An Exercise in Declarative Modeling for Relational Query Mining

Sergey Paramonov, Matthijs van Leeuwen, Marc Denecker, and Luc De Raedt

KU Leuven, Celestijnenlaan 200A, 3001 Heverlee - Belgium,
firstname.lastname@cs.kuleuven.be

Abstract. Motivated by the declarative modeling paradigm for data mining, we report on our experience in modeling and solving relational query and graph mining problems with the IDP system, a variation of the answer set programming paradigm. Using IDP or other ASP languages for modeling appears to be natural given that they provide rich logical languages for modeling and solving many search problems and that relational query mining (and ILP) employs the same type of representation.
Nevertheless, our results indicate that second order extensions to these languages are necessary for expressing the model as well as for efficient solving, especially for what concerns subsumption testing. We propose such second order extensions and evaluate their potential effectiveness with a number of experiments in subsumption as well as in query mining.

Keywords: knowledge representation, answer set programming, data mining, query mining, pattern mining

1 Introduction

In the past few years, many pattern mining problems have been modeled using constraint programming techniques [1]. While the resulting systems are not always as efficient as state-of-the-art pattern mining systems, the advantages of this type of declarative modeling are now generally accepted: they support more constraints, they are easier to modify and extend, and they are built using general purpose systems. However, so far, the declarative modeling approach has not yet been extended to inductive logic programming. This paper investigates whether such an extension would be possible. To realize this, we consider frequent query mining, the ILP version of frequent pattern mining, as well as the answer set programming paradigm, the logic programming version of constraint programming. More specifically, we address the following three questions:

- **Q_1** Is it possible to design and implement a declarative query miner that uses a logical and relational representation for both the data and the query mining problem?
- **Q_2** Is it possible to take advantage of recent progress in the field of computational logic by adopting an Answer Set Programming (ASP) [2] framework for modeling and solving?
- **Q_3** Would such a system be computationally feasible? That is, can it tackle problems of at least moderate size?

Our study is not only relevant to ILP, but also to the field of knowledge representation and ASP as query mining (and ILP) is a potentially interesting application that may introduce new challenges and suggest solver extensions.

More concretely, the main contributions of this work can be summarized as follows:

1. We present two declarative models and corresponding solving strategies for the query mining problem that support a variety of constraints. While one model can be expressed in the ASP paradigm, the other model requires a second order extension that we believe to be essential for modeling ILP tasks.
2. We implement and evaluate the presented models in the IDP system [3], a knowledge base system that belongs to the ASP paradigm.
3. We empirically evaluate the proposed models and compare them on classical datasets with state-of-the-art ILP methods.

This paper is organized as follows: Section 2 formally introduces the problem. In Section 3, we introduce a second order model for frequent query mining that addresses Question $\mathbf{Q_1}$. In Section 4 we present a first order model for query mining, demonstrate main issues with this approach and address Question $\mathbf{Q_2}$. In Section 5 we provide experimental evidence to support our answers to Questions $\mathbf{Q_2}$ and $\mathbf{Q_3}$. In Section 6 we discuss advantages (such as extendability) and disadvantages of the models and the approach overall. In Section 7 we present an overview of the related work in the ILP context of frequent query mining. Finally, we conclude in Section 8 with a summary.

2 Problem statement

The problem that we address in this paper is to mine queries in a logical and relational learning setting. Starting with the work the Warmr system [4], there has been a line of work that focusses on the following frequent query mining problem [5, 6, 7]:

Given:

- a relational database D,
- the entity of interest determining the *key* predicate,
- a frequency threshold t,
- a language $\mathcal{L}$ of logical queries of the form $key(X) \leftarrow b_1, ..., b_n$ defining $key/1$ (b_i's are atoms).

Find: all queries $c \in \mathcal{L}$ s.t. $freq(c, D) \geq t$, where $freq(c, D) = |\{\theta \mid D \cup c \models key(X)\theta\}|$.

Notice that the substitutions θ only substitute variables that appear in the conclusion part of the clause, i.e., only for variables X that occur in *key*.

In this paper, we focus our attention on graph data, as this forms the simplest truly relational learning setting and allows us to focus on what is essential for extending the declarative modeling paradigm to a relational setting. In principle, this setting can easily be generalized to the full inductive logic programming problem.

As an example, consider a graph database D, represented by the facts

$$\{edge(g_1, 1, 2),\ edge(g_1, 2, 3),\ edge(g_1, 1, 3),\ edge(g_2, 1, 2),\ edge(g_2, 2, 3),\ edge(g_2, 1, 3), \ldots\},$$

where the ternary relation $edge(g, e_1, e_2)$ states that in graph g there is an edge between e_1 and e_2 (we assume graphs to be undirected, so there is also always an edge between e_2 and e_1). The frequency of $key(K) \leftarrow edge(K, B, C), edge(K, C, D), edge(K, B, D)$

in this database is 2 as the query returns g_1 and g_2. If $key(g)$ holds, then the graph g is subsumed by the query specified defined in the body of the clause for *key*.

The goal of this paper is to explore how such typical ILP problems can be encoded in ASP languages. So, we will need to translate the typical ILP or Prolog construction into an ASP format. In the present paper, we employ IDP, which belongs to the ASP family of formalisms. Most statements and constraints written in IDP can be translated into standard ASP mechanically. A brief example-based introduction to IDP is in Appendix A and for a detailed system and paradigm description we refer to the IDP system and language description [3] and to the ASP primer [2].

To realize frequent query mining in IDP, we need to tackle four problems: 1) encode the queries in IDP; 2) implement the subsumption test to check whether a query subsumes a particular entity in the database; 3) choose and encode a language bias; and 4) determine, in addition to frequency, further constraints that could be used and encode them. We will now address each of these problems in turn.

3 Encoding

We assume that the dataset D is encoded as two predicates: $\textit{edge}(g, e_1, e_2)$, described before, and the ternary relation $\textit{label}(g, n, l)$ that states that there is a node n with label l in graph g (for a discussion on how to extend this approach, see Section 6) .

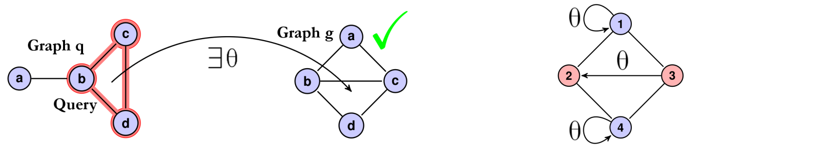
Encoding a query The coverage test in ILP is often θ- or OI-subsumption.

Definition 1 **(OI and θ-subsumption [8]).** *A clause c_1 θ-subsumes a clause c_2 iff there exists a substitution θ such that $c_1\theta \subseteq c_2$. Furthermore, c_1 OI-subsumes c_2 iff there exists substitution θ such that $\textit{com}(c_1)\theta \subseteq \textit{com}(c_2)$, where $\textit{com}(c) = c \cup \{t_i \neq t_j \mid t_i \text{ and } t_j \text{ are two distinct terms in } c\}$.*

As ASP and, in particular, IDP are model-generation approaches, they always generate a set of ground facts. This implies that ultimately the queries will have to be encoded by a set of ground facts for *edge* and *label* as well (e.g., $\{\textit{edge}(q, 77, 78),\ \textit{edge}(q, 77, 79),\ \textit{label}(q, 77, a),\ \textit{label}(q, 78, b),\ \textit{label}(q, 79, c), \dots\}$) and that we need to explicitly encode subsumption testing, rather than, as in Prolog, simply evaluate the query on the knowledge base. We use the convention that if a quantifier for a variable is omitted, then it is universally quantified. Furthermore, we use the convention that variables start with an upper-case and constants with a lower-case character. To illustrate the idea, consider the program in Eq. 1 that has a model if and only if the query q OI-subsumes the graph g. If we remove the last constraint in Eq. 1, we obtain θ-subsumption. Notice that the function θ will be explicit in the model. This program can be executed in IDP directly.

$$
\begin{aligned}
\textit{edge}(q, X, Y) &\implies \textit{edge}(g, \theta(X), \theta(Y)). \\
\textit{label}(q, X, L) &\implies \textit{label}(g, \theta(X), L). \\
X \neq Y &\implies \theta(X) \neq \theta(Y).
\end{aligned}
\tag{1}
$$

Notice that, in theory, it would be possible to simply encode subsumption testing in IDP or ASP as rule evaluation. That is, in the above example, to assert the knowledge base and the rule define *key* and then to ask whether $\textit{key}(g)$ succeeds. Computationally,



(a) Single graph subsumption by a subgraph query ($inq(x)$ in red)

(b) Canonicity check: the subgraph on nodes 1-3-4 is mapped to 1-2-4 (lexicographically smaller)

Fig. 1: Examples of single graph subsumption (left) and of a canonicity check (right)

this would however be infeasible as the sizes of the grounded rules grow exponentially with the number of distinct variables in the rule.

Throughout the paper we use OI-subsumption for all tasks, except for the experimental comparison with Subsumer in Section 5 (since it is not designed to perform OI-subsumption).

Encoding the language bias In practice, one often bounds query languages, for instance by using a bottom clause and only considering queries or clauses that subsume the bottom clause.

Definition 2 (Language bias of a bottom clause $\bot$). *Let $\bot$ be a clause that we call bottom, then the language bias L is a set of clauses: $L = \{c \mid c$ OI-subsumes $\bot\}$.*

This approach also works for ASP. We select one graph from the data and the set of all atoms in that instance will serve as bottom clause $\bot$. When fixing such an instance, we can encode queries by listing the nodes in that entity that will be present in the pattern using the unary predicate $inq(x)$ (for *in query*), as visualized in Figure 1a.

We now present a modification of the previous encoding in Eq. 1 that takes into account the selection of the subgraph of the picked graph q as a query (marked in red in Figure 1a). We refer to the edges in the bottom clause as *bedge* and labels in the bottom clause as *blabel*. When we refer to a node in the bottom clause, we call it a *bnode*.

$$\begin{aligned} inq(X) \wedge inq(Y) \wedge bedge(X,Y) &\implies edge(g, \theta(X), \theta(Y)). \\ inq(X) \wedge blabel(X,L) &\implies label(g, \theta(X), L). \\ inq(X) \wedge inq(Y) \wedge X \neq Y &\implies \theta(X) \neq \theta(Y). \end{aligned} \tag{2}$$

The intuition behind these rules is that we select a subgraph by picking nodes in the bottom clause (i.e., a graph), and then we enforce the constraints on the nodes that have been selected. Eq. 2 implements this by adding an *inq* predicate at the beginning of each clause as a guard, the rule is activated iff the corresponding node is selected.

Encoding the multiple subsumption test In frequent query mining one is interested in mining *frequent* queries, which implies that there is a bag of graphs to match.

Figure 2a illustrates this setup. We can see that we need to test whether the query θ-subsumes each of the graphs in the dataset. To do so, we quantify over a function representing θ, the homomorphism. This makes this formulation second order. Why do we quantify over a function here and not in the previous example? Before the function was quantified existentially since the whole program was asking for a model, which is the same as asking whether the functions exists. Here however, we need a separate function for each graph, since the reasoning process is going to take into account the existence and non-existence of homomorphisms for particular graphs and to reason on top of that.

$$\textit{homo}(G) \iff \exists\theta : \big(\textit{bedge}(X,Y) \wedge \textit{inq}(X) \wedge \textit{inq}(Y) \implies \textit{edge}(G,\theta(X),\theta(Y)).$$
$$\textit{blabel}(X,L) \implies \textit{label}(G,\theta(X),L).$$
$$X \neq Y \implies \theta(X) \neq \theta(Y)\big). \tag{3}$$

This constraint mimics the single graph subsumption test but introduces a new predicate *homo*, indicating a matched graph.

We refer to the group of constraints in Eq. 3 as *Matching-Constraint*. The syntax used above is not yet supported by ASP-solvers such as clasp [9] or IDP [3], but we will argue that adding such second order logic syntax is crucial to enable effective modeling and solving of any structural mining problem. This argument will be backed up by experiments in Section 5.

Encoding the frequency constraint We consider two typical query mining settings: frequent and discriminative query mining. In the frequent setting a query is accepted if it subsumes at least t graphs. In the discriminative setting, each graph in the dataset is labeled as either positive or negative, as indicated by the corresponding predicates *positive*(G) (*negative*(G)) that marks a positive (negative) graph G, and we are interested in queries that match more than t_p graphs with a positive label and do not match more than t_n negatively labeled graphs.

To model the frequent setting we use an aggregation constraint (Eq. 4) and the discriminative setting can be modeled similarly (Eq. 5):

$$|\{G : \textit{homo}(G)\}| \geq t. \tag{4}$$
$$|\{G : \textit{positive}(G) \wedge \textit{homo}(G)\}| \geq t_p \wedge |\{G : \textit{negative}(G) \wedge \textit{homo}(G)\}| \leq t_n. \tag{5}$$

Encoding the canonical form constraint It is well-known since the seminal work of Plotkin [10] that the subsumption lattice contains many clauses that are equivalent, and there has been a lot of work in ILP devoted to avoid the generation of multiple clauses from the same equivalence class (e.g., the work on optimality of refinement operators [11] and many others [12, 13]).

This can be realized in ASP by checking that the current query is not isomorphic to a lexicographically smaller subgraph (using a lexicographic order on the identifiers of the entities or nodes). For example, in Figure 1b, consider the subgraph induced by the nodes 1-3-4 (the numbers are identifiers and the colors are labels). Notice that there is an isomorphic subgraph 1-2-4, i.e., there exists a function θ preserving edges and labels such that a string representation of the latter subgraph is smaller than the former. We call the graph with the smallest lexicographic representation *canonical*. Canonicity can be enforced by the following group of constraints, called *Canonical-Form-Constraint*.

$$
\begin{aligned}
\neg\exists\theta\big(& X \neq Y \implies \theta(X) \neq \theta(Y).\\
& \mathit{inq}(X) \iff \exists Y : \theta(X) = Y.\\
& \mathit{inq}(X) \wedge \mathit{inq}(Y) \wedge \mathit{bedge}(X,Y) \iff \mathit{bedge}(\theta(X),\theta(Y)).\\
& \mathit{inq}(X) \wedge \mathit{blabel}(X) = Y \iff \mathit{blabel}(\theta(X)) = Y.\\
& \mathit{in\theta}(Y) \iff \theta(X) = Y.\\
& d_1(X) \iff \mathit{inq}(X) \wedge \neg \mathit{in\theta}(X).\\
& d_2(X) \iff \mathit{in\theta}(X) \wedge \neg \mathit{inq}(X).\\
& \min(d_1(X)) > \min(d_2(X))\big).
\end{aligned}
\tag{6}
$$

We refer to this group of constraints as *Canonical-Form-Constraint*. It enforces a query to have the smallest lexicographic representation, like the canonical code of mining algorithms [6, 7]. We define a canonical representation in terms of the lexicographic order over the bottom clause node identifiers. If there is a query satisfying the constraints, then there is no other lexicographically smaller isomorphic graph.

Note: Intuition on Eq. 6 The first four rules ensure the existence of a homomorphism under OI-assumption (which can be relaxed by removing a constraint) between the current query on the node in *inq* and another subgraph of the bottom clause. The other rules ensure that the other subgraph has a smaller lexicographic order: *inθ* is an auxiliary predicate that stores nodes of the other subgraph, in d_1 (d_2) there are nodes that only belong to the first, current query, (or second) subgraph. If the minimal node in d_1 (current query) is larger than in d_2 (the other graph), the query is not in canonical form.

We now present additional types of constraints that allow solving variations of the query mining problem.

Connectedness-Constraint As in graph mining [6], we are often only interested in connected queries. The constraint to achieve this consists of two parts: 1) a definition of path and 2) a constraint over path. The first is defined inductively over the bnodes selected by $\mathit{inq}(X)$, the second enforces that any two nodes in the pattern are connected. Note, that the usage of $\{\dots\}$ indicates an inductive definition: the predicate on the left side is completely defined by the rules specified between curly brackets as a transitive closure. If both variables are in the pattern, there must be a path between them.

$$
\begin{aligned}
\{& \mathit{path}(X,Y) \leftarrow \mathit{inq}(X) \wedge \mathit{inq}(Y) \wedge \mathit{bedge}(X,Y).\\
& \mathit{path}(X,Y) \leftarrow \exists Z : \mathit{inq}(Z) \wedge \mathit{path}(X,Z) \wedge \mathit{bedge}(Z,Y) \wedge \mathit{inq}(Y).\\
& \mathit{path}(Y,X) \leftarrow \mathit{path}(X,Y).\}\\
& \mathit{inq}(X) \wedge \mathit{inq}(Y) \wedge X \neq Y \implies \mathit{path}(X,Y).
\end{aligned}
\tag{7}
$$

The Objective-Function An objective function is one way to impose a ranking on the queries. A constraint in the form of an objective function is defined over a model to maximize certain parameters. We consider only objective functions over the queries and matched graphs in the dataset: 1) no objective function, i.e., the frequent query mining problem; 2) maximal size of a query Eq. 8 (in terms of bnodes); 3) maximal coverage Eq. 9 (i.e., the number of matched graphs); 4) discriminative coverage Eq. 10, i.e., the difference between the number of positively and negatively labeled graphs that are covered by a query.

$$|\{X : \mathit{inq}(X)\}| \mapsto \max \tag{8}$$
$$|\{G : \mathit{homo}(G)\}| \mapsto \max \tag{9}$$
$$|\{G : \mathit{positive}(G) \land \mathit{homo}(G)\}| - |\{G : \mathit{negative}(G) \land \mathit{homo}(G)\}| \mapsto \max \tag{10}$$

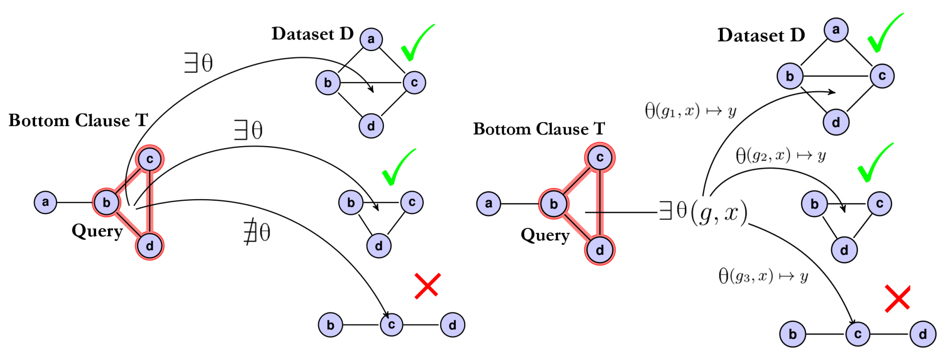
Mining the top-k queries with respect to a given objective function is often a more meaningful task than enumerating all frequent queries, since it provides a more manageable number of solutions that are best according to some function [14].

Topological-Constraint Enforces parts of the bottom clause to be in the query. Let $\mathcal{X}$ be the desired subset of the nodes, then the constraint is $\bigwedge_{X \in \mathcal{X}} \mathit{inq}(X)$.

Cardinality-Constraint This constraint ensures that the size of a graph pattern is at least n (at most, equal) $\exists \circ n\ X : \mathit{inq}(X)$, where $\circ \in \{=, \leq, <, \geq, >\}$.

If-Then-Constraint This constraint ensures that if a node is present in the query, then another node must be present in the query, e.g., a node Y must be present in the query, if a node X is in. We encode this constraint as a logical implication: $\mathit{inq}(X) \implies \mathit{inq}(Y)$.

4 First Order Model



(a) Frequent query mining

(b) First order version of the graph mining model

Fig. 2: Conceptual comparison between first and second order models

In the previous section we have given a positive answer to Question $\mathbf{Q_1}$ using second order logic. However, after formalizing a problem the next step is to actually solve it. In this section we address Question $\mathbf{Q_2}$: can we use existing ASP solvers to model the frequent query mining problem directly? Is it necessary to add constructs to existing modeling languages, or is it possible to write down an efficient and elegant first order logic model for which existing solvers can be used?

To answer Question $\mathbf{Q_2}$, we will encode the problem of frequent query mining as an ASP problem using enumeration techniques. We shall also show how to approach top-k querying mining with this encoding.

The matching and occurrence constraints Since we are restricted to FOL here, we have to encode θ as a binary predicate, adding graph G as a parameter. In the encoding there are five clauses: the first enforces edge preservation; the second enforces that a mapping exists only for the bnodes in the pattern and only for the matched graphs, i.e., $\theta(G, X)$ is a partial function; the third enforces $\theta(G, X)$ to be an injective function on the bnodes for each of the graphs; the fourth enforces label matching; the fifth ensures occurrence frequency (the same as before).

$$\begin{aligned}
&\mathit{homo}(G) \wedge \mathit{inq}(X) \wedge \mathit{inq}(Y) \wedge \mathit{bedge}(X, Y) \Longrightarrow \mathit{edge}(G, \theta(G, X), \theta(G, Y)). \\
&\mathit{homo}(G) \wedge \mathit{inq}(X) \Longleftrightarrow \exists Y : Y = \theta(G, X). \\
&\mathit{homo}(G) \wedge \mathit{inq}(X) \wedge \mathit{inq}(Y) \wedge X \neq Y \Longrightarrow \theta(G, X) \neq \theta(G, Y). \\
&\mathit{homo}(G) \wedge \mathit{inq}(X) \wedge \mathit{blabel}(X, L) \Longrightarrow \mathit{label}(G, \theta(G, X), L). \\
&|\{G : \mathit{homo}(G)\}| \geq t.
\end{aligned} \tag{11}$$

Encoding the model enumeration constraints A common technique in existing clause learning solvers with restarts for generating all solutions, is by asserting for each found solution a model invalidating clause which excludes this model. We use it to denote that certain combinations of nodes in the bottom clause are no longer valid solutions. E.g., if we find a graph on nodes 1-2-3 to be a solution and we store these nodes, then we prohibit this combination of nodes to ensure that the solver will find a new solution.

We present a version of *model invalidation clauses (MIC)* [15] for frequent query enumeration. Our *MIC*s are designed for Algorithm 1, since we enumerate queries of a fixed length: once we increase the length we remove all previous *MIC*s. This allows us to use *MIC*s of a very simple form: given a set of nodes $\mathcal{X}$, we define a constraint $C_{\mathcal{X}}$ as $C_{\mathcal{X}} = \overline{\bigwedge_{X \in \mathcal{X}} \mathit{inq}(X)}$. For example, if a query q on the nodes 1-2-3 is found to be frequent, we would generate the following constraint, so not to generate 1-2-3 again: $\overline{\mathit{inq}(1) \wedge \mathit{inq}(2) \wedge \mathit{inq}(3)}$. Note, that this simple form of model invalidation clauses can only be used if an algorithm iterates over the length of a query and removes *MIC*s once the length is increased.

Anti-monotonicity property It is easy to integrate avoidance of infrequent supergraph generation in Algorithm 1: once a query is established to be infrequent, the corresponding *MIC* is kept, when the query length is increased. However, our experiments in Section 5 indicate that the current computational problems come from a different source and cannot be solved using this property.

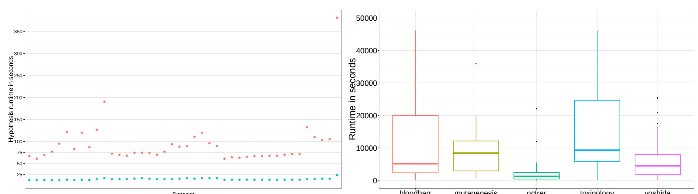
Frequent query enumeration in Algorithm 1 We enumerate all frequent queries starting from the smallest ones (with only 1 node) to the largest ones (the bottom clause). Algorithm 1 has two loops. The first sets the current query size and sets *MIC* to be empty (since we do not want to prohibit generation of supersets of already found queries). In the inner loop, we obtain a candidate for a frequent query by calling IDP once, then we check if the query is in canonical form and also obtain all isomorphic queries to this canonical form. After that we generate a *MIC* for each of them and prohibit the whole isomorphic class of queries to be generated. Note that generating all isomorphic queries is prohibitive, that is why we obtain a canonical query and remove all other isomorphic queries. The algorithm terminates if either it cannot find a new frequent query of any size or the required number of queries has been enumerated.

Algorithm 1 First Order Model: Iterative Query Enumeration

Input: D, k, t ▷ Dataset, #Queries, Threshold
Output: *queries* – set of frequent queries
queries $\leftarrow \emptyset$
$\bot \leftarrow$ *pick-language-bias(D)* ▷ Language bias obtained from data
maxsize $\leftarrow$ *#nodes*$(\bot)$
$i \leftarrow 0$
for *size* $\in 1 \ldots$ *maxsize* **do**
    MICs $\leftarrow \emptyset$
    while True **do**
        query $\leftarrow$ *mine-query*$(D, \bot, t, \textit{size}, \textit{MICs})$ ▷ IDP call, Eq. 11
        if *query* is None **then**
            break ▷ the line below: IDP call, adopted Eq. 6
        canonical, *isomorphic* $\leftarrow$ *get-canonical-and-isomorphic*$(\textit{query}, \bot)$
        queries $\leftarrow$ *queries* $\cup \{\textit{canonical}\}$
        MICs $\leftarrow$ *MICs* $\cup$ *isomorphic*
        $i \leftarrow i + 1$
        if $i \geq k$ **then return** *queries*
return *queries*

Name	Graphs	Vertices	Edges	Labels
Mutagenesis	230	26	27	9
Enzymes	600	33	124	3
Toxinology	417	26	26	22
Bloodbarr	413	21	23	9
NCTRER	232	19	20	9
Yoshida	265	20	23	9

(a) Dataset description (avg for vertices, edges, labels)



(b) Single subsumption test. IDP (red) and Subsumer (blue) (avg time per hypothesis in seconds)

(c) Top-1 runtime distribution (maximal patterns; first order model)

Fig. 3: Dataset description and the summary of subsumption and top-1 experiments

Top-k problem Current ASP solvers, including IDP, can perform optimization described in Constraints 8, 9 and 10. However, Algorithm 1 enumerates patterns with respect to their size and therefore needs to be modified. The key change is to remove the outer *for* loop with the size variable together with *Cardinality-Constraint*. In the experiment section we demonstrate that even top-1 is already excessively complex for modern ASP solvers and requires further investigation and development of the systems.

5 Experiments

In this section we evaluate the encoding on three problems: 1) classical θ-subsumption performed by IDP as encoded in Eq. 1, 2) the first order model in Algorithm 1 on the frequent query mining task, and 3) the first order model on the top-1 query mining task. In all experiments the frequency threshold t is set to 5%. Since the task involves making a stochastic decision, i.e., *pick-language-bias* in Algorithm 1 picks a graph from the dataset at random, this may significantly influence the running time. To resolve this issue, for the graph enumeration problem we average over multiple runs for each dataset: each run involves the enumeration of many queries, i.e., each run generates many data points (runtimes to enumerate N queries). For the top-one mining problem, we present multiple runs for each dataset, since each run computes only one data point (runtime for the top-1 query).

Subsumption We evaluate how the IDP model 1 encoding of θ-subsumption compares with subsumption engine The Subsumer [16]. We used the data from the original Subsumer experiments (transition phase on the subsumption hardness [8, p. 327]) and evaluated IDP subsumption programs and Subsumer on a single hypothesis-example test, i.e., for each hypothesis and example we have made a separate call to IDP and Subsumer to establish subsumption.

The goal of this experiment is to compare how both systems perform if computations are done on a single example. We would like to estimate the potential gain in IDP, if we could specify a homomorphism existence check for each graph independently, like in a higher-order model Eq. 3, i.e., for each graph we would check existence of some $\theta(X)$ instead of checking existence of one function $\theta(G, X)$ like in the first order model.

Figure 3b indicates that IDP and Subsumer perform within a constant bound when we make a separate call for each example and hypothesis. That is, for all but one dataset, the runtimes are within the same order of magnitude for the two methods. If the internal structures of the system are reused and the call is made only for one hypothesis per *set* of examples, we observe a speedup of at least one order of magnitude in Subsumer. This indicates that the system is able to efficiently use homomorphism independence. Once the solver community will have built extensions of systems like IDP to take advantage of homomorphism independence, the resulting systems will perform substantially better than the current ones and their performance will be close to that of special purpose systems such as Subsumer. More precisely, systems should exploit that it is computationally easier to find n functions $\theta(X)$, than one $\theta(G, X)$ for n values of G.

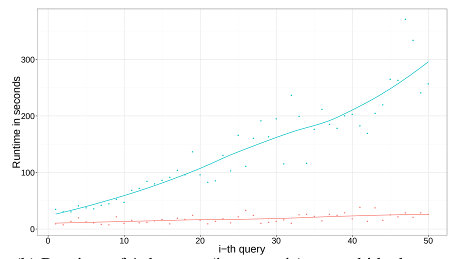
Datasets and implementation We now evaluate the query mining models on a number of well-known and publicly available datasets: Blood Barrier, NCTRER and Yoshida datasets are taken from [17], Mutagenesis and Enzymes datasets are from [18], Toxinology dataset is from [19]. A summary of the dataset properties is presented in Table 3a.

Top-1 *performance evaluation* We present the results of evaluating the FOL model for top-1 mining in Figure 3c. Results were only obtained for the maximal size, and no solutions were computed for Enzymes in a reasonable time (<10h). The discriminative setting cannot be modeled and solved, since we need higher order primitives and for maximal coverage we already experienced an explosion of the search space. This experiment demonstrates that current satisfiability solvers cannot effectively perform search on both categories of query variables and homomorphism coverage; solver extensions are necessary. The first extension is to specify independence of the homomorphisms in the model, i.e., higher order primitives as indicated in the model Eq. 3, the second is to give more control over the solver's decisions, e.g., by allowing to specify a decision order.

Frequent query mining The goal of the next experiment is to estimate the potential gain of the introduction of a higher order modeling construction. To do this, we mimicked the higher order behavior in Algorithm 1 by making several calls to IDP (one per graph) in the method *mine-query*. We call this mimicking model *decomposed*. The results of the evaluation are presented in Figure 4a, a comparison of the decomposed and the original FOL model are in Figure 4b. Homomorphism dependence in the first order model causes serious computational difficulties, which makes higher order primitives one of the main priorities for solving query mining. The runtime also grows with the size of a pattern since it affects the search space and runtime as a result.

dataset	**model**	1	5	10	25	50
mutagenesis	FOL	35s	2m25s	7m22s	23m22s	1h29m
	decomposed	**17s**	**43s**	**84s**	**3m50s**	**9m26s**
nctrer	FOL	30s	3m24s	7m17s	24m28s	1h27m
	decomposed	**5s**	**35s**	**1m16s**	**6m0s**	**9m20s**
yoshida	FOL	35s	2m53s	6m34s	31m13s	1h56m
	decomposed	**9s**	**1m1s**	**2m0s**	**5m48s**	**15m8s**
bloodbarr	FOL	1m4s	6m26s	15m51s	1h47m	8h2m
	decomposed	**9s**	**57s**	**2m3s**	**9m55s**	**14m52s**
toxinology	FOL	1m21s	8m20s	22m5s	2h42m	16h3m
	decomposed	**11s**	**56s**	**2m35s**	**7m54s**	**18m5s**
enzymes	FOL	7m3s	45m48s	4h0m	—	—
	decomposed	**15s**	**1m27s**	**2m57s**	**13m26s**	**38m16s**

(a) Frequent query enumeration time of N queries



(b) Runtime of i-th query (in s; y-axis) on yoshida dataset (x-axis is the query index i)

Fig. 4: Frequent query enumeration FOL (blue) vs decomposed model (red)

Summary of the experimental results The results show that the model performs reasonably and we therefore conclude that it can be considered a first step towards the development of declarative languages for relational query mining. From Figure 3b and 4a it is clear that the core computational difficulty lies in the inability to state homomorphism independence. This follows from the candidate generation and canonicity check runtime and from the speedup that Subsumer has when it is applied to the whole set of examples with one call. After comparing the performance of IDP and Subsumer on a single example and observing the speed up of the decomposed model in Figure 4a, one would expect a significant speedup if it were implemented within the solver.

We have observed in Figure 3c that query mining introduces interesting computational challenges and might be of interest to the solver development community as well as to the ILP community. We also pointed out the reasons for these computational difficulties and suggested possible ways to enhance the solvers.

6 Model discussion and generalization

Advantages and disadvantages of the model There is a number of advantages of the declarative approach to pattern mining compared to the classical imperative methods: compact and clear representation; extendability and generality; provability and formal semantics of the models; and reliable and portable implementation (the solvers are developed by the community, well tested and already applied to a variety of tasks; the solvers are available for all popular operation systems: Linux, MacOS, and Windows).

In particular, our model compactly represents not only the model of the graph mining problem but also the code that can be executed to solve the problem. Adding the source code of gSpan [20] to a paper ($\sim$ 2000 lines of C++) would be impossible. Also, the model allows easy extension by adding constraints to the theory, e.g., adding a constraint to handle labels on edges is just an extra line with a straightforward logical formula. Last but not least our formulation allows formal rigorous reasoning on the constraints that constitute the model.

These advantages come, of course, at a cost. Typically, specialized algorithms perform an order of magnitude (at least) faster. The main reason for such a speed up is that most mining algorithms incorporate and heavily rely on the structure of the problem in the search strategy. Our experiments in Figure 4b demonstrate that the declarative approach can reduce the runtime gap by extending the language to better incorporate the structure of a problem. Also, modeling this kind of mining problems influences the way declarative solvers are built and potentially can lead to better solving system that would perform reasonable on mining problems.

Generalization of the approach Let us demonstrate how the model can be extended to handle labels on the edges as an example of how declarative systems can be adapted to solve new tasks.

Assume that edge labels are represented using the predicate $\textit{edgelabel}(g, e_1, e_2, l)$ and the edge labels of the bottom clause are stored in the predicate $\textit{tedgelabel}(e_1, e_2, l)$. Then, the first order formulation Eq. 11 can be extended by adding the constraint:

$$\textit{inq}(X) \wedge \textit{inq}(Y) \wedge \textit{tedgelabel}(X, Y, L) \implies \textit{edgelabel}(G, \theta(G, X), \theta(G, Y), L).$$

Intuitively, this constraint ensures that if there is an edge (X, Y) with a label L in the bottom clause, then there is an edge with a label L in the graph G. If each edge has a label, the constraint above can replace the first constraint in Eq. 11.

The smallest change, according to the principle, is the addition of a rule or a fact. In general, this example demonstrates how our declarative approach and our model in particular implements the elaboration tolerance principle [21], i.e., a small change in the problem formulation should lead to a small change in the model. The smallest changes, according to the principle, is the addition of a rule or a fact. With this respect our model satisfies the elaboration tolerance principle and can be called an additive elaboration.

7 Related work

WARMR [4] and FARMER [6] are extensions of the Apriori algorithm for discovering frequent structures in multiple relations. Even though they use ILP techniques (e.g., a declarative language bias) to determine frequent queries, they imperatively specify computations and the algorithm does not support the addition of arbitrary constraints due to the restricted nature of the algorithm, and hence focuses on a specific task. B-AGM, Biased-Apriori-based Graph Mining [5], is another system based on Apriori. C-Farmr [7] is an ILP system for frequent Datalog clauses that uses so-called condensed representations to avoid the generation of semantically equivalent clauses.

The XHAIL system [22] uses ASP as a computational engine to perform abductive inductive reasoning. It is similar in the way we use an ASP engine as computational core, but XHAIL is focused on the abduction task, whereas we focus on query mining which always involves aggregation and model enumeration.

In the work on sequence testing [23] ASP is used as a subroutine in a cycle and also model projection on a predicate is also present similarly to our Algorithm 1, but this similarity is technical, since the tasks are of different nature. ASP has also been applied to itemset mining [24] and sequence mining [25]; the presented results are preliminary, the suggested methods only deal with one particular task in its basic formulation.

gSpan [20] is a specialized algorithm designed to solve the frequent graph pattern mining problem. The algorithm is tailored to solve only one task exclusively and requires significant changes in its core to be extended to solve other mining tasks (and, to the best of our knowledge, there is no extension to the full relation setting). It is similar, however, in the computational challenges: canonicity checks, homomorphism checks, language bias, etc.

8 Conclusions

We have shown that modern ASP solvers can be applied to ILP query mining tasks. We have provided experimental evidence that these models can be used as prototypes for developing declarative mining languages. We have also indicated the reasons why the solvers need to be extended to make computation efficient and proposed concrete extensions as well as estimated their potential effectiveness. The query mining models and the experimental setup we developed provide an interesting challenge for the ASP solver developer community and a potentially useful tool for the ILP community.

References

[1] Tias Guns et al. "MiningZinc: A Modeling Language for Constraint-Based Mining". In: *IJCAI*. 2013.

[2] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. "Answer Set Programming: A Primer". In: *5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone, Italy, August 30 – September 4, 2009*. Vol. 5689. 2009.

[3] Broes De Cat et al. "Predicate Logic as a Modelling Language: The IDP System". In: *CoRR* abs/1401.6312 (2014).

[4] Ross D. King, Ashwin Srinivasan, and Luc Dehaspe. "Warmr: a data mining tool for chemical data." In: *Journal of Computer-Aided Molecular Design* 15.2 (2001), pp. 173–181.

[5] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. "An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data". In: 2000, pp. 13–23.

[6] Siegfried Nijssen and Joost N. Kok. "Efficient frequent query discovery in FARMER". In: *In Proc. of the 7th PKDD, volume 2838 of LNCS*. 2003, pp. 350–362.

[7] Luc De Raedt and Jan Ramon. "Condensed Representations for Inductive Logic Programming". In: *KR*. 2004, pp. 438–446.

[8] Luc De Raedt. *Logical and Relational Learning: From ILP to MRDM (Cognitive Technologies)*. Springer-Verlag New York, Inc., 2008.

[9] Martin Gebser et al. "*clasp* : A Conflict-Driven Answer Set Solver". In: *LPNMR*. 2007, pp. 260–265.

[10] G. D. Plotkin. "A further note on inductive generalization". In: volume 6 (1971), pages 101–124.

[11] S. Muggleton. "Inverse Entailment and Progol". In: *New Generation Computing, Special issue on Inductive Logic Programming* 13.3-4 (1995), pp. 245–286.

[12] Patrick R.J. van der Laag and Shan-Hwei Nienhuys-Cheng. "Completeness and properness of refinement operators in inductive logic programming". In: *The Journal of Logic Programming* 34.3 (1998), pp. 201 –225.

[13] Jens Lehmann and Pascal Hitzler. "Foundations of Refinement Operators for Description Logics". In: *Inductive Logic Programming*. Vol. 4894. 2008, pp. 161–174.

[14] Jilles Vreeken, Matthijs Leeuwen, and Arno Siebes. "Krimp: Mining Itemsets That Compress". In: *Data Min. Knowl. Discov.* 23.1 (2011), pp. 169–214.

[15] Broes De Cat. “Separating Knowledge from Computation: An FO(.) Knowledge Base System and its Model Expansion Inference”. PhD thesis. KU Leuven.

[16] Jose Santos and Stephen Muggleton. “Subsumer: A Prolog theta-subsumption engine”. In: *ICLP Technical Communications*. Vol. 7. 2010, pp. 172–181.

[17] Ulrich Rückert and Stefan Kramer. “Optimizing Feature Sets for Structured Data”. In: ECML ’07. Warsaw, Poland, 2007, pp. 716–723.

[18] Asim Kumar Debnath et al. “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity”. In: *Journal of medicinal chemistry* 34.2 (1991), pp. 786–797.

[19] Christoph Helma and Stefan Kramer. “A Survey of the Predictive Toxicology Challenge 2000-2001.” In: *Bioinformatics* 19.10 (2003), pp. 1179–1182.

[20] Xifeng Yan and Jiawei Han. “gSpan: Graph-Based Substructure Pattern Mining”. In: ICDM ’02. 2002.

[21] John McCarthy. “Elaboration Tolerance”. In: (1999).

[22] Oliver Ray. “Nonmonotonic Abductive Inductive Learning”. In: *Journal of Applied Logic* (2008).

[23] Esra Erdem et al. “Answer-Set Programming as a New Approach to Event-Sequence Testing”. In: *Proc. of the 3rd International Conference on Advances in System Testing and Validation Lifecycle*. 2011.

[24] Matti Jrvisalo. “Itemset Mining as a Challenge Application for Answer Set Enumeration”. In: *LPNMR’11*. 2011, pp. 304–310.

[25] Thomas Guyet, Yves Moinard, and René Quiniou. “Using Answer Set Programming for pattern mining”. In: *CoRR* abs/1409.7777 (2014).

A Appendix: Introduction to IDP

Listing 1.1: IDP source code example – map coloring

```
vocabulary V{
  type area
  type color
  border(area,area)
  coloring(area):color
}
theory T:V{
  // Adjacent countries can not have the same color
    ∀A1 A2 : border(A1, A2) ⟹ coloring(A1) ≠ coloring(A2).
}
structure S:V{
  area={belgium; holland; germany; luxembourg; austria; swiss;france }
  color={blue;red;yellow;green}
  border={
   (belgium,holland);(belgium, germany);(belgium,luxembourg);(belgium,france);
   (holland,germany);(germany,luxembourg);(germany,austria);(germany,swiss);
   (germany,france);(luxembourg,france);(austria,swiss);(swiss,france)
  }
}
```

The IDP language [3, 15] is an extension of first order logic with inductive definitions and aggregration. The IDP system implements finite satisfiability and can be considered as an ASP system.

The particular type of inference we use in our work is *model expansion*. The task of model expansion is to expand a finite interpretation S for the subvocabulary V of a given logic theory T to a model of T. In the example above, V is the vocabulary of the map colouring problem i.e. *area*, *color*, *border*$(area, area)$, *coloring* $::$ *area* $\mapsto$ *color*; S consists of 7 countries, 4 colours and a border relation between the countries; T is the constraint that two bordering countries cannot have the same color.

In this example, the model expansion task is to find an extension of S, i.e. *coloring* function, such that the constraint in T is satisfied, i.e. all bordering countries have different colours.

The example can be tried online (select file "Map Colouring"):

adams.cs.kuleuven.be/idp/server.html.