Noname manuscript No.
(will be inserted by the editor)

Finding Relational Redescriptions

Esther Galbrun · Angelika Kimmig

Received: date / Accepted: date

Abstract We introduce relational redescription mining, that is, the task of finding two structurally different patterns that describe nearly the same set of object pairs in a relational dataset. By extending redescription mining beyond propositional and real-valued attributes, it provides a powerful tool to match different relational descriptions of the same concept.

We propose an alternating scheme for solving this problem. Its core consists of a novel relational query miner that efficiently identifies discriminative connection patterns between pairs of objects. Compared to a baseline Inductive Logic Programming (ILP) approach, our query miner is able to mine more complex queries, much faster. We performed extensive experiments on three real world relational datasets, and present examples of redescriptions found, exhibiting the power of the method to expressively capture relations present in these networks.

1 Introduction

With the increasing amount of data available from heterogenous sources nowadays, establishing links between different perspectives on the same concept becomes ever more important, as recognized, for instance, in schema matching and ontology alignment for the semantic web (Shvaiko and Euzenat, 2005) and can contribute to the discovery of patterns in knowledge bases (Galárraga et al, 2013). One way of creating such links is to find sets of objects together with their descriptions in different terminologies, as done in *redescription mining* (Ramakrishnan et al, 2004; Galbrun and Miettinen, 2012). However, this technique has so far only considered propositional or real valued attributes. In this paper, we extend redescription mining to the relational or network-based setting. In other words, we consider the task of finding two structurally different patterns that describe nearly the same set of object pairs in a relational dataset. We focus on a restricted language (binary

Department of Computer Science and Helsinki Institute for Information Technology HIIT, PO Box 68, FI-00014 University of Helsinki, Finland E-mail: esther.galbrun@cs.helsinki.fi · Departement Computerwetenschappen, KU Leuven, Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium E-mail: angelika.kimmig@cs.kuleuven.be

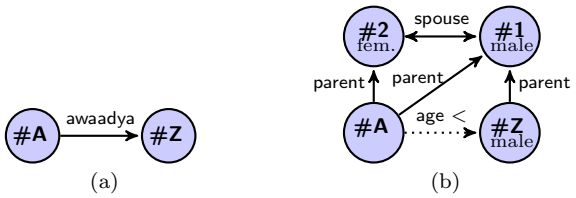


Fig. 1 Example of redescription from the `Kinship` dataset: (a) kinship relation *Awaadya* between $(\#A, \#Z)$ and (b) corresponding genealogical connection. The description formalism is introduced in Section 2.

relations with object identity) amenable to efficient graph mining techniques, resulting in an exploratory method that relies primarily on occurrence information and requires no extensive background knowledge such as a declarative bias.

Consider the following example from the `Kinship` dataset, which provides information about kinship terminology and family relationships within an Australian indigenous community (cf. Section 6). In Figure 1, graph (a) represents the kinship relation *Awaadya* between the speaker $\#A$ and another person $\#Z$, corresponding to the relation between a child and his older brother, as given by graph (b). These two graphs are alternative ways to describe the same pairs of individuals $(\#A, \#Z)$ and hence form a redescription.

Given a dataset and a query language, the underlying principle of redescription mining is to find pairs of queries, i.e. descriptions, that are structurally different yet describe (nearly) the same entities. Propositional redescription mining considers as its entities single objects characterized by their individual properties. Instead, in this novel relational setting, entities consist of pairs of objects characterized by both relations linking them and individual properties.

To find pairs of descriptions, redescription mining can adopt an alternating approach: one description is fixed, the other one is updated, and roles are swapped in the next iteration (Ramakrishnan et al, 2004). Following this approach, we present ARRM, an algorithm for Alternating Relational Redescription Mining.

Our alternating scheme relies on an approach to finding relational descriptions, such as relational query mining (Dehaspe and Toivonen, 1999; De Raedt and Ramon, 2004). However, the generate-and-test approach of query mining systems requires large numbers of expensive coverage tests based on subgraph isomorphism. More importantly, they typically do not ensure that patterns connect the nodes of interest, thus producing many patterns that do not correspond to redescriptions. Hence, we propose an efficient algorithm, which we call FPQM, that finds descriptions for a given set of example pairs. It first mines for path patterns that connect many example pairs, then combines those into more expressive graph patterns. This reduces the number of coverage tests needed by constructing queries based on the data.

A comparison of FPQM to a baseline ILP tool on real world data shows that our approach can identify complex descriptions matching known ones, and is much faster. Examples of results obtained with ARRM illustrate the power of the method to capture the relations present in a network. Compared to individual relational queries, the added expressivity brought by redescriptions permits to better elucidate different uses of the same predicate and find more accurate patterns.

This paper extends upon our previous work (Galbrun and Kimmig, 2012). As our main contributions, we define Relational Redescription Mining, present novel algorithms to solve this task and provide an experimental evaluation of the proposed approach. Compared to the earlier version, we improved the combination and selection procedures of the query mining algorithm and realized an alternating scheme to address the full relational redescription mining problem. Also, we considered two additional datasets in our experiments and performed an extensive evaluation of the alternating scheme, including comparisons to several baselines.

We proceed as follows. Section 2 introduces relational redescription mining, Section 3 discusses related work, Sections 4 and 5 present the proposed path-based relational query miner and alternating scheme for relational redescription mining, respectively. Both are then experimentally evaluated in Section 6, before we conclude in Section 7.

2 Definitions and Notations

This paper introduces *relational redescription mining*, the task of finding two structurally different queries that describe nearly the same set of object pairs in a relational dataset. Informally, we view descriptions as connected graphs expressed in terms of attributes of the data. For instance, graph (b) in Figure 1 is an example of such descriptions for the objects of interest $\#A$ and $\#Z$ in terms of relations and attributes `spouse`, `parent`, `male`, `female` and $\texttt{age}^{<}$. We now introduce the concepts required for a more formal definition of the problem. We focus on binary relations, as these can be represented in the form of graphs, allowing us to base our algorithms on graph concepts.

We view *relational data* as a directed graph $(\mathcal{O}, \mathcal{R})$, where nodes correspond to the objects $\mathcal{O}$, and edges to relations $\mathcal{R}$ between them. Two families of functions, $\mathcal{N}$ and $\mathcal{E}$, label nodes and edges with their *attributes*, respectively.

For instance, in the kinship domain, $\mathcal{O}$ is the set of individuals from the community. Node attributes are $\mathcal{N} = \{\texttt{sex}, \texttt{age}\}$ and edge attributes $\mathcal{E} = \{\texttt{kin}, \texttt{gen}\}$, where `kin` maps into the set of kinship terms and the values of `gen` are the genealogical relations `parent` and `spouse`.

From node and edge attributes, we obtain three types of Boolean functions, or predicates, that serve as basic building blocks of queries. The first type, a *node predicate* $\nu^{v}_{N_i}(o)$, is true for an object o if and only if the node label $N_i(o)$ is defined and takes value v. The second type, an *edge predicate* $\epsilon^{u}_{E_i}(o_1, o_2)$, is true for a pair of objects (o_1, o_2) if and only if the edge label $E_i(o_1, o_2)$ is defined and takes value u. The third type, a *comparison predicate* $\phi^{rel}_{N_i}(o_1, o_2)$ for a binary relation rel over the range of node labeling function N_i is true for a pair of objects (o_1, o_2) if and only if both node labels $N_i(o_1)$ and $N_i(o_2)$ are defined and $rel(N_i(o_1), N_i(o_2))$ holds.

As an example, graph (b) in Figure 1 uses, among others, node predicates $\nu^{\texttt{male}}_{\texttt{sex}}(\#1)$ and $\nu^{\texttt{female}}_{\texttt{sex}}(\#2)$, edge predicates $\epsilon^{\texttt{parent}}_{\texttt{gen}}(\#A, \#1)$ and $\epsilon^{\texttt{spouse}}_{\texttt{gen}}(\#1, \#2)$, and comparison predicate $\phi^{<}_{\texttt{age}}(\#A, \#Z)$.

For an object o, the set $F_N(o)$ of its *node features* contains the node predicates that hold true for that object. For a pair of objects (o_1, o_2), the sets $F_E(o_1, o_2)$ and $F_C(o_1, o_2)$ of *edge* and *comparison features* contain the edge and comparison predicates that hold true for that pair, respectively. Note that the data, or network,

is fully specified by the features of all objects, which implicitly provide all relevant information about the objects and their relations and attributes.

A *graph query* is a definite clause of the form $q(X, Y)$:- $b_1, \ldots, b_n$, where the body elements b_i are node, edge or comparison predicates, q is a special predicate denoting the pattern and the *query variables* X and Y in the head also occur in the body. Instantiations of query variables are the object pairs of interest. We require graph queries to be *linked*, meaning that the set of edge predicates in the body connects the query variables. That is, a query is linked if there exists a sequence of variables $Z_0, \ldots, Z_k$ with $Z_0 = X$, $Z_k = Y$, such that for all $i = 1, \ldots, k$, there is an index j such that $b_j \in F_E(Z_{i-1}, Z_i) \cup F_E(Z_i, Z_{i-1})$. A *path query* is a graph query whose query variables are connected by an acyclic path consisting of all edge predicates in the body. We denote the set of attributes for which the body of query q contains predicates by $\mathrm{att}(q)$.

In the remainder of this paper, we use $\#A$ and $\#Z$ to denote the query variables and $\#1$, $\#2$, *et cetera* to denote any other intermediate variables. Node attributes are indicated inside the node under the identifier, using lowercase, as for the edge attributes also. For instance, graph (b) in Figure 1 corresponds to the graph query

$$\begin{aligned} q_b(\#A, \#Z) \text{ :- } & \epsilon^{\texttt{parent}}_{\texttt{gen}}(\#A, \#1), \epsilon^{\texttt{parent}}_{\texttt{gen}}(\#A, \#2), \nu^{\texttt{male}}_{\texttt{sex}}(\#1), \\ & \epsilon^{\texttt{spouse}}_{\texttt{gen}}(\#1, \#2), \epsilon^{\texttt{spouse}}_{\texttt{gen}}(\#2, \#1), \nu^{\texttt{female}}_{\texttt{sex}}(\#2), \\ & \epsilon^{\texttt{parent}}_{\texttt{gen}}(\#Z, \#1), \phi^{<}_{\texttt{age}}(\#A, \#Z), \nu^{\texttt{male}}_{\texttt{sex}}(\#Z). \end{aligned}$$

This query has attribute set $\mathrm{att}(q_b) = \{\texttt{sex}, \texttt{gen}, \texttt{age}\}$ and is linked due to the spouse and parent edges. Note that the $\texttt{age}^{<}$ edge in the graphical representation corresponds to a comparison predicate and is thus not considered for linkage and represented with a dotted line.

As common in graph mining, we use subgraph isomorphism or, in terms of logic, θ_{OI}-subsumption (Esposito et al, 1994), to match queries against the data graph. That is, each variable in the query has to be matched to a different node in the graph, respecting the predicates in the query body. This choice is motivated by the intuitiveness and interpretability of the resulting queries and the ease of search. We denote such a match of variables V_j to objects o_{i_j} by the corresponding substitution $\theta = \{V_1/o_{i_1}, \ldots, V_n/o_{i_n}\}$; θ reduced to query variables is called *answer substitution*. The set of all (distinct) answer substitutions of query q on the given network is its support, $\mathrm{supp}(q)$. For instance, for query q_b above, the support contains all pairs of nodes (n_a, n_z) such that when matching $\#A$ to n_a and $\#Z$ to n_z, there is at least one match of $\#1$ and $\#2$ to other nodes that satisfies the query body.

For simplicity, we use the closed world assumption throughout this work. In other words, given a set of positive example pairs, all remaining pairs are considered to be negative examples. Altering the functions for scoring and filtering the queries presented in Section 4.3 allows to modify this assumption.

For a given set of example pairs E^+ and query q, we denote the sets of true positives (example pairs covered), false positives (other pairs covered) and false negatives (example pairs not covered) by $E_{1,1} = \mathrm{supp}(q) \cap E^+$, $E_{0,1} = \mathrm{supp}(q) \setminus E^+$, and $E_{1,0} = E^+ \setminus \mathrm{supp}(q)$, respectively.

A redescription is a pair of queries $R = (q_{\mathbf{L}}, q_{\mathbf{R}})$. Extending the previous notation, let $E_{1,0}$ be the set of entities (i.e. object pairs) which support only the first query (i.e. $E_{1,0} = \mathrm{supp}(q_{\mathbf{L}}) \setminus \mathrm{supp}(q_{\mathbf{R}})$), $E_{0,1}$ those which support only

the second query and $E_{1,1}$ those which support both queries. The *accuracy* of a redescription is commonly measured using the Jaccard coefficient, that is,

$$\mathrm{J}(q_{\mathbf{L}}, q_{\mathbf{R}}) = \frac{|\mathrm{supp}(q_{\mathbf{L}}) \cap \mathrm{supp}(q_{\mathbf{R}})|}{|\mathrm{supp}(q_{\mathbf{L}}) \cup \mathrm{supp}(q_{\mathbf{R}})|} = \frac{|E_{1,1}|}{|E_{1,0} + E_{0,1} + E_{1,1}|}.$$

This measure takes values in the unit interval without using the number of all existing pairs of nodes for scaling, which would result in practically undistinguishable values. Furthermore, $E_{1,0}$ and $E_{0,1}$ are weighted equally, in agreement with the symmetric view of redescription mining.

In order to avoid trivial redescriptions in the form of queries that are small variants of each other, we impose a syntactic requirement on the queries, which need to have disjoint attribute sets. While it would also be possible to require the use of attributes from two disjoint vocabularies, such a fixed split may not always be easily specified upfront. If available, such a split can be incorporated into the disjointness condition.

Given this background, we define *relational redescription mining* as follows:

Problem 1 (Relational Redescription Mining) ***Given*** *a relational dataset in the form of node, edge and comparison features* $\{F_N, F_E, F_C\}$ *and an accuracy threshold* j, ***find*** *redescriptions* $(q_{\mathbf{L}}, q_{\mathbf{R}})$ *such that* $\mathrm{att}(q_{\mathbf{L}}) \cap \mathrm{att}(q_{\mathbf{R}}) = \emptyset$ *and* $\mathrm{J}(q_{\mathbf{L}}, q_{\mathbf{R}}) \geq j$.

For simplicity of exposition, we restricted our discussion to queries of arity two. However, the definitions extend naturally to queries of higher arity, that is, queries whose body consists of binary predicates but with more than two variables occuring in their head.

3 Related Work

Redescription mining emphasizes the insights obtained from expressive, interpretable patterns and their instances in the given data rather than prediction over unseen data. Relational pattern languages are thus a natural candidate for redescriptions, but existing approaches have focused on propositional features (Ramakrishnan et al, 2004) and real-valued attributes (Galbrun and Miettinen, 2012). They operate on matrices with a row for every object in the data and a column for every attribute or feature. Our ARRM algorithm follows an alternating scheme similarly to the former method.

These approaches have been shown to find simple redescriptions of single nodes in a bibliographic network, describing a researcher either in terms of the conferences he publishes at, or in terms of his co-authors, that is, using only attributes in the form of the labels of neighboring nodes (Gallo et al, 2008; Galbrun and Miettinen, 2012). However, as we illustrate in Section 6, considering pairs of objects and features based on their connections in terms of complex longer distance relations would inflate the size of the matrix and thus the search space of the algorithm. In contrast, our relational approach dynamically adapts the feature space to the subtask at hand, allowing for more focused exploration of connection patterns.

Learning relational queries is a key goal in Inductive Logic Programming (ILP) and multi-relational data-mining, and a central component of our relational redescription mining scheme. Multi-relational query miners often use a refinement operator to extend frequent queries found at the previous level, typically by adding

a literal with at least one already used variable to the end of the clause body (Dehaspe and Toivonen, 1999; De Raedt and Ramon, 2004). While this principle results in connected clauses for unary patterns, patterns of higher arity are likely to ignore some of the query variables, or to contain disconnected components around individual query variables, and thus fail to provide insight into the relations between them. This connectivity problem has been addressed by relational pathfinding (Richards and Mooney, 1992; Ong et al, 2005) and function learning (Santos et al, 2009). Pathfinding refines clauses by adding a sequence of literals if no single literal is able to connect query variables, where candidate sequences are generated based on connections of a single example's query variables in the data rather than by enumerating abstract paths. Function learning avoids evaluating unconnected queries by generating candidate queries from individual examples. The queries in our method are similarly anchored in the data, but are directly selected based on their frequency across all examples, in an approach inspired by graph mining techniques (Yan and Han, 2002). Furthermore, to reduce the search-space, logic-based methods typically make heavy use of declarative bias, which needs to be provided by the user. This is not the case with our method, where the search space is pruned using solely occurrence information computed on the dataset.

Relational patterns play an important role in various techniques that explore and analyze relational datasets. For instance, the aim of subgroup discovery (Wrobel, 1997; Lavrac et al, 2002) is to identify groups of objects that differ from the overall population in an interesting way. There, individual objects are described in terms of the relations they participate in. In contrast, relational redescription mining is interested in finding pairs of queries that describe different connection patterns between two objects. In the context of making predictions based on relational patterns, query mining has also been extended to learn association rules with conjunctive heads (Goethals and Van den Bussche, 2002), which can be seen as associations between conjunctive redescriptions, and to flexible numbers of query variables (Goethals et al, 2005). Yet, the type of rules mined is fairly restrictive compared to the descriptions considered in our approach. These complex association rules are similar to the tuple-generating dependencies used in schema mapping and data exchange. The CLIO system learns such mappings by exploiting relational dependencies in the two schemata, where the user indicates some correspondences between attributes, as input (Miller et al, 2000). Our work uses shared objects to determine attribute correspondences, and does not rely on explicit information on how relations can be combined. Aligning or mapping objects can also be considered part of the overall discovery process, as for instance in the PARIS approach to ontology alignment (Suchanek et al, 2011), which discovers correspondences on the level of both instances and schemas. But as most other approaches to schema matching (Shvaiko and Euzenat, 2005), PARIS focuses on one-to-one mappings of relations and does not consider more complex queries. A recent exception is the work of Zhang et al (2012), who nevertheless only consider paths up to length four and rely on approximation schemes for Markov Logic inference to keep the approach feasible.

Recently, there is increasing interest in large knowledge bases like DBpedia (Auer et al, 2007), NELL (Carlson et al, 2010) or YAGO (Suchanek et al, 2007), which store binary relations between millions of objects. Such massive amounts of incomplete and often noisy information are a challenge for most existing relational learning approaches, and call for adapted representations and learning methods (Ret-

Input: A network with a set of positive examples E^+, a frequency threshold γ, an extension threshold κ, a contribution threshold δ and a set of quality criteria Γ.
Output: A set of relational queries Q.

1: $Q \leftarrow \emptyset$; $D \leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \emptyset$
2: $C \leftarrow \textsc{FrequentPQ}(E^+, \gamma, \kappa)$
3: **for** $e \in E^+$ **do**
4: $\quad a \leftarrow \textsc{AlignedBottomClause}(e, C)$
5: $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{(e, \textsc{ToTransaction}(a))\}$
6: **for** $f \in \text{FIM}\textsc{Graphs}(\mathcal{T}, \gamma)$ **do**
7: $\quad q \leftarrow \textsc{computeQuery}(f)$
8: $\quad$ **if** q is acceptable according to Γ **then**
9: $\quad\quad D \leftarrow D \cup \{q\}$
10: **for** $q \in D$ ordered according to Γ **do**
11: $\quad$ **if** $|E_{1,1}(q) \setminus \mathcal{S}| \geq \delta$ **then**
12: $\quad\quad Q \leftarrow Q \cup \{q\}$
13: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup E_{1,1}(q)$
14: **return** Q

Fig. 2 FpQm: Frequent-paths based relational query miner. Details on FrequentPQ are provided in Figure 3 and discussed in Section 4.1; ToTransaction is discussed in Section 4.1, AlignedBottomClause, FIMGraphs and computeQuery in Section 4.2.

tinger et al, 2012), such as the graph based techniques used in our redescription miner. Nebot and Llavori (2012) propose to extract relational association rules from a medical ontology by applying frequent itemset mining. Their method, however, requires extensive expert knowledge. In the context of NELL, weighted combinations of path patterns, learnt based on random walks, have been used for retrieval tasks (Lao et al, 2011; Lao and Cohen, 2010). In contrast to our approach, paths are not combined into graphs, but are instead weighted by their importance. Galárraga et al (2013) introduce a fast association rule mining approach that, as our work, is inspired by ILP techniques, but tailored towards binary relations, in this case using a database with aggressive indexing to speed up querying.

4 The Frequent-Paths Based Relational Query Miner

Before introducing the alternating relational redescription mining algorithm in Section 5, we now discuss its core component, FpQm, a novel relational query miner based on frequent paths. Given a network, FpQm aims to find the best graph queries with respect to quality criteria Γ that discriminate a set of positive object pairs E^+. The specific criteria in Γ are discussed in Section 4.3.

An outline of the FpQm algorithm is presented in Figure 2. The three-phase approach considers only linked queries and limits the number of costly subgraph isomorphism-based coverage tests. First, the path queries that cover at least a given number of positive examples are mined (line 2). Second, such path queries are combined to construct graph queries (lines 3–9). Finally, the best describing among these graph queries are selected and returned (lines 10–13). We now discuss each of these steps in turn.

Input: A network with a set of positive example object pairs E^+, a frequency threshold γ, and an extension threshold κ.
Output: A set of frequent paths queries C.

1: $k \leftarrow 0$
2: $\mathcal{P}_k \leftarrow$ paths of length 0, i.e. starting nodes in E^+
3: **while** $\mathcal{P}_k \neq \emptyset$ **do**
4: $\quad k \leftarrow k+1$; $\mathcal{P}_k \leftarrow \emptyset$; $\mathcal{T} \leftarrow \emptyset$
5: $\quad$ **for** each $P' \in \mathcal{P}_{k-1}$ **do** ▷ $P[i]$ is node at position i in path P
6: $\quad\quad$ **for** each $n \in neighbors(P'[k-1])$ **do**
7: $\quad\quad\quad P \leftarrow P'$
8: $\quad\quad\quad$ **if** $n \notin P$ **then**
9: $\quad\quad\quad\quad P[k] \leftarrow n$
10: $\quad\quad\quad\quad \mathcal{P}_k \leftarrow \mathcal{P}_k \cup \{P\}$
11: $\quad\quad\quad\quad$ **if** $(P[0], P[k]) \in E^+$ **then** ▷ example pair connected
12: $\quad\quad\quad\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{((P[0], P[k]), \textsc{ToTransaction}(P))\}$
13: $\quad F \leftarrow \textsc{FIMPaths}(\mathcal{T}, \gamma)$
14: $\quad C \leftarrow C \cup F$
15: $\quad E_k \leftarrow E_{k-1} \cup \bigcup_{f \in F} E_{1,1}(f)$
16: $\quad$ **if** $k > \kappa$ and $E_{k-\kappa} = E_k$ **then** ▷ no new example pair covered for κ steps
17: $\quad\quad \mathcal{P}_k \leftarrow \emptyset$
18: **return** C

Fig. 3 FrequentPQ: Mining frequent path queries from a network. Details on ToTransaction and FIMPaths are provided as part of the discussion in Section 4.1.

4.1 Mining Frequent Path Queries

The first phase of FPQM (line 2) finds the set of linked path queries that are frequent among the example pairs, as any frequent graph query connecting pairs of interest has to be a combination of such paths. It is similar in spirit to relational pathfinding, but using frequency, that is, the number of positive examples covered, as a selection criterion. The key idea behind this phase of the algorithm is to transform the problem into a sequence of constrained frequent itemset mining tasks, which can be solved efficiently using an off-the-shelf tool. More specifically, these subtasks are defined as follows:

Given a set of transactions $\mathcal{T}$, each representing all features of a path of length k connecting a positive example pair, and a frequency threshold γ,
find all itemsets that (a) cover transactions for at least γ different example pairs, and (b) correspond to a path query, that is, contain an edge feature for every pair of neighboring nodes on the path.

Algorithm FrequentPQ, detailed in Figure 3, extracts linked paths of increasing length that connect example pairs in the network, for each length creating and solving the corresponding frequent itemset mining task. We use the set of all starting nodes in the examples (line 2) as seed paths for the main loop that processes paths of increasing length. The algorithm terminates if no example pair has been covered for the first time in the last κ iterations. In the kth iteration, the nested loop in lines 5–12 extends paths in $\mathcal{P}_{k-1}$ to paths of length k, discards cyclic paths, and stores the resulting paths in $\mathcal{P}_k$. For each path connecting an example pair, ToTransaction produces the corresponding transaction, and adds it along with the connected example pair to $\mathcal{T}$ (line 12). Next, frequent path queries for a given frequency threshold γ are mined from $\mathcal{T}$ (line 13) and added to the set C of

queries to be returned. We keep track of the set E_k of covered examples for the termination criterion.

For a given path, ToTransaction creates a transaction based on the following encoding. An item is a tuple (i, f), where i is either a single node or a pair of nodes, and f a feature. We refer to pairs of nodes adjacent on a path as *backbone edges*, and to all other pairs of nodes appearing in the path as *crossing edges*. Given a path of length k, its first and last nodes receive identifiers $\#A$ and $\#Z$, respectively, and intermediate nodes along the path $\#1$ to $\#(k-2)$. The corresponding transaction created by ToTransaction contains an item (n, a) for each node feature a of a node n on the path, an item $((n, m), e)$ for each edge feature e of a backbone edge (n, m), and an item $((n, m), c)$ for each comparison feature c of a backbone or crossing edge (n, m). For example, the path of length three (#A, #1, #Z) from Figure 1(b) is represented by the following itemset:

$$\{((\#A, \#1), \texttt{parent}), (\#1, \texttt{male}), ((\#Z, \#1), \texttt{parent}), ((\#A, \#Z), \texttt{age}^{<}), (\#Z, \texttt{male})\}.$$

Note that this encoding is reversible, that is, given any transaction, we can extract the corresponding path query, replacing each identifier by a unique variable.

The transactions for all paths of the current length form the input for the corresponding frequent itemset mining task (line 13). Any algorithm that solves this mining task could be used here. Our solution exploits a declarative approach to mining patterns under constraints (Guns et al, 2011), as the corresponding system allows for user-defined constraints on frequent itemset mining tasks. This is important for our approach, as the definition of support employed when mining for frequent path queries in $\mathcal{T}$ differs from the usual: the support of an itemset is not the number of transactions that contain it, but instead the number of distinct corresponding example pairs. In other words, we count the number of answer substitutions rather than the number of instantiations and are interested in finding path queries that connect many different example pairs rather than path queries having many instances for a given pair. Furthermore, in order to maintain connectivity, we require that some item corresponding to an edge constraint must be present for each backbone edge. More specifically, the approach of Guns et al (2011) combines a declarative task specification language with a generic constraint programming solver. The user provides the transactions to mine itemsets from and specifies the desired constraints on itemsets in this language. The system interprets the task as a constraint program, to which it finds all solutions by calling the constraint solver. Algorithm FIMPaths thus simply combines the transactions $\mathcal{T}$ with a specification of our support and connectivity constraints, passes them to the system of Guns et al (2011) to obtain the result, and transforms each itemset in the result into the corresponding path query as discussed above.

4.2 Combining Path Queries into Graph Queries

The second phase of FpQm obtains more expressive graph queries by combining frequent path queries. Again, we use a reduction to frequent itemset mining to obtain frequent queries. For each positive example pair, we combine the frequent paths covering it into a graph and represent it as a transaction (Figure 2, lines 3–5). Frequent graph queries are then mined from these transactions and further filtered based on user-defined acceptance criteria (lines 6–9).

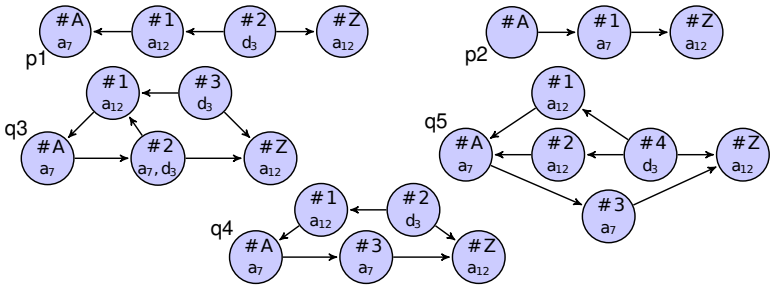


Fig. 4 Example of three graph queries (q3-q5) combining path queries p1 and p2

As an illustration of graph queries, Figure 4 depicts two path queries p1 and p2 as well as three example graph queries that are obtained by merging query variables, and potentially other nodes as well, of one or more copies of these paths. Clearly, allowing multiple copies of a path permits an infinite number of combinations. However, merging intermediate nodes that assign conflicting values to attributes results in invalid queries, and only finitely many among the valid queries are supported by the data. Therefore, we merge paths based on their instantiations in the data rather than based on their query representation. This ensures that we only construct valid queries with non-empty support.

More specifically, given a set C of path queries with query variables $(\#A, \#Z)$ and a positive example (o_1, o_2), we call *bottom clause* the union of all possible instantiations of bodies of queries in C that map $(\#A, \#Z)$ to (o_1, o_2). This is similar to the bottom clause obtained in some ILP approaches, but consists only of features from frequent path queries.

In order to represent such a bottom clause as a transaction, we need to assign identifiers to its intermediate nodes. This is done by AlignedBottomClause (line 4), which iteratively labels nodes based on the identifiers of nodes they instantiate in the paths queries as well as their neighboring nodes. This identifies groups of nodes occurring in the same context. Figure 5 illustrates the alignment of the bottom clause for positive example $(13, 82)$ and two path queries p1 and p3. The graph with rectangular nodes and numerical identifiers represents the relevant part of the data network. In this example, node 44 and node 81 both are instantiated by identifier #1 in p3 only, with identical neighbors, and therefore receive the same new identifier #2. While such *duplicate variables* are interesting from an expressivity point of view (as under θ_{OI}-subsumption they implement counting), they also result in multiple query instantiations for the same pair of answer nodes, which can be undesirable from an efficiency point of view. In this paper, we do not exploit the extra expressivity. Such duplicate variables receive the same label and will not be distinguished later on. Note that multiple copies do not necessarily generate duplicate nodes. For instance, in Figure 4, nodes #3 and #2 of query q3 both correspond to node #2 of p1 in two copies, yet are no duplicates because one is combined with node #1 of p2 and the other is not.

Once the bottom clauses have been aligned via the new identifers, all their features are collected into transactions using ToTransaction (line 5) as in Section 4.1. Given these transactions $\mathcal{T}$ and the frequency threshold γ, the next step consists in finding all itemsets that are frequent among $\mathcal{T}$. As each example has

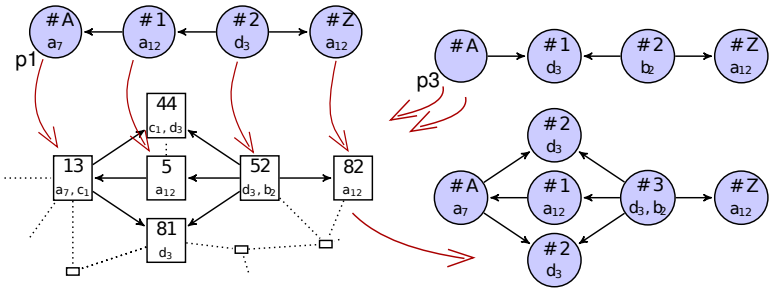


Fig. 5 Example of construction of an aligned bottom clause; see Section 4.2 for details

at most one associated transaction, the support of an itemset here is simply the number of distinct transactions containing it, that is, a frequent itemset covers at least γ transactions. Similarly to FIMPaths, FIMGraphs (line 6) combines the transactions $\mathcal{T}$ with a specification of this frequency constraint and passes them to the system of Guns et al (2011) to obtain the result. We do not include connectivity constraints, as those cannot be enforced at the time of mining here. As its last step, FIMGraphs therefore filters out unconnected queries. For each frequent itemset returned by FIMGraphs, computeQuery recovers the corresponding query as discussed for path queries above, maps it onto the data to determine all instances and computes the support (line 7). Note that only graphs that are frequent among positive examples are generated as a result of this procedure. Queries satisfying the user provided criteria Γ (discussed next) are collected in D.

4.3 Selecting a Subset of Queries

The final phase of FpQm (lines 10–13) greedily selects a subset of best queries with limited overlap among the ones stored in D. The queries are processed in turn, from best to worst according to the ranking criterion in Γ. If the part of the support of the current query that has not yet been covered, called its *contribution*, is sufficiently large (larger than δ), it is accepted and supported positive examples are marked as covered. Otherwise it is rejected. All accepted queries are returned as the output of FpQm.

More specifically, Γ offers a number of parameters that can (but need not) be used to orient the search toward queries with preferred characteristics and to rank queries. First, thresholds can be imposed on the coverage of queries, including the following:

$$\text{cover diff.} = |E_{1,1}| - |E_{0,1}| \quad \text{support} = |E_{1,1}| + |E_{0,1}| \quad \text{precision} = \frac{|E_{1,1}|}{|E_{1,1}|+|E_{0,1}|}$$
$$\text{cover ratio} = \frac{|E_{1,1}|}{|E_{0,1}|} \quad \text{accuracy} = \frac{|E_{1,1}|}{|E_{1,0}|+|E_{1,1}|+|E_{0,1}|} \quad \text{recall} = \frac{|E_{1,1}|}{|E_{1,0}|+|E_{1,1}|}.$$

Second, additional constraints can be imposed on the form of queries. For instance, one can restrict the number of nodes or edges to limit the complexity of the patterns and facilitate interpretation, or require intermediate nodes to have a minimum number of instantiations in order to exclude queries that only apply in the neighborhood of a specific node.

The frequency threshold γ used in the two frequent itemset mining steps can be automatically determined from the minimum support as well as from the accuracy or the recall, given the number of positive examples.

4.4 An Efficient Relational Query Miner

To summarize, FpQm mines frequent path queries, combines them into graph queries, and finally selects a good subset of queries.

As discussed above, we filter out duplicate variables when constructing graph clauses to avoid a combinatorial explosion of the number of instances. To reduce the amount of filtering necessary, the frequent path mining step drops paths with high multiplicity, that is, with a number of instances a factor greater than the number of supporting pairs, as these will likely create duplicate variables. Also, when mapping the queries onto the data, we fix a limit on the number of instances that can be generated. For each query edge, the number of instances that will be obtained after mapping it is estimated from the current number of instances and the number of matching data edges. Query edges with lowest estimates are processed first and if the current estimate crosses the chosen threshold, remaining edges are dropped from the query.

To further improve efficiency, the computationally costly operation of finding graph instances is shared between queries. Indeed, if an itemset I is a subset of another itemset J, the query represented by J will be a refinement of that corresponding to I. Hence, the instances of I's query are a superset of those of J's query and can be used to initialize its mapping. Only the extra predicates will have to be added to the mapping.

5 An Alternating Scheme for Relational Redescription Mining

We now turn to the alternating scheme for finding relational redescriptions, Arrm, sketched in Figure 6. Given an initial set of queries, the algorithm first grows a forest of queries by repeatedly running FpQm to find the best descriptions of each query, which it adds as the query's children. The second phase then extracts redescriptions from this forest.

5.1 Initialization

The algorithm expects an input set I of queries, based on which its two key data structures, the candidate list $\mathcal{K}$ and the set of explored descriptions $\mathcal{M}$ are initialized (lines 2–4). The simplest means to generate this input set is to consider the queries obtained from each edge predicate taken individually, that is, for each value v of each edge predicate a

$$c(\#A, \#Z) \text{ :- } \epsilon_a^v(\#A, \#Z).$$

In addition, one might consider simple combinations such as

$$c(\#A, \#Z) \text{ :- } \epsilon_a^v(\#1, \#A), \epsilon_a^v(\#1, \#Z), \text{ or}$$
$$c(\#A, \#Z) \text{ :- } \epsilon_a^v(\#A, \#1), \epsilon_a^v(\#Z, \#1).$$

In particular, this might be useful in cases where the nodes that appear in the first (respectively second) position of edges supporting ϵ_a^v do not appear in any other predicate. The user may also specify a set of initial queries manually.

5.2 Growing a Forest of Queries

Given the current candidate list $\mathcal{K}$, each query K in $\mathcal{K}$ is processed in turn, using the FpQm algorithm presented above to find new queries describing the same set of examples, but using different attributes (lines 5–12).

That is, the supporting pairs of the current candidate K constitute the positive examples, and the network for that round is obtained from the original network by removing all predicates based on attributes used in K. Given this input, FpQm is used to find the best describing children queries (line 7), which are added to the set of explored descriptions $\mathcal{M}$, linked to the candidate in the query forest, and, if they meet criteria discussed below, appended to the list of candidates $\mathcal{K}$. This process is repeated until $\mathcal{K}$ is empty.

The algorithm thus grows a forest of queries breadth first, branching from the initial candidates. A branch might be interrupted for one of three reasons. First, if no child is returned by the query mining procedure the expansion stops naturally. Second, if a child has a support (modulo symmetry) and attributes set identical to some query found previously (non empty $\mathcal{E}$, line 11), it is not added to the candidates for expansion. Indeed, since the algorithm is entirely deterministic this would not generate new queries but practically introduce a loop in the exploration. Third, a maximum exploration depth τ can be fixed as part of Γ and branches that reach this length will not be expanded further (line 11).

The shape of the forest is affected primarily by the quality criteria Γ (cf. Section 4.3). Strict criteria limit the *fertility* of queries, i.e. the number of results returned by FpQm, and thus the branching factor of the forest. With high fertility, only shallow exploration will be manageable. On the other hand, stricter selection of the children, provided that some degree of diversity is maintained, allows one to explore more generations. To direct the exploration towards more accurate queries, the accuracy of the current query can be applied as a filtering criterion for its children queries.

5.3 Retrieving Redescriptions

Once the candidate list $\mathcal{K}$ has been exhausted, the next step consists in retrieving good redescriptions from the forest of queries. We gather into $\mathcal{U}$ all pairs of adjacent queries along the forest. By construction, such pairs have overlapping supports and disjoint attribute sets. Also, we scan the forest for pairs of queries that are not adjacent but have identical support and disjoint attribute sets. These form extra candidate redescriptions that are added to $\mathcal{U}$ (line 13).

Finally, the same greedy procedure as in FpQm, but considering redescriptions instead of individual queries, is applied to $\mathcal{U}$ to select the final set of redescriptions (lines 14–17).

Input: A network, a set of initial queries I and quality criteria Γ.
Output: A set of relational redescriptions $\mathcal{R}$.
$\mathcal{S} \leftarrow \emptyset$; $\mathcal{M} \leftarrow \emptyset$; $\mathcal{K} \leftarrow \emptyset$;
for $i \in I$ **do**
  $K.query \leftarrow i$; $K.parent \leftarrow \emptyset$; $K.generation \leftarrow 0$
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{K\}$; $\mathcal{M} \leftarrow \mathcal{M} \cup \{K\}$
while $\exists K \in \mathcal{K}$ **do**
  $\mathcal{K} \leftarrow \mathcal{K} \setminus \{K\}$
  for $q \in$ FpQm(supp($K.query$), Γ) **do**
    $L.query \leftarrow q$; $L.parent \leftarrow K$; $L.generation \leftarrow K.generation + 1$
    $\mathcal{E} \leftarrow \{S \in \mathcal{M}, \text{supp}(S) = \text{supp}(L) \wedge \text{att}(S) = \text{att}(L)\}$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{L\}$
    if $L.generation \leq \tau \wedge \mathcal{E} = \emptyset$ **then**
      $\mathcal{K} \leftarrow \mathcal{K} \cup \{L\}$
$\mathcal{U} \leftarrow \{(M, N) \in \mathcal{M}^2, (M.parent = N) \vee (\text{supp}(M) = \text{supp}(N) \wedge \text{att}(M) \cap \text{att}(N) = \emptyset)\}$
for $R \in \mathcal{U}$ ordered according to Γ **do**
  if $|E_{1,1}(R) \setminus \mathcal{S}| \geq \delta$ **then**
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup E_{1,1}(R)$
return $\mathcal{R}$

Fig. 6 ArRM: Alternating Relational Redescription Mining

6 Experiments

We now turn to the experimental evaluation of our approach. In the first experiment, we focus on FpQm alone. We explore the influence of its parameters and compare the approach to an existing relational query miner, which we use as a baseline for both pattern quality and running time. The second set of experiments concerns the full alternating scheme. It again explores algorithm parameters, compares to a propositional approach, and also provides a qualitative assessment of redescriptions discovered.

Our algorithm was implemented in Python, using Fim Cp (Guns et al, 2011) to mine path and graph queries (FIMPaths and FIMGraphs). In all experiments, it was run on a single core of an 8 core Intel Xeon 2.8GHz processor and with 32GB of memory. In all runs, we limited the number of trials for mining paths to $\kappa = 2$, the number of instances when mapping queries to 20000, the mining time per Fim Cp call to 1min and the absolute minimum support of any query to 3.

The implementation of our algorithms and the prepared datasets are available online.[1]

6.1 Datasets

Three relational datasets are used throughout the experiments. The first dataset, `Kinship`, was extracted from the *Alyawarra Ethnographic Database*.[2] The other two datasets, `UWCSE` and `UMLS`, were obtained from the Alchemy repository.[3] Table 1 summarizes the characteristics of the datasets.

[1] `http://www.cs.helsinki.fi/u/galbrun/redescriptors/`
[2] `http://habc.eu/csac/wiki/knsrc/KinSources/AU01Alyawarra1971`
[3] `http://alchemy.cs.washington.edu/`

Table 1 Datasets statistics: number of nodes, edges, node predicates, edge predicates and comparison predicates

Dataset	#nodes	#edges	#node. pred.	#edge. pred.	#comp. pred.
`Kinship`	381	24053	3	31	1
`UMLS`	135	4181	–	46	–
`UWCSE`	1042	1674	6	7	5

`Kinship` provides personal and genealogical information about individual members of an indigenous community of Australia, the Alyawarra, as well as the kinship terms they use for their relationships to other persons. A glossary of kinship terms is available, to which we can compare our findings. To simplify the notation, we use the indices of kinship terms from the glossary, rather than the terms themselves. For instance, `awaadya` is later denoted as `kin10`. Kinship information is not available for all living individuals; we mark as `relevant` those for whom it is complete.

`UMLS` characterizes the relations between biomedical concepts in terms of the Unified Medical Language System ontology.

`UWCSE` contains information about relationships between persons and courses within the computer science department of the university of Washington. It includes two predicates of arity three, namely `taught by(Course,Person,Time)` and `ta(Course,Person,Time)`, which our algorithm cannot handle natively. We therefore split those into binary predicates linked together by a newly introduced course identifier.

6.2 Mining Relational Queries

The first series of experiments focuses on our proposed path-based relational query mining algorithm, FpQm. After reporting general observations about the behaviour of the algorithm, it is compared to a baseline relational query miner.

Algorithm behaviour. As expected, the minimum support γ strongly affects the number of path queries found in the first step of FpQm (for instance, from 2 when $\gamma = 0.30$ up to 368 when $\gamma = 0.05$, on average for `Kinship`). Raising γ can actually result in increased running times because in the absence of frequent short queries, paths will be extended to greater lengths, possibly at high computational expenses and without success. On the other hand, the algorithm will be overwhelmed by the quantity of patterns when γ is set too low. Clearly, it is advisable to set γ as low as possible according to the size and density of the dataset.

Raising the extension threshold κ can lead to the generation of numerous variations of shorter connecting paths. This potential for enriching the queries typically comes at a high computational cost and κ should thus be kept low.

We also studied the effect of the greedy selection of queries on the cover of positive examples. The aim of this phase is mainly to reduce the high redundancy of the results. Indeed, we observed that before greedy selection, we easily obtained hundreds of queries, but a very limited number of queries (1–3) is generally sufficient to obtain almost the same cover. Hence, this pruning phase is crucial for limiting the fertility of clauses while maintaining the coverage quality.

Table 2 Comparison of C-ARMR and FPQM on `Kinship`: number of positive examples $|E^+|$, number of true and false positives $|E_{1,1}|$ and $|E_{0,1}|$, aggregated accuracy J, number of queries $|Q|$, and running time (T) in seconds

		C-ARMR					FPQM				
Predicate	$\|E^+\|$	$\|E_{1,1}\|$	$\|E_{0,1}\|$	J	$\|Q\|$	T(s)	$\|E_{1,1}\|$	$\|E_{0,1}\|$	J	$\|Q\|$	T(s)
`kin1`	228	16	50	0.06	12	531	12	7	0.05	1	5
`kin2`	489	38	29	0.07	5	532	154	74	0.27	4	40
`kin3`	231	36	102	0.11	14	423	78	40	0.29	5	41
`kin4`	379	24	31	0.06	179	663	0	0	0.00	0	7
`kin5`	493	11	9	0.02	12	564	0	0	0.00	0	2
`kin6`	508	87	2	0.17	20	433	148	33	0.27	4	19
`kin7`	453	50	7	0.11	6	462	209	93	0.38	8	20
`kin8`	817	56	1	0.07	21	502	92	1	0.11	2	2
`kin9`	805	64	203	0.06	6	513	166	33	0.20	3	19
`kin10`	462	41	3	0.09	7	413	49	41	0.10	2	1
`kin11`	505	38	4	0.07	6	442	42	34	0.08	2	0
`kin12`	739	75	11	0.10	23	598	81	61	0.10	2	1
`kin13`	299	0	0	0.00	0	396	159	123	0.38	14	46
`kin14`	447	0	0	0.00	0	449	87	17	0.19	2	14
`kin15`	43	0	0	0.00	0	445	20	6	0.41	3	10
`kin16`	943	130	148	0.12	15	551	203	53	0.20	3	85
`kin17`	1256	157	127	0.11	11	582	0	0	0.00	0	93
`kin18`	392	61	3	0.15	9	466	61	3	0.15	1	3
`kin19`	569	36	24	0.06	16	507	0	0	0.00	0	88
`kin20`	13	0	0	0.00	0	338	7	0	0.54	2	5
`kin21`	272	43	13	0.15	6	437	109	38	0.35	4	48
`kin22`	142	20	132	0.07	9	453	49	32	0.28	6	8
`kin23`	193	0	0	0.00	0	343	53	29	0.24	5	15
`kin26`	6	0	0	0.00	0	219	0	0	0.00	0	3

Relational Query Miner Comparison Next, we compare our proposed path-based algorithm FPQM to a baseline relational query miner on the three datasets. For each edge predicate in turn, we take the supporting pairs of nodes as positive examples and mine queries over the remaining attributes. We do not consider cases with less than four positive examples. Furthermore, with `Kinship` we only consider pairs of nodes that are both relevant.

As a baseline, we use a modified version of C-ARMR (De Raedt and Ramon, 2004) (implemented in Prolog) that mines top-k queries with respect to the difference in support on positive and negative examples. Given a set of positive examples and considering all remaining pairs of nodes as negative examples, we allowed C-ARMR to mine for top-5 queries with positive score. As discussed in Section 3, the implementation does not ensure that query variables are linked. To address this problem, we refine unlinked queries if they cover at least one positive example, but never include them in the result. This is similar in spirit to generating candidates based on the data as common in relational pathfinding and function learning (Richards and Mooney, 1992; Ong et al, 2005; Santos et al, 2009), but avoids the need to adapt the canonical refinement operator used in our implementation. Experiments with C-ARMR have been performed on a single core of a C2Q machine (2.4GHz 4GB for `Kinship`, 2.83GHz 8GB for `UMLS` and `UWCSE`).

In order to obtain results comparable to C-ARMR's, we also use the cover difference to select queries with FPQM (cf. Section 4.3). Similarly, only positively scoring queries are output. In addition, we require the accuracy of the queries to exceed $j = 0.05$, so as to obtain a minimum frequency threshold for mining paths and graphs.

Table 3 Comparison of C-ARMR and FPQM on `UMLS`. Legend as in Table 2

		C-ARMR					FPQM				
Predicate	$\lvert E^+\rvert$	$\lvert E_{1,1}\rvert$	$\lvert E_{0,1}\rvert$	J	$\lvert Q\rvert$	T(s)	$\lvert E_{1,1}\rvert$	$\lvert E_{0,1}\rvert$	J	$\lvert Q\rvert$	T(s)
adjacent to	7	5	4	0.45	11	38	3	1	0.38	1	2
affects	1022	437	0	0.43	7	129	635	124	0.55	5	65
analyzes	52	50	15	0.75	7	38	52	14	0.79	2	1
ass. eff. of	65	50	2	0.75	29	55	65	15	0.81	1	9
associated with	239	110	46	0.39	5	102	187	122	0.52	11	107
carries out	38	36	36	0.49	5	46	36	0	0.95	1	2
causes	360	280	116	0.59	5	72	354	82	0.80	5	164
complicates	263	189	95	0.53	5	71	263	29	0.90	4	171
concep. part of	18	0	0	0.00	0	283	3	1	0.16	1	2
connected to	4	4	8	0.33	82	45	0	0	0.00	0	2
consists of	9	9	8	0.53	18	38	8	0	0.89	1	6
contains	11	6	2	0.46	5	50	5	0	0.45	1	1
co-occurs with	67	42	24	0.46	16	66	61	34	0.60	5	190
degree of	34	30	6	0.75	43	52	30	0	0.88	1	90
dev. form of	4	4	2	0.67	15	58	3	0	0.75	1	5
diagnoses	48	20	4	0.38	9	66	48	0	1.00	4	61
disrupts	154	112	68	0.50	5	74	125	77	0.54	4	24
evaluation of	63	28	8	0.39	7	52	46	6	0.67	2	33
exhibits	45	18	0	0.40	6	34	45	24	0.65	1	1
indicates	27	18	4	0.58	5	189	18	0	0.67	1	10
ingredient of	28	0	0	0.00	0	203	0	0	0.00	0	2
interacts with	451	155	137	0.26	5	73	451	451	0.50	6	9
isa	500	7	1	0.01	10	188	0	0	0.00	0	22
issue in	268	68	0	0.25	6	94	256	0	0.96	4	3
location of	319	159	117	0.36	5	94	269	97	0.65	9	21
manages	6	0	0	0.00	0	152	4	0	0.67	1	3
manif. of	194	138	82	0.50	5	69	193	92	0.67	9	154
measurement of	64	31	1	0.48	5	80	62	9	0.85	5	95
measures	180	65	0	0.36	18	56	164	86	0.62	3	203
method of	25	18	16	0.44	7	39	22	15	0.55	2	0
occurs in	90	60	18	0.56	5	79	87	8	0.89	4	36
part of	200	176	122	0.55	5	63	102	0	0.51	3	4
performs	90	18	0	0.20	7	47	90	0	1.00	5	83
precedes	73	72	13	0.84	62	60	72	0	0.99	3	268
prevents	32	30	0	0.94	13	59	30	0	0.94	1	9
process of	437	437	165	0.73	5	64	0	0	0.00	0	97
produces	276	140	65	0.41	5	86	254	47	0.79	6	61
property of	44	34	0	0.77	1	156	34	0	0.77	1	18
result of	586	306	63	0.47	5	95	361	137	0.50	10	63
surrounds	8	8	9	0.47	84	51	4	0	0.50	1	0
treats	56	50	0	0.89	21	68	50	10	0.76	3	25
uses	65	48	0	0.74	6	81	50	6	0.70	2	2

Table 4 Comparison of C-ARMR and FPQM on `UWCSE`. Legend as in Table 2

		C-ARMR					FPQM				
Predicate	$\lvert E^+\rvert$	$\lvert E_{1,1}\rvert$	$\lvert E_{0,1}\rvert$	J	$\lvert Q\rvert$	T(s)	$\lvert E_{1,1}\rvert$	$\lvert E_{0,1}\rvert$	J	$\lvert Q\rvert$	T(s)
advised by	113	11	14	0.09	8	646	9	4	0.08	1	0
ta	195	0	0	0	0	257	0	0	0	0	0
taught by	286	21	30	0.07	8	446	0	0	0	0	4
tempAdvised by	37	0	0	0	0	392	0	0	0	0	0

Tables 2–4 present quantitative results and running times for the three datasets respectively. For C-ARMR, which only scores individual patterns, we use the disjunction of top-5 patterns with positive scores. As C-ARMR returns all equally scoring patterns in case of ties, these disjunctions can have more than five elements. For each case, we report the total number of positive examples $\lvert E^+\rvert$, as well as the number of queries returned $\lvert Q\rvert$, the number of true and false positives

$|E_{1,1}|$ and $|E_{0,1}|$, the aggregated accuracy J (i.e. the Jaccard coefficient between the set of example pairs and the union of the supports of output queries) and the running times T for both c-armr and FpQm.

On `Kinship`, cf. Table 2, we restrict the number of body literals in c-armr's queries to at most five, as running times become prohibitive for longer queries due to large numbers of unlinked or non-discriminative queries. Under this restriction, we observe comparable or better accuracies for patterns found by FpQm, which moreover is at least one order of magnitude faster. On average, FpQm returns fewer queries than c-armr because it includes a selection procedure to remove redundant queries, which is not the case with c-armr.

Furthermore, as a direct consequence of this restriction, no pattern with positive score was found for six of the kinship terms, whereas FpQm is able to identify more complex patterns for these cases. For four predicates, c-armr found queries where FpQm did not return any. In three of these cases, the positive support of individual queries is lower than 3 or their accuracy lower than 0.05. These values are below the thresholds used in FpQm hence they did not qualify as good quality queries. For the remaining predicate, `kin17`, FpQm was overwhelmed by the quantity of frequent queries, an issue easily solved by raising the accuracy threshold.

On inspection, the obtained queries correspond to definitions provided in the glossary. Some deviations are observed, such as an intermediate genealogical level or a difference in gender of some individual.

On `UMLS`, the difference in running time gets even more pronounced. Here, c-armr could not identify the top-5 patterns up to four body literals within two hours for any term, and failed to do so for up to three literals for twelve predicates, taking between 9 and 95 minutes for the remaining ones. Table 3 therefore reports results with up to two body literals. FpQm, on the other hand, takes seconds or at most up to a few minutes per predicate, and often finds more accurate queries, as it does not suffer from the restricted expressiveness of short queries.

On `UWCSE`, no queries could be found for most predicates. Contrarily to c-armr, FpQm did not return any query for `taught by` due to individual queries having insufficient accuracy. We observe that in the absence of good connecting queries, c-armr still takes several minutes to complete the search, while FpQm returns within a few seconds.

These experiments thus indicate that the FpQm algorithm can find more complex queries much faster, compared to a standard query mining approach.

6.3 Finding Relational Redescriptions

We now turn to the full relational redescription mining algorithm, Arrm. First, we study its behaviour, particularly the impact of queries selection. Next, we compare it to three alternative approaches, namely using c-armr instead of FpQm within the alternating scheme, using propositionalization followed by a propositional redescription miner, and restricting Arrm to path queries rather than graph queries. Finally, we provide concrete illustrative examples of redescriptions found by Arrm.

Table 5 Quantitative results and running times (T) for mining redescriptions under different parameterizations (cf. Section 6.3 for details). $|\mathcal{R}|$ and $|\mathcal{M}|$ are the number of redescriptions returned and of queries explored, respectively. Fert. is the average clause fertility.

Dataset	$\|\mathcal{R}\|$	$\|E_{1,1}\|$			J			$\|\mathcal{M}\|$	Fert.	Tot. T	T/clause	
Parameters		min	max	avg	min	max	avg				max	avg
`Kinship`												
Gj τ_1	39	4	784	141	0.05	0.18	0.10	31	1.90	32min	4min	62s
Gj τ_5	62	5	784	91	0.05	0.50	0.20	228	1.13	3h 14min	10min	51s
Gj τ_8	64	5	784	88	0.05	0.53	0.22	263	1.05	3h 58min	10min	54s
Gr τ_1	59	4	283	84	0.05	0.11	0.07	31	2.77	33min	4min	64s
Gr τ_5	177	4	753	31	0.05	0.61	0.25	744	1.36	8h 14min	10min	39s
Gr τ_{10}	205	4	753	27	0.06	0.66	0.27	1190	1.08	14h 37min	10min	44s
Gr τ_{12}	203	4	753	27	0.06	0.66	0.27	1203	1.08	14h 45min	10min	44s
Gs τ_1	37	4	784	153	0.05	0.17	0.08	31	1.77	32min	4min	62s
Gs τ_5	57	4	784	110	0.05	0.28	0.11	202	1.46	2h 41min	8min	48s
Gs τ_{10}	65	8	784	101	0.05	0.28	0.12	320	1.32	4h 47min	9min	53s
Gs τ_{15}	69	8	784	98	0.05	0.28	0.13	350	1.29	5h 20min	10min	55s
Pj τ_1	34	4	784	155	0.05	0.18	0.10	31	1.55	23min	2min	44s
Pj τ_5	57	4	784	102	0.05	0.43	0.15	178	1.17	1h 11min	2min	24s
Pj τ_7	56	4	784	102	0.05	0.43	0.15	207	1.06	1h 28min	2min	25s
Pr τ_1	43	4	466	108	0.05	0.16	0.08	31	2.06	23min	2min	46s
Pr τ_5	117	4	732	43	0.05	0.43	0.16	374	1.36	2h 10min	2min	20s
Pr τ_{10}	139	4	732	36	0.05	0.66	0.19	601	1.13	3h 47min	2min	22s
Ps τ_1	34	4	784	163	0.05	0.17	0.09	31	1.52	23min	2min	44s
Ps τ_5	49	4	784	123	0.05	0.28	0.12	169	1.33	1h 6min	2min	23s
Ps τ_{11}	57	4	784	111	0.05	0.35	0.13	222	1.23	1h 21min	2min	21s
`UMLS`												
Gj τ_1	19	5	404	100	0.35	0.94	0.65	42	0.71	9min 31s	76s	13s
Gj τ_3	20	6	552	118	0.35	1.00	0.79	87	0.68	13min 59s	76s	9s
Gr τ_1	19	5	374	99	0.35	0.94	0.64	42	0.83	9min 30s	77s	13s
Gr τ_4	20	6	374	106	0.35	1.00	0.80	111	0.80	14min 51s	77s	8s
Gs τ_1	18	5	437	107	0.33	0.75	0.51	42	0.67	9min 24s	76s	13s
Gs τ_4	19	6	552	130	0.35	1.00	0.64	86	0.65	16min 51s	76s	11s
Pj τ_1	16	5	404	103	0.36	0.94	0.63	42	0.62	8min 2s	77s	11s
Pj τ_4	17	6	552	124	0.38	1.00	0.78	80	0.62	11min 21s	77s	8s
Pr τ_1	16	5	404	104	0.36	0.94	0.62	42	0.76	8min 1s	77s	11s
Pr τ_4	18	6	404	107	0.38	1.00	0.79	101	0.78	11min 26s	77s	6s
Ps τ_1	15	5	437	110	0.33	0.75	0.48	42	0.57	8min 4s	77s	11s
Ps τ_4	16	6	552	138	0.35	1.00	0.63	78	0.62	13min 49s	77s	10s
`UWCSE`												
Gj τ_1	15	5	278	53	0.02	0.24	0.07	8	2.50	5s	2s	<1s
Gj τ_3	19	5	278	49	0.02	0.28	0.10	35	0.94	24s	10s	<1s
Gr τ_1	16	5	278	53	0.02	0.24	0.07	8	3.75	5s	2s	<1s
Gr τ_4	21	5	278	47	0.02	0.28	0.10	50	1.16	31s	9s	<1s
Gs τ_1	14	5	278	42	0.02	0.24	0.06	8	2.00	6s	2s	<1s
Gs τ_3	17	5	278	41	0.02	0.24	0.06	28	0.89	23s	10s	<1s
Pj τ_1	15	5	278	53	0.02	0.24	0.07	8	2.50	4s	2s	<1s
Pj τ_3	19	5	278	49	0.02	0.28	0.10	35	0.94	21s	9s	<1s
Pr τ_1	16	5	278	53	0.02	0.24	0.07	8	3.50	5s	2s	<1s
Pr τ_4	21	5	278	47	0.02	0.28	0.10	48	1.17	26s	8s	<1s
Ps τ_1	14	5	278	42	0.02	0.24	0.06	8	2.00	3s	1s	<1s
Ps τ_3	17	5	278	41	0.02	0.24	0.06	28	0.89	18s	9s	<1s

Algorithm behaviour. The criterion for ranking queries is the major lever in the selection procedure, directly impacting the exploration. We investigate this aspect by letting ARRM mine redescriptions from all three datasets with varying parameters. More specifically, we considered positive support (s), cover ratio (r) and accuracy (j) as options for descending ranking of the queries. We used various maximum exploration depths (τ_x), from a single round up to the number of alternations after which the algorithm stopped with no candidates left, or at most 15.

In addition, as an alternative to graph queries (G), we limited the algorithm to paths queries (P) by shortcutting the combining phase.

Looser selection methods that yield more outputs per candidate query (upwards of 5 on average) were also studied. They result in very broad query trees whose dimensions become unmanageable already after the first round. They showed no improvements in the quality of redescriptions after uncomplete runs lasting over a day and hence were abandoned.

The remaining criteria in Γ control the complexity of the queries, avoiding the generation of patterns that are unduly specific or contain overly many constraints. Such patterns have a very limited explanatory power and are considered worthless. These criteria were fixed as follows for each dataset in all runs. We required any node to be instantiated by at least 3 distinct data nodes. For `Kinship`, we limited the number of nodes and of edges in a query to $n = 7$ and $e = 10$, respectively. The minimum contribution of a clause was set to $d = 0.66$ of its support and the minimum accuracy for the first turn was set to $j = 0.05$. For subsequent turns the jaccard coefficient of the parent query was used as a minimum threshold. For `UMLS`, we set these thresholds to $n = 5$, $e = 10$, $d = 0.25$ and $j = 0.33$. For `UWCSE`, we set $j = 0.33$ and the absolute mininum contribution to 3 but did not limit the number of nodes and edges.

Table 5 presents statistics of the results obtained for each run as well as running times. We observe important variations in the support and accuracy across datasets, reflecting the variety of the redescriptions found. First, alternating for a few turns does allow one to find more accurate patterns than simply matching isolated predicates (τ_1). Note that contrarily to Tables 2–4, accuracies here are not aggregated over multiple patterns but obtained from individual redescriptions. Running times per query vary from less than a second up to several minutes. Of course, the total running time needed for the full exploration depends heavily on the number of queries explored, $|\mathcal{M}|$. As mentioned previously, an important factor impacting the running times is the presence of symmetries which results in duplicate variables, leading to numerous instances and hence more costly mapping.

The effects of different parameterizations are limited with `UMLS` and `UWCSE`, where the algorithm consistently mined a small number of relatively simple queries. Parameterization is more critical with `Kinship`, where more complex queries are needed to capture the various meanings of kinship terms. In particular, with the former two datasets, the algorithm stops after a few alternations upon finding only redundant queries. Therefore, further raising τ does not affect the outcome. In all datasets, ranking queries by cover ratio allowed to find the best redescriptions while ranking by support appears suboptimal.

Alternative query miner. As the alternating scheme is independent of the query mining algorithm used (line 7 in Figure 6), we created a modified version of ArRM, where we replace FpQm by c-armr. We use the same parameterizations of c-armr as in the experiments above, with the exception of the number of body literals for `Kinship`, which we had to restrict to at most three (rather than five) here to keep running times feasible. This is due to the much larger search space over kinship terms compared to that over genealogical terms. We rank queries by cover ratio, as this was the best setting in the previous experiment.

Table 6 presents quantitative results and running times for mining redescriptions from the three datasets under this replacement. In all three cases, the al-

Table 6 Quantitative results and running times (T) for mining redescriptions with C-ARMR as the query miner (cf. Section 6.3 for details). Legend as in Table 5.

Dataset Parameters	$\|\mathcal{R}\|$	$\|E_{1,1}\|$ min	$\|E_{1,1}\|$ max	$\|E_{1,1}\|$ avg	J min	J max	J avg	$\|\mathcal{M}\|$	Fert.	Tot. T	T/clause max	T/clause avg
`Kinship` τ_3	8	4	75	34	0.07	0.15	0.11	53	2.04	19h 59min	2h	22min
`UMLS` τ_8	49	4	437	76	0.08	1.00	0.72	178	1.35	3h	3min	61s
`UWCSE` τ_2	3	40	61	52	0.07	0.15	0.10	32	2.67	4min 16s	51s	8s

Table 7 Quantitative results and running times for mining redescriptions by applying the REREMI algorithm on the propositionalized dataset (cf. Section 6.3 for details). $|D|$, $|F|$, T prop. and T mining are the number of rows and columns in the propositionalized dataset and the running time for propositionalization and for mining, respectively.

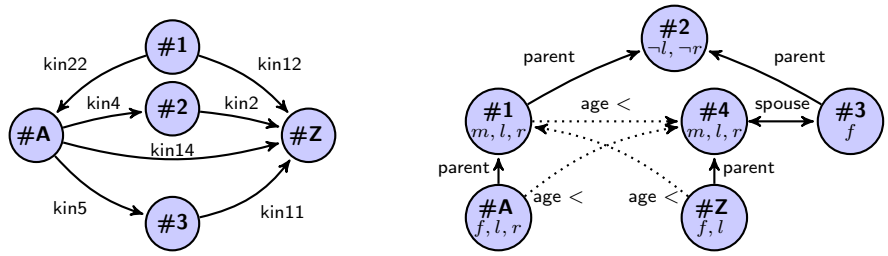
Dataset Parameters	$\|\mathcal{R}\|$	$\|E_{1,1}\|$ min	$\|E_{1,1}\|$ max	$\|E_{1,1}\|$ avg	J min	J max	J avg	$\|D\|$	$\|F\|$	T prop.	T mining
`Kinship`	66	3	1489	178	0.00	0.17	0.07	49074	13169	3 min 42s	34 min 19s
`UMLS`	100	6	182	36	0.04	1.00	0.74	18088	4998	23s	13 min 13s
`UWCSE`	58	3	41	19	0.01	0.24	0.09	55882	2048	15s	1min 45s

gorithm stopped after a few alternations (3 for `Kinship`, 8 for `UMLS` and 2 for `UWCSE`) without candidates for further expansion. In the case of `UMLS` and `UWCSE` the obtained redescriptions are almost on par with those found using FPQM, albeit somewhat less accurate, while the redescriptions found for `Kinship` have clearly lower accuracy. This can be explained as an effect of the length restriction required to keep C-ARMR running times feasible. As `UMLS` and `UWCSE` contain simple redescriptions, the length limit of two only moderately affects the quality of the outcome. With `Kinship`, however, restricting query length to at most three prevents the algorithm from finding the more complex redescriptions of higher accuracy the original ARRM identifies. Together with the significantly higher running times of the C-ARMR-based variant, these results show that we can find more accurate redescriptions more quickly by using our new FPQM method rather than C-ARMR within the alternating scheme.

A propositional approach. We next compare to an existing propositional redescription mining algorithm, following ideas from propositionalization (Kuzelka and Železný, 2009; Dinh et al, 2012) to suitably transform the data. Specifically, we extracted features from the dataset by enumerating paths up to a given length joining each pair of objects and ran the REREMI propositional redescription mining algorithm (Galbrun and Miettinen, 2012) on the resulting propositional dataset. We considered paths of length one and two for all datasets. In addition we also extracted paths over genealogical attributes of length up to five for the `Kinship` dataset.

As can be seen from Table 7, this method is able to find comparable redescriptions in competitive running times when applied to `UMLS` and `UWCSE`, where paths of length at most two suffice to capture the relations in the network. However, it fails with `Kinship`, where longer paths and more complex combinations are needed.

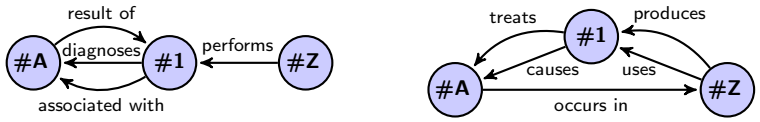
Especially in the latter case, our relational approach to redescription mining has clear advantages over the propositional approach. First, it can easily explore longer paths, whose inclusion results in feature matrices too large to handle with the propositional miner. Second, it bases the selection of paths considered on the



$|E_{1,1}| = 10$, J $= 0.58$

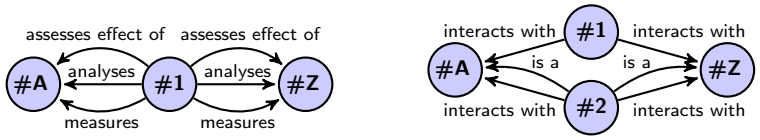
(1a) (1b)

Fig. 7 Example redescription found from `Kinship`. Node predicates are `male` (m), `female` (f), `living` (l) and `relevant` (r). Negated Boolean attributes are denoted with ¬.



$|E_{1,1}| = 60$, J $= 1$

(2a) (2b)



$|E_{1,1}| = 506$, J $= 0.62$

(3a) (3b)

Fig. 8 Example redescriptions found from `UMLS`

current set of positive examples, rather than on all possible pairs as necessary when performing propositionalization once before mining. Third, it explicitly combines paths into more expressive graph queries.

The second point could be addressed by repeated propositionalization during mining, which would require to modify the propositional approach. To simulate this setting, we performed experiments where we restricted our algorithm to path queries. Results are reported in Table 5. Once more, we observe that results are often comparable for the simpler cases of `UMLS` and `UWCSE`, but our fully relational method finds substantially more accurate redescriptions on `Kinship`, with an average accuracy of 0.27 when using graphs (Gr) compared to 0.19 when using paths only (Pr).

Examples of Redescriptions. We now provide a few example redescriptions found by ArRM on the three datasets. Figure 7 shows an example of redescription found from the `Kinship` dataset. Graph (1b) represents the genealogical link between a female individual and the daughter of her paternal aunt, which is one of the meanings of `kin14` found in the glossary. Kinship terms often have several meanings and may also be used in broader senses. In such cases, a configuration of several terms, as in graph (1a), better captures one of the senses than a term taken in isolation. For instance, this redescription has an accuracy of 0.59, a significant improvement

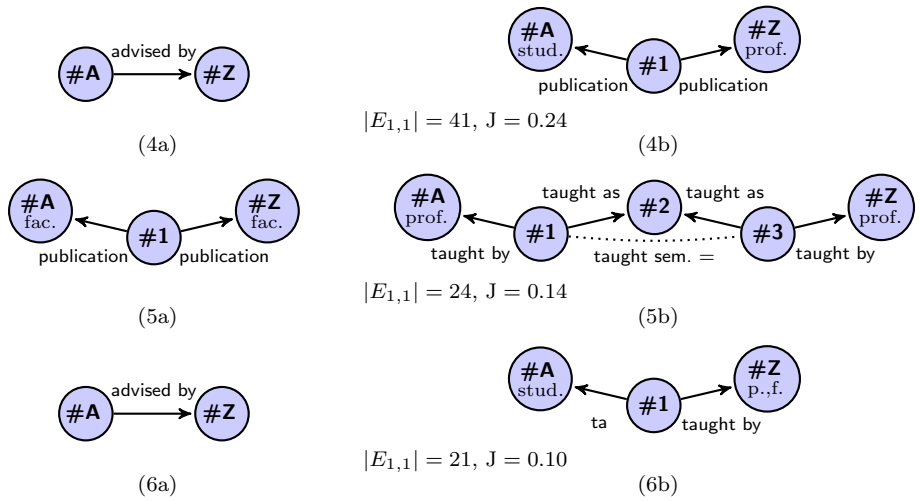


Fig. 9 Example redescriptions found from `UWCSE`. Node predicates are `professor`, `student` and `faculty`

over the best match found for edge predicate `kin14` alone, of accuracy 0.06 or the best pair of path queries similarly involving this predicate, of accuracy 0.17.

Figure 8 shows two examples of redescriptions found for `UMLS`: a perfect redescription, i.e. with accuracy 1, and a pair of symmetrical queries.

Finally, three redescriptions from `UWCSE` are displayed in Figure 9. Graphs (4a) and (6a) both represent the advisee–advisor relationship, with different matching queries. One states that they share a common publication (4b), the other that the advisee is a teaching assistant for the course taught by his supervisor (6b), forming two redescriptions of accuracy 0.24 and 0.10, respectively. The other redescription involves a coauthorship relation again, this time between faculty members (5a) and the matching query indicates that the persons of interest have taught different sessions of a same course (5b).

To summarize, we observe that our alternating scheme is able to find redescriptions that capture the relations existing in a network. It can exploit the added expressivity compared to individual queries to better elucidate the different uses of the same predicate and find more accurate patterns.

7 Conclusions

We have introduced the problem of relational redescription mining. As a solution, we proposed an alternating scheme with a novel efficient relational query miner based on frequent paths as its core.

We demonstrated that our query miner can find more complex queries than a baseline ILP approach, much faster. The proposed alternating scheme is able to capture the relations existing in a network with expressive redescriptions, as shown in experiments with three relational datasets.

The power of relational redescription mining should be investigated further on other datasets and application domains. On the algorithmic side, extending the

approach to probabilistic networks and considering queries of higher arity provide essential directions for future work.

Acknowledgements AK is a postdoctoral fellow of the Research Foundation Flanders (FWO Vlaanderen). Part of the work was done while EG was visiting KU Leuven.

References

Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives ZG (2007) DBpedia: A nucleus for a web of open data. In: Proceedings of the 6th International Semantic Web Conference (ISWC/ASWC 2007), pp 722–735

Carlson A, Betteridge J, Kisiel B, Settles B, Hruschka Jr ER, Mitchell TM (2010) Toward an architecture for never-ending language learning. In: Fox M, Poole D (eds) AAAI, AAAI Press

De Raedt L, Ramon J (2004) Condensed representations for inductive logic programming. In: 9th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, pp 438–446

Dehaspe L, Toivonen H (1999) Discovery of frequent DATALOG patterns. Data Min Knowl Discov 3(1):7–36

Dinh QT, Exbrayat M, Vrain C (2012) A link-based method for propositionalization. In: 22nd International Conference on Inductive Logic Programming, to appear

Esposito F, Malerba D, Semeraro G, Brunk C, Pazzani M (1994) Traps and pitfalls when learning logical definitions from relations. In: Ras ZW, Zemankova M (eds) ISMIS, Springer, LNCS, vol 869, pp 376–385

Galárraga L, Teflioudi C, Hose K, Suchanek FM (2013) AMIE: Association rule mining under imcomplete evidence in ontological knowledge bases. In: Proceedings of the 22th International Conference on World Wide Web, WWW 2013, to appear

Galbrun E, Kimmig A (2012) Towards finding relational redescriptions. In: Ganascia JG, Lenca P, Petit JM (eds) Discovery Science, Springer, Lecture Notes in Computer Science, vol 7569, pp 52–66

Galbrun E, Miettinen P (2012) From Black and White to Full Colour: Extending Redescription Mining Outside the Boolean World. Statistical Analysis and Data Mining 5(4):284–303

Gallo A, Miettinen P, Mannila H (2008) Finding subgroups having several descriptions: Algorithms for redescription mining. In: Proceedings of the SIAM International Conference on Data Mining, SDM 2008, pp 334–345

Goethals B, Van den Bussche J (2002) Relational association rules: Getting WARMeR. In: Hand DJ, Adams NM, Bolton RJ (eds) Pattern Detection and Discovery, Springer, LNCS, vol 2447, pp 125–139

Goethals B, Hoekx E, Van den Bussche J (2005) Mining tree queries in a graph. In: 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 61–69

Guns T, Nijssen S, De Raedt L (2011) Itemset mining: A constraint programming perspective. Artif Intell 175(12-13):1951–1983

Kuzelka O, Zelezný F (2009) Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In: Danyluk AP, Bottou

L, Littman ML (eds) ICML, ACM, ACM International Conference Proceeding Series, vol 382, p 72

Lao N, Cohen WW (2010) Relational retrieval using a combination of path-constrained random walks. Machine Learning 81(1):53–67

Lao N, Mitchell TM, Cohen WW (2011) Random walk inference and learning in a large scale knowledge base. In: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011), pp 529–539

Lavrac N, Zelezný F, Flach PA (2002) Rsd: Relational subgroup discovery through first-order feature construction. In: Matwin S, Sammut C (eds) ILP, Springer, Lecture Notes in Computer Science, vol 2583, pp 149–165

Miller RJ, Haas LM, Hernandez MA (2000) Schema mapping as query discovery. In: International Conference on Very Large Data Bases

Nebot V, Llavori RB (2012) Finding association rules in semantic web data. Knowledge-Based Systems 25(1):51–62

Ong IM, de Castro Dutra I, Page D, Santos Costa V (2005) Mode directed path finding. In: Gama J, Camacho R, Brazdil P, Jorge A, Torgo L (eds) ECML, Springer, LNCS, vol 3720, pp 673–681

Ramakrishnan N, Kumar D, Mishra B, Potts M, Helm RF (2004) Turning CARTwheels: An alternating algorithm for mining redescriptions. In: 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 266–275

Rettinger A, Lösch U, Tresp V, d'Amato C, Fanizzi N (2012) Mining the semantic web - statistical learning for next generation knowledge bases. Data Min Knowl Discov 24(3):613–662

Richards BL, Mooney RJ (1992) Learning relations by pathfinding. In: 10th National Conference on Artificial Intelligence, AAAI Press / The MIT Press, pp 50–55

Santos JCA, Tamaddoni-Nezhad A, Muggleton S (2009) An ILP system for learning head output connected predicates. In: Lopes LS, Lau N, Mariano P, Rocha LM (eds) EPIA, Springer, LNCS, vol 5816, pp 150–159

Shvaiko P, Euzenat J (2005) A survey of schema-based matching approaches. In: Spaccapietra S (ed) Journal on Data Semantics IV, Springer Berlin / Heidelberg, LNCS, vol 3730, pp 146–171

Suchanek FM, Kasneci G, Weikum G (2007) YAGO: a core of semantic knowledge. In: Proceedings of the 16th International Conference on World Wide Web (WWW 2007), pp 697–706

Suchanek FM, Abiteboul S, Senellart P (2011) PARIS: Probabilistic Alignment of Relations, Instances, and Schema. PVLDB 5(3):157–168

Wrobel S (1997) An algorithm for multi-relational discovery of subgroups. In: Komorowski HJ, Zytkow JM (eds) PKDD, Springer, Lecture Notes in Computer Science, vol 1263, pp 78–87

Yan X, Han J (2002) gSpan: Graph-based substructure pattern mining. In: 2nd IEEE International Conference on Data Mining, IEEE Computer Society, pp 721–724

Zhang C, Hoffmann R, Weld DS (2012) Ontological smoothing for relation extraction with minimal supervision. In: AAAI Conference on Artificial Intelligence