,mA,P1), P > P1, time(P), time(P1).
moved_before(X,P) :- seq(X,mUS,P1), P > P1, time(P), time(P1)
% collect those who moved to Austria or US after P and before masters
moved_after(X,P) :- seq(X,mA,P1), seq(X,ma,P2), P < P1,
                    P1 < P2, time(P), time(P1), time(P2).
moved_after(X,P) :- seq(X,mUS,P1), seq(X,ma,P2), P < P1,
                    P1 < P2, time(P), time(P1), time(P2).
% keep German speakers who moved to Germany straight before masters
keep(X) :- seq(X,ma,P+1), seq(X,mG,P), gs(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 German speakers whose last move before masters was to Germany
keep(X) :- seq(X,mG,P1), seq(X,ma,P2), P1 < P2,
           gs(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 2. Moving habits of people during studies.

To combine all local constraints from $\mathcal{C}_l$ 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 $\mathcal{C}_g$.

3.2 Encoding global constraints

The key for encoding global constraints is the declarative formalization of the dominance relation (Defintion 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 $\mathcal{L}$ in this section is supplied with a rule, which guesses (`selected/1` predicate) a single `valid` pattern to be a candidate for inclusion in

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

the condensed representation, and a constraint that rules out `dominated` patterns thus enforcing a different guess.

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

:- dominated.
```

```
% J is not a superset of I if I has items that are not in J
not_superset(J) :- selected(I), item(I,V), not item(J,V),
                   valid(J), I != J.
% derive dominated whenever I is a subset of J
dominated :- selected(I), valid(J),
             I != J, not not_superset(J).
```

Listing 3. Maximal itemsets encoding.

In what follows, we discuss concrete realizations of the dominance relation both for itemsets and sequences for various global constraints, that is, 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 Listing 3. 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. We provide a correctness proof of this encoding in the Appendix. The correctness of the rest of the encodings can be analogously shown.

For closed itemset mining, a simple modification of Listing 3 is required. An itemset is *closed* if none of its supersets has the same support. Thus to both of the rules from Listing 3, we need to add atoms `support(I,X), support(J,X)`, which store the support 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:

```
% support and size comparison among patterns
g_size_geq_fr(J) :- selected(I), valid(J), support(I,X),
                    support(J,Y), size(I,Si), size(J,Sj),
                    Si <  Sj, X <= Y.
geq_size_g_fr(J) :- selected(I), valid(J), 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_size_geq_fr(J).
dominated :- valid(J), geq_size_g_fr(J).
```

Listing 4. Skyline itemsets encoding.

```
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 Listing 4, where the first two rules specify the conditions (a) and (b) for skyline itemsets as specified in Section 2.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

```
% if V appears in a valid pattern I, derive in(V,I)
in(V,I) :- seq(I,V,P), valid(I).
% J is not a superset of I if I has V that J does not have
not_superset(J) :- selected(I), valid(J), I != J,
                   seq(I,V,P), not in(V,J).
% if for a subseq <V,W> in I there is V followed
% by W in J then deduce domcand(V,J)
domcand(V,J,P) :- selected(I), seq(I,V,P), seq(I,W,P+1), I != J
                  valid(J), seq(J,V,Q), seq(J,W,Q'), Q'>Q.
% if domcand(V,J) does not hold for some V in I
% and a pattern J then derive not_dominated_by(J)
not_dominated_by(J) :- selected(I), seq(I,V,P), seq(I,W,P+1),
                       I != J, valid(J), not domcand(V,J,P).
% if neither not_dominated_by(J) nor not_superset(J)
% are derived for some J, then I is dominated
dominated :- selected(I), valid(J), I != J,
             not not_superset(J), not not_dominated_by(J).
```

Listing 5. Maximal sequence encoding.

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 Listing 5, 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, `J` is not a superpattern of `I` if it is not a superset of `I` [lines (4) and (5)], that is, if `not_superset(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 Listing 5 with the following ones:

```
not_superset(J) :- selected(I), in(V,J),
                   not in(V,I), I != J.
domcand(V,J,P):- selected(I), seq(J,V,P), seq(J,W,P+1),
                 seq(I,V,Q),seq(I,W,Q'), Q'>Q, I != J.
not_dominated_by(J) :- selected(I), valid(J), I != J,
                       seq(J,V,P), seq(J,W,P+1),
                       not domcand(V,J,P).
```

Finally, the encoding for mining skyline sequences coincides with the skyline itemsets encoding, which is provided in Listing 4.

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.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Uniquely labeled graphs. One of such graph types are undirected graphs, where every node has a unique label [and consequently, every edge (u, v) is labeled uniquely as $(l(u), l(v))$]. For instance, the first two graphs in Figure 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 Listing 6. 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.

```
4 % I is not a subgraph of J if I has an edge (X,Y) that J does not have
5 not_supergraph(J) :- selected(I), graph_u(I,X,Y), valid(J),
6                      not graph_u(J,X,Y), not graph_u(J,Y,X), I != J.
7 % derive dominated, whenever I is a subgraph of J
8 dominated :- selected(I), valid(J), not not_supergraph(J), I !=J.
```

Listing 6. Maximal graph encoding (unique labeled; IDs represent labels as well; graph_u(I,X,Y) represents here a graph I with an edge between X and Y, whose values are their labels, respectively).

The encoding for closed frequent graph patterns differs from the one for maximal graph patterns only in that in the second rule in Listing 6 the atoms `support(I,X)`, `support(J,X)` are added. Encodings for other condensed representations are analogous.

Example 14
The edges of the graph G_1 in Figure 1 are represented with the following facts `graph_u(g1,e,c)`, `graph_u(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. □

General case of graph mining. 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 Listing 7 (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 Listing 7 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 labeled 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 line (2) of Listing 7 the guess is performed on a graph in the set F of frequent patterns, to which G can be mapped, that is, a dominating candidate pattern is guessed.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Second, in line (4) the guess on a mapping from G to the dominating candidate graph is done. Finally, in lines (6)–(9) the mapping is validated by two integrity constraints. Finally, the constraint in lines (10)–(11) ensures the injectivity of the constructed mapping. If the solver returns an answer set, then the selected graph G is removed from the data set as a dominated pattern, that is, it is guaranteed to be not maximal.

```
1     % pick one graph to map to the originally selected
2 1 { mapped_to(X) : graph(X,_,_,_)} 1.
3 % guess the mapping to the selected graph
4 1 { map(X,N) : node(I,N,_) } 1 :- selected_node(X,_),  mapped_to(I).
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 graph(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).
10 % enforces injectivity of the mapping
11 :- map(X,V1), map(X,V2), V1 != V2.
```

Listing 7. Maximal graph encoding (core); general case with non-unique labels; the method picks a graph from the set of patterns and (a) removes it from the graph data set (we know the picked graph would always match itself) (b) reify it as `selected_node(X,L)` and `selected_edge(X,Y,L)`, then the encoding is trying to construct an injective mapping `map(X,N)`, which maps nodes in the originally selected graph to the one picked in the data set, that is, `mapped_to(I)` to one of the graphs in the data set, that is, the graph predicate (since it constructs a mapping names or ordering of nodes does not play a role here) such that it preserves edges and labels. If it fails to construct such a mapping, then the original selected graph is 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 study approximate patterns, that is, patterns that are considered as 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 $\tau(\alpha, D) = \{(t_{id}, i) \mid t_{id} \in D_\alpha, i \in \alpha\}$. When D is clear from the context, we write $\tau(\alpha)$. The area of $\tau(\alpha)$ is equal to its cardinality.

Example 15
The tile corresponding to the itemset $\{b, e\}$ from Table 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

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

of k tiles with the maximum area, whereas in the *Minimum Tiling* the goal is to find the least number of tiles such that for all $t_{id} \in D$ and for all $i \in t_{id}$, there exists at least one tile $\tau(\alpha)$ such that $(t_{id}, i) \in \tau(\alpha)$ (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 $\mathcal{I}$. 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. 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 16
The transaction database in Table 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 15 corresponds to the matrix

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

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 $\mathbf{T}_\ell = \mathbf{u}_\ell \mathbf{v}_\ell^T$, 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 possibly 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 17
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. $\square$

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

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 11 (*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

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

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

It is worth noticing that equation (3) corresponds to the Hamming distance between $\mathbf{D}$ and $\mathbf{S}$. 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 $\Omega(2^{\log^{1-\epsilon}|\mathbf{D}|})$ for any $\epsilon > 0$, where $|\mathbf{D}|$ is the number of 1s in $\mathbf{D}$ (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 $\mathbf{A}$, and a binary target vector $\mathbf{b}$, find a binary vector $\mathbf{x}$ that minimizes the Hamming distance between $\mathbf{b}$ and $\mathbf{Ax}$, where the matrix-vector product is over Boolean algebra.

```
% 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).
% among tiles guess at least one to be included in the tiling
1 { intiling(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 one(Y,Z).
numberofinsidezeros(N) :- N = #count{Y, Z : insidezero(Y,Z)}.
% compute the total error
er(0,K) :- numberofinsidezeros(K).
er(1,K) :- numberofoutsideones(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 8. Encoding for the approximate tiling problem from Definition 11.

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,

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

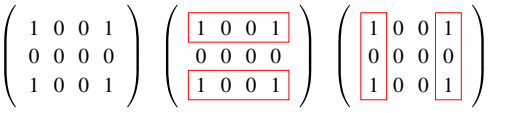


Fig. 3. Examples of tiles in a matrix.

which will be a solution to the problem from Definition 11. In Listing 8, 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. A natural encoding of tiles is via the positions of 1s that they contain, that is, for a tile with ID `id` that has 1 in the intersection of a column `i` and a row `t`, we could store a fact `tile(id,i,t)`. For instance, provided that the ID of the red tile from Example 17 is `1`, it can be represented as a set of facts `tile(1,1,1),tile(1,2,1),tile(1,1,2),tile(1,3,2)`. However, storing only information about 1s in a tile is insufficient for its encoding, since multiple tiles can have 1s in the same positions, as illustrated next.

Example 18
Consider the matrix in Figure 3 and two of its tiles. If we only encode tiles using the positions of 1s in them, then both of the tiles in Figure 3 will have the exact same representation given by the following facts `tile(1,1,1), tile(1,4,1), tile(1,1,3), tile(1,4,3).` □

In order to ensure that the encoding represents a unique tile, along with the positions of 1s, we also need to identify columns and rows that are parts of a tile but contain only 0s. We do that by storing IDs of all items and transactions that are part of a tile (i.e. relevant to it) using the facts `relitem(id,i)` and `reltrans(id,t),` respectively.

Example 19
The facts in Example 18 and {`relitem(1,1),relitem(1,2),relitem(1,3),relitem(1,4), reltrans(1,1), reltrans(1,2)`} uniquely represent the tile in the middle of Figure 3. Analogously, if we instead {`relitem(1,1), relitem(1,2), reltrans(1,3)`} are added to the facts in Example 18, then the encoding of the tile on the right-hand side of Figure 3 is obtained. □

Note that if there exists a 1 in the original data that is not a part of any tile, no method will be able to cover it. Hence such 1s are accumulated into a constant error, and we ignore them in our setting.

In lines (1)–(3) of Listing 8 we collect tiles' IDs and locations of ones using the dedicated predicates `tileid` and `one` respectively. Than in lines (4) and (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.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

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 (in terms of runtime and memory usage) than declarative mining approaches (Négrevergne and Guns 2015), yet obviously less flexible (in terms of modeling flexibility, i.e., how general the method is and how easily it can be modified to model a variation of the problem).

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 Definition 11.

We report our evaluation results on two transaction data sets[3]: *Mushrooms* (8124 transactions/119 items) and *Vote* (435/48); three sequence data sets (full)[4]: *JMLR* (788 sequences/3847 distinct symbols), *Unix Users* (9099/4093), and *iPRG* (8628/21); three graph data sets[5]: *Yoshida* (265 graphs/20 avg. vertices/23 avg. edges/9 distinct labels), *Nctrer* (232/19/20/9), and *Bloodbarr*(413/21/23/9), and two data sets for the tiling problem: *Divorce*[6] (50 transactions/9 items) and *Glass*[7] (214/48). All experiments have

[3] https://dtai.cs.kuleuven.be/CP4IM/datasets/.
[4] https://dtai.cs.kuleuven.be/CP4IM/cpsm/datasets.html.
[5] https://github.com/amaunz/ofsdata.
[6] https://sparse.tamu.edu/Pajek/divorce.
[7] http://cgi.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

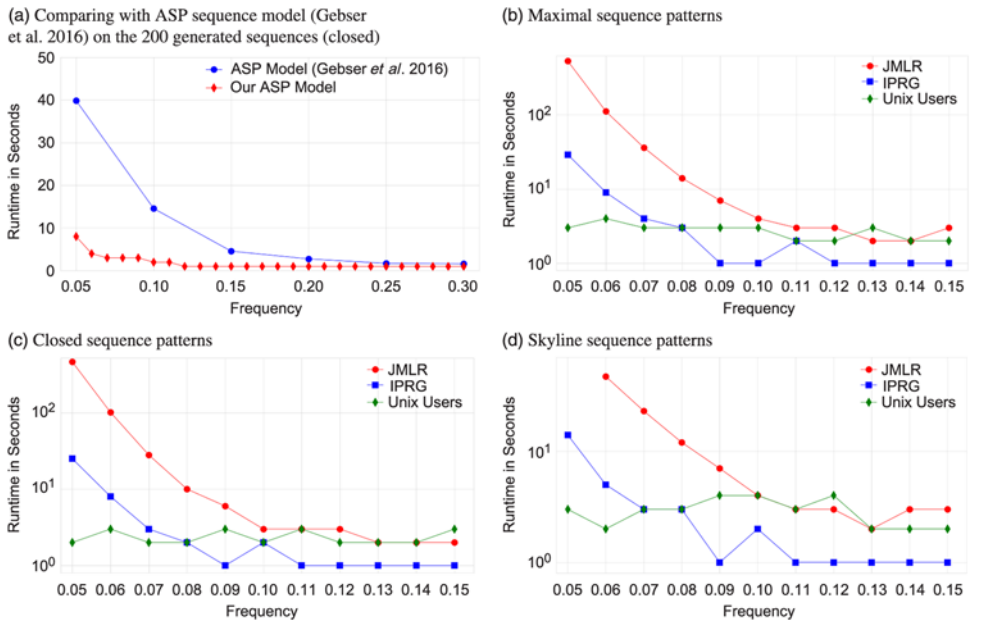


Fig. 4. Investigating $\mathbf{Q_1}$: comparison with pure ASP model (a) and maximal (b), closed (c), and skyline (d) sequence mining on JMLR, Unix Users, and iPRG data sets.

been performed on a desktop with Ubuntu 14.04, 64-bit environment, Intel Core i5-3570 4xCPU 3.40 GHz and 8 GB memory using clingo 4.5.4[8] 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.[9] The timeout was set to 1 h. 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 Figure 4(a), we compare the ASP model (Gebser *et al.* 2016) with our method on the default 200 sequence sample, generated by the tool[10] of Gebser *it 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 data sets for any support threshold value within the timeout. One can observe that our method consistently outperforms the purely declarative approach of Gebser *et al.* (2016) and the advantage naturally becomes more apparent for smaller frequency threshold values.

In Figure 4(b)–(d) (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 data sets. In contrast to Gebser *et al.* (2016), our method managed to produce results on all of these data sets 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

[8] http://potassco.sourceforge.net.
[9] https://github.com/alviano/wasp.
[10] https://sites.google.com/site/aspseqmining.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

standard data sets Vote and Mushrooms. As we see in Figure 5(a) and (b), 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 [Figure 5(c)]; this holds also for the Mushrooms data set, 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 Figure 5(c). 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 Figure 5(d), 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 is the filtering technique used to select candidate patterns among only valid patterns. Its effect can be clearly seen, for example, for the Vote data set in Figure 5(d), 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 Figure 5(a), 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 data sets in Figure 6(a), (b), and (c). One can observe, that our system is indeed able to handle real-sized data sets and process hundreds or even thousands of graph patterns within the time limit. Furthermore, the comparison with the purely declarative framework presented in Figure 6(d) 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 Definition 11, 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 setting. In Tables 4 and 5, we report the results for the Divorce and Glass data sets, respectively. We compute the candidate tiles using an approach based on association confidences (see Miettinen *et al.* 2008). Given the set of

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

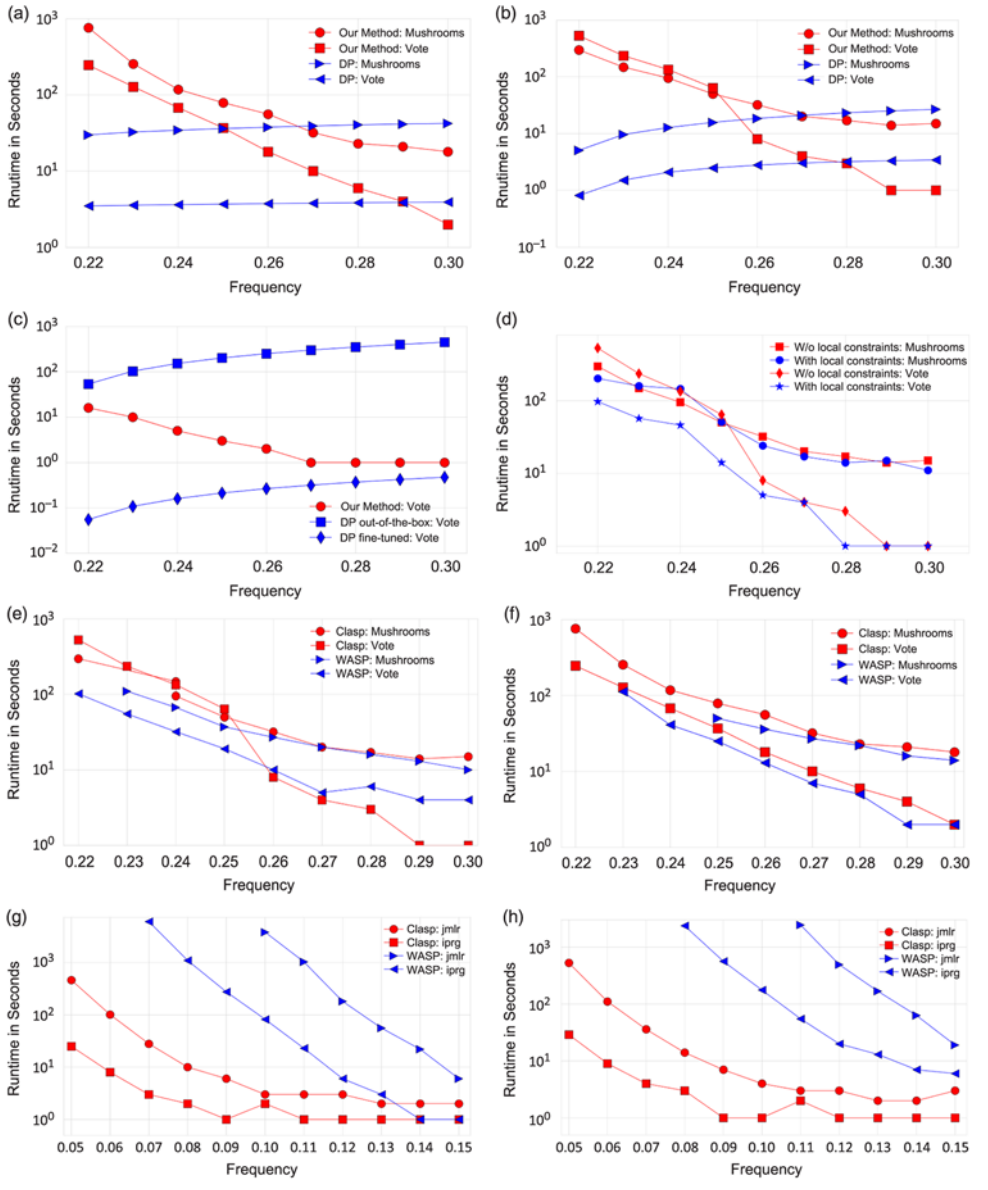


Fig. 5. $\mathbf{Q_4}$: comparing ASP solvers performance – WASP and Clasp on itemset and sequence mining tasks. (a) Maximal itemset mining: comparing with DP on Vote and Mushrooms. (b) Closed itemset mining: comparing with DP on Vote and Mushrooms. (c) Skyline itemset mining: comparing with out-of-the-box and fine-tuned DP on Vote. (d) Closed itemset mining: our method with (w/o) local constraints on Vote and Mushrooms. (e) Itemset mining: closed patterns. (f) Itemset mining: maximal patterns. (g) Sequence mining: closed patterns. (h) Sequence mining: maximal patterns.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

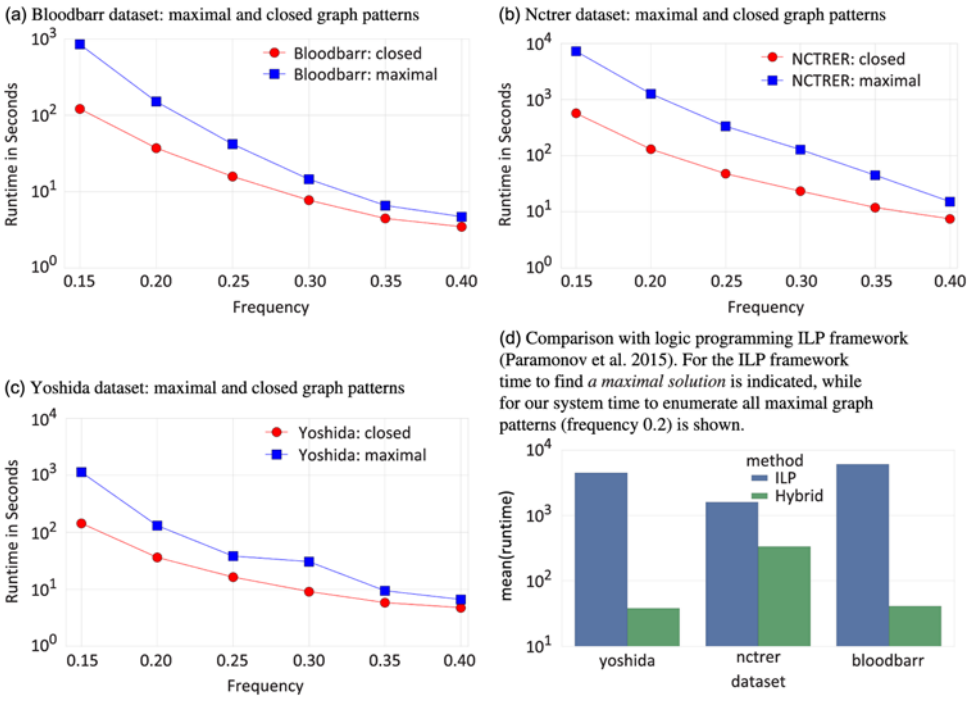


Fig. 6. Investigating $\mathbf{Q_5}$: evaluation on standard graph data sets (a),(b),(c) and comparison to the declarative ILP system for graph mining (d). Log-scale on all plots.

candidate tiles, we use our ASP encoding from Listing 8 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, proposed by Miettinen *et al.* (2008), 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 Table 4 (respectively Table 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 greedy method, that is, 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, 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 greedy and our hybrid algorithms is not entirely fair, as the former is not complete (i.e. it might not find

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Table 4. *Addressing* $\boldsymbol{Q_6}$*: approximate tiling problem solution using specialized native algorithm for tile candidate generation and ASP encoding for selecting the resulting tiling (divorce data set)*

	Greedy		Hybrid						
			First						
n/σ	k	Error	k	Error	Time	k opt	Error opt	Time all	|All solutions|
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 5. *Addressing* $\boldsymbol{Q_6}$*: approximate tiling problem solution using specialized native algorithm for tile candidate generation and ASP encoding for selecting the resulting tiling (glass data set)*

	Greedy		Hybrid						
			First						
n/σ	k	Error	k	Error	Time	k opt	Error opt	Time all	|All solutions|
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

any tiling, satisfying the conditions imposed by the error threshold even if one exists), which is in contrast to our hybrid approach.

The greedy algorithm is capable of computing the optimal tiling only for small instances of the Divorce data set. Starting from $n = 5$, our hybrid method outperforms the greedy one with respect to the quality of the found solutions. For the Glass data set 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 data sets 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

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

discussed in $\mathbf{Q_4}$. In $\mathbf{Q_5}$, it 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, for example, 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 among 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 CP (Rojas *et al.* 2014; Négrevergne and Guns 2015; Métivier *et al.* 2013), SAT solving (Jabbour *et al.* 2015, 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. While the ASP (Gebser *et al.* 2016) and CP models (Négrevergne and Guns 2015) for frequent pattern mining cannot be hybridized completely out-of-the-box, due to dependencies between constraints and assumptions on the input–output structure, in

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

principle, as we see from our work that it is possible to re-use ideas and the general modeling approach to turn them into hybrid models to reach more practical performance, for example, our ASP model for sequence mining is inspired by the non-hybrid ASP model of Gebser *et al.* (2016). The main observation here is that typically these systems have a number of constraint groups: to generate patterns, check patterns validity and perform dominance or group-property checks. Our hybridization idea suggests to replace one of these groups with a highly optimized solver, while keeping the other groups (with minor changes to adapt for the input–output structure) in the model to guarantee generality.

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 toward 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.* 2009a) 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 data set 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 1993*, ACM, New York, NY, USA, 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, Menlo Park, 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, Berlin, 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*, Springer, Cham, 315–330.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

Babai, L. 2015. Graph isomorphism in quasipolynomial time. *CoRR abs/1512.03547*.

Bonchi, F. and Lucchese, C. 2006. On condensed representations of constrained frequent patterns. *Knowledge and Information Systems 9,* 2, 180–201.

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

Eiter, T., Brewka, G., Dao-Tran, M., Fink, M., Ianni, G. and Krennwallner, T. 2009a. Combining nonmonotonic knowledge bases with external sources. In *Frontiers of Combining Systems, 7th International Symposium*, S. Ghilardi and R. Sebastiani, Eds. Frontiers of Combining Systems. FroCoS 2009. Lecture Notes in Computer Science, vol. 5749. Springer, Berlin, Heidelberg, 16–18 Sep. 2009, 18–42.

Eiter, T., Ianni, G. and Krennwallner, T. 2009b. Answer set programming: A primer. In *5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone*, Italy, 30 Aug.–4 Sep. 2009. LNCS, vol. 5689. Springer, Berlin.

Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T. and Ielpa, G. 2008. Design and implementation of aggregate functions in the DLV system. *CoRR abs/0802.3137*.

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*, AAAI, New York, NY, USA, 9–15 July 2016.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, San Rafael.

Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T. 2007. *clasp*: A conflict-driven answer set solver. In *LPNMR*, Springer, 260–265.

Geerts, F., Goethals, B. and Mielikäinen, T. 2004. Tiling databases. In *Proc. of the 7th International Conference on Discovery Science*, DS 2004, Springer, Padova, Italy, 2–5 Oct. 2004, 278–289.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP*, MIT Press, 1070–1080.

Guns, T., Dries, A., Nijssen, S., Tack, G. and De Raedt, L. 2017. MiningZinc: A declarative framework for constraint-based mining. *Artificial Intelligence 244*, 6–29.

Guns, T., Nijssen, S., and de Raedt, L. 2013. k-pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering 25,* 2, 402–418.

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*, AAAI Press, Phoenix, AZ, USA, 13 Feb. 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 2013, San Francisco, CA, USA, 27 Oct.–1 Nov. 2013, ACM, 649–658.

Jabbour, S., Sais, L. and Salhi, Y. 2015. Decomposition based SAT encodings for itemset mining problems. In *PAKDD*, Springer, 662–674.

Järvisalo, M. 2011. Itemset mining as a challenge application for answer set enumeration. In *Proc. of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR 2011, Vancouver, Canada, 16–19 May 2011, Springer, 304–310.

Kimelfeld, B. and Kolaitis, P. G. 2014. The complexity of mining maximal frequent subgraphs. *ACM Transactions on Database Systems 39,* 4, 32–33.

Lifschitz, V. 2008. What is answer set programming? AAAI, Chicago, IL.

Mannila, H. and Toivonen, H. 1997. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery 1,* 3, 241–258.

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

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. *Information Processing Letters 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 2015*, Springer, 36–52.

Miettinen, P., Mielikäinen, T., Gionis, A., Das, G. and Mannila, H. 2008. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering 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, 7–10 Dec. 2013, IEEE, 557–566.

Négrevergne, B. and Guns, T. 2015. Constraint-based sequence mining using constraint programming. In *CPAIOR*, Springer, 288–305.

Neumann, S. and Miettinen, P. 2017. Reductions for frequency-based data mining problems. In *Proc. of the 2017 IEEE International Conference on Data Mining*, IEEE, New Orleans, LA, USA, 997–1002.

Paramonov, S., van Leeuwen, M., Denecker, M. and De Raedt, L. 2015. An exercise in declarative modeling for relational query mining. In *ILP*, Springer, 166–182.

Pei, J. and Han, J. 2000. Can we push more constraints into frequent pattern mining? In *ACM SIGKDD*, ACM, 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 2005*, Springer, 212–223.

Rojas, W. U., Boizumault, P., Loudni, S., Crémilleux, B. and Lepailleur, A. 2014. Mining (soft-) skypatterns using dynamic CSP. In *CPAIOR*, Springer, 71–87.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1–2, 181–234.

Tyukin, A., Kramer, S. and Wicker, J. 2014. BMaD – A boolean matrix decomposition framework. In *ECML PKDD 2014*, T. Calders, F. Esposito, E. Hüllermeier and R. Meo, Eds., Springer, 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 2002*, Maebashi City, Japan, Japan, IEEE.

Yan, X., Han, J. and Afshar, R. 2003. CloSpan: Mining closed sequential patterns in large datasets. In *SDM*. 166–177.

Zaki, M. J., Parthasarathy, S., Ogihara, M. and Li, W. 1997. New algorithms for fast discovery of association rules. Technical report, University of Rochester, Rochester, NY, USA.

Appendix A Correctness of ASP encoding for maximal itemset computation

Proposition A.1. Let S be a set of frequent itemsets, represented as facts of the form `item(i,j)` reflecting that the itemset with ID `i` contains an item `j`. Moreover, let `valid(i)` be in S if `i` is a valid pattern, and let P be the following logic program:

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press

```
(1)  1 {selected(I) : valid(I)} 1.
(2) not_superset(J) :- selected(I), item(I,V), not item(J,V), valid(J), I != J.
(3) dominated :- selected(I), valid(J), I != J, not not_superset(J).
(4) :- dominated.
```

Then it holds that

(i) if $I \supseteq$ `selected(x)` is an answer set of $P \cup S$, then x is a valid maximal itemset and
(ii) if an itemset with ID `y` is a valid maximal itemset then an answer set I of $P \cup S$ exists, such that `selected(y)` $\in I$.

Proof.

(i) Let $I \supseteq$ `selected(x)` be an answer set of $P \cup S$. Since S does not contain any facts over the `selected` predicate by our assumption, we have that `selected(x)` must have been obtained due to the rule (1), that is, `valid(x)` $\in S$, meaning that the itemset `x` is valid. Since the set S contains only frequent itemsets, `x` must be also frequent. Moreover, we know that S also stores all other frequent itemsets. Therefore, in order to show that `x` is maximal, we need to show that no valid itemset in S is a superset of `x`. Toward a contradiction, assume the contrary, and let `x'` be such an itemset. Then it holds that every item that is in `x` is also contained in `x'`. However, in this case the body of the grounding of the rule (2) with `I` substituted by `x` and `J` by `x'` is not satisfied. Thus, `not_superset(x')` $\notin I$, as the respective predicate does not appear in any other rule head of P. However, then the body of (3) is satisfied for the grounding with `I` and `J` being substituted by `x` and `x'`, respectively, meaning that `dominated` must be present in I, for it to be a model of $P \cup S$, but then the constraint (4) is violated, leading to the contradiction of I being a model of $P \cup S$.

(ii) Suppose that `x` is a valid maximal itemset. Then by construction of S, `valid(x)` $\in S$. Consider an interpretation $I = S \cup \{$`selected(x)`$,$`valid(x)`$\cup \{$`not_superset(x')` $\mid$ `valid(x')` $\in S,$ `x'` $\neq$ `x`$\}\}$. We show that this interpretation is an answer set of $P \cup S$. Toward a contradiction, assume the contrary. Then either (a) I is not a model of $P \cup S$ or (b) I is not minimal. First suppose that (a) holds. Then we have that $body(r) \in I$, but $head(r) \notin I$ for some rule r among the rules (1)–(4). Note that since by construction, I contains just a single atom over the `selected` predicate, that is, `selected(x)`, such that `valid(x)` $\in S$, the rule (1) must be satisfied. Assume that (2) is not satisfied. Then the itemset `x` must contain some item `j`, which another valid itemset `x'` does not contain, but `not_superset(x')` $\notin I$. By construction of I, we have `not_superset(x)` for all valid itemsets in S apart from `x`; in particular, `not_superset(x')` $\in I$, meaning that (2) is satisfied. Therefore, (3) must be a problematic rule. However, since by construction `not_superset(x')` $\in I$ for all `x'` such that `valid(x')` $\in S$, the body of (3) cannot be satisfied. Finally, again by construction `dominated` $\notin I$. This means that I is a model of $P \cup S$. Hence, (b) must hold, that is, $I' \subseteq I$ must exists, such that I' is a model of $P \cup S$. Note that for I' to be a model, it must contain all facts in S. The assumption that `selected(x)` $\in I \backslash I'$ is not valid by construction of I. If `not_superset(x')` $\in I \backslash I'$ for some `x'`, such that `valid(x')` $\in S$ then the body of rule (3) is satisfied by I' but not its head, as `dominated` $\notin I$ by construction. This means that I must be a minimal model of $P \cup S$ and thus its answer set. □

https://doi.org/10.1017/S1471068418000467 Published online by Cambridge University Press