Towards Finding Relational Redescriptions

Esther Galbrun[1] and Angelika Kimmig[2]*

[1] Department of Computer Science and
Helsinki Institute for Information Technology HIIT
PO Box 68, FI-00014 University of Helsinki, Finland
esther.galbrun@cs.helsinki.fi

[2] Departement Computerwetenschappen, KU Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
angelika.kimmig@cs.kuleuven.be

Abstract. This paper introduces *relational redescription mining*, that is, the task of finding two structurally different patterns that describe nearly the same set of object tuples 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. As a first step towards solving this general task, we introduce an efficient algorithm that mines one description of a given binary concept. A set of graph patterns is built from frequent path patterns connecting example pairs. Experiments in the domain of explaining kinship terms show that this approach can produce complex descriptions that match explanations by domain experts, while being much faster than a direct relational query mining approach.

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 [1]. One way of creating such links is to find sets of objects together with their descriptions in different terminologies, as done in *redescription mining* [2, 3]. However, so far, this technique has only considered propositional or real valued attributes. As our first contribution, we extend redescription mining to the relational or network-based setting, that is, to sets of object *tuples* and their descriptions not only in terms of attributes of individual objects or nodes in a network, but also in terms of *relations* between them. Relational redescription mining thus provides a powerful data exploration technique, revealing structurally different connection patterns among objects.

As a concrete example of a redescription mining problem, we use the kinship terminology of the Alyawarra, an Australian indigenous community, where the goal is to describe kinship words in terms of family relationships. For instance, in Figure 1, the graph labeled $k16$ represents the kinship relation *Aleryia* between

* Also at University of Maryland, College Park, USA

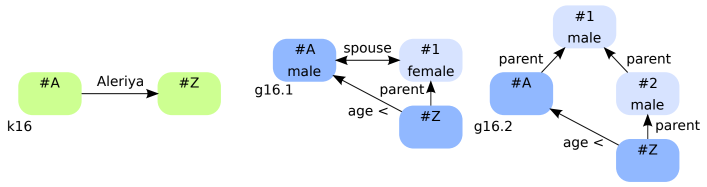

Fig. 1. Example of redescription of a kinship relation for pair $(\#A, \#Z)$ (graph labeled k16) in terms of two alternative genealogical connections (graphs labeled g16.1, g16.2).

the speaker $\#A$ and another person $\#Z$, corresponding to the relation between a father and his child, or between a person and his or her brother's child, as given by the graphs labeled $g16.1$ and $g16.2$, respectively.

Instead of finding two different descriptions simultaneously, redescription mining often takes an alternating approach: one description is fixed, the other one is updated, and roles are swapped in the next iteration [2]. In our setting, such a scheme relies on an approach to finding relational patterns, such as relational query mining [4, 5]. 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 all nodes of interest, thus producing many patterns that do not correspond to redescriptions. Hence, our second contribution is an efficient algorithm that finds one description for a given set of example pairs by first mining for path patterns that connect many example pairs, then combining those into more expressive graph patterns. This reduces the number of coverage tests needed by constructing queries based on the data. Our experiments in the kinship domain show that our approach can identify complex descriptions matching known ones, and is much faster than a basic relational query miner.

We proceed as follows. Section 2 introduces relational redescription mining, Section 3 discusses related work, Sections 4 and 5 present our algorithm and its experimental evaluation. We conclude and touch upon future work in Section 6.

2 Definitions and Notations

This paper introduces *relational redescription mining*, that is, the task of finding two structurally different patterns that describe nearly the same set of object tuples in a relational dataset. Informally, we view descriptions as sets (or disjunctions) of connected graphs expressed in terms of attributes of the data. For instance, graphs $g16.1$ and $g16.2$ in Figure 1 are an example of such a disjunctive pattern that describes the nodes of interest $\#A$ and $\#Z$ in terms of node attributes ($male$, $female$), relations between nodes ($spouse$, $parent$), and comparisons of node attributes ($age^{<}$). 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, which allows us to base the algorithm introduced in Section 4 on graph concepts.

We view *relational data* as a directed graph $(\mathcal{O}, \mathcal{R})$, where nodes correspond to the objects, and edges to relations 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, and we use node attributes $\mathcal{N} = \{sex, age\}$ and edge attributes $\mathcal{E} = \{kin, gen\}$, where *kin* maps into the set of kinship terms (cf. Table 1, Section 5) 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 patterns. The first type, a *node predicate* $\nu_{N_i}^{\mathcal{D}}(o)$, is true for an object o if and only if the node label $N_i(o)$ is defined and takes a value in the domain $\mathcal{D}$. The second type, an *edge predicate* $\epsilon_{E_i}^{\mathcal{D}}(o_1, o_2)$, 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 a value in $\mathcal{D}$. If $\mathcal{D}$ contains a single value d, we simply write $\nu_{N_i}^{d}(o)$ or $\epsilon_{E_i}^{d}(o_1, o_2)$. The third type, a *comparison predicate* $\phi_{N_i}^{rel}(o_1, o_2)$ 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, $g16.1$ in Figure 1 uses node predicates $\nu_{sex}^{female}(\#1)$ and $\nu_{sex}^{male}(\#A)$ along with edge predicates $\epsilon_{gen}^{parent}(\#Z, \#1)$, $\epsilon_{gen}^{spouse}(\#1, \#A)$ and $\epsilon_{gen}^{spouse}(\#A, \#1)$ and comparison predicate $\phi_{age}^{<}(\#Z, \#A)$.

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 clause* is a definite clause of the form $c(X_1, \ldots, X_m) : -b_1, \ldots, b_n$, where the body elements b_i are node, edge or comparison predicates, c is a special predicate denoting the pattern and the *query variables* $X_1, \ldots, X_m$ in the head also occur in the body. Instantiations of query variables are the object tuples of interest. We require graph clauses to be *linked*, meaning that the set of edge predicates in the body connects any two query variables (X_a, X_b). More formally, a graph clause is linked if for each pair of query variables (X_a, X_b) there is a sequence of variables $Z_0, \ldots, Z_k$ with $Z_0 = X_a$, $Z_k = X_b$, and 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 clause* is a graph clause with two query variables that are connected by an acyclic path consisting of all edge predicates in the body. A *description* or *pattern* is a set of graph clauses. We denote the set of attributes for which the body of clause C contains predicates by $att(C)$; for a pattern P, $att(P)$ is the union of the attribute sets of its clauses.

For instance, the middle graph in Figure 1 corresponds to the path clause

$$\begin{aligned} g16.1(\#A, \#Z) \quad :- \quad & \nu_{sex}^{male}(\#A), \epsilon_{gen}^{spouse}(\#A, \#1), \epsilon_{gen}^{spouse}(\#1, \#A), \\ & \nu_{sex}^{female}(\#1), \epsilon_{gen}^{parent}(\#Z, \#1), \phi_{age}^{<}(\#Z, \#A). \end{aligned}$$

This clause has query variables $\#A$ and $\#Z$ and is linked due to the spouse and parent edges.[3] Its attribute set is $\{sex, gen, age\}$.

As common in graph mining, we use subgraph isomorphism, or, in terms of logic, OI-subsumption [6], to match patterns against the data graph, that is, each variable in the pattern has to be matched to a different node in the graph, respecting the predicates in the clause body. 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 clause C is its support, $\text{supp}(C)$. With respect to a given set of example tuples $\mathcal{O}^+$, we define the *positive support* of a clause C as $\text{supp}^+(C) = \text{supp}(C) \cap \mathcal{O}^+$, and its *negative support* as $\text{supp}^-(C) = \text{supp}(C) \setminus \mathcal{O}^+$. We measure *similarity* of clauses C and C' using the Jaccard coefficient, that is, $sim(C, C') = |\text{supp}(C) \cap \text{supp}(C')| \,/\, |\text{supp}(C) \cup \text{supp}(C')|$.

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 a similarity threshold* δ*,* ***find*** *pairs of relational patterns* $(p_{\mathbf{A}}, p_{\mathbf{B}})$ *such that* $att(p_{\mathbf{A}}) \cap att(p_{\mathbf{B}}) = \emptyset$ *and* $sim(p_{\mathbf{A}}, p_{\mathbf{B}}) \geq \delta$*.*

One common strategy for mining redescriptions uses an alternating scheme [2]. That is, instead of searching for both patterns simultaneously, one pattern is fixed, the best corresponding pattern is determined, and used as the fixed pattern in the next round. In such a setting, relational redescription mining reduces to a sequence of relational learning tasks, where the support of the fixed pattern provides positive training examples (and other tuples could serve as negative training examples). In the remainder of this paper, we focus on the subtask of finding a good description given one pattern. We restrict our discussion to patterns of arity two. Redescriptions of higher arity could be obtained for instance by including additional body variables in the head of clauses, or by combining clauses of lower arity that share some, but not all, query variables.

3 Related Work

Relational redescription mining as introduced here is an extension of redescription mining, which so far has focused on propositional features [2] and real-valued attributes [3]. Redescription mining emphasizes the insights obtained from expressive, interpretable patterns and their instances in the given data rather than the classification of unseen data. Relational pattern languages are thus a natural candidate for redescriptions, but require adapted redescription mining algorithms tailored towards patterns that link objects of interest.

The frequentist approach used in our algorithm is inspired by graph mining techniques [7]. Transactional data mining aims at discriminating whole graphs

[3] Note that the $age^{<}$ edge in the graphical representation corresponds to a comparison predicate and is thus not considered for linkage.

based on the occurrence of subgraph patterns (cf. [8] and references therein). The present work instead seeks descriptions of node tuples in terms of their relations.

Learning relational patterns is a key task in multi-relational data mining and Inductive Logic Programming (ILP). Multi-relational query miners often follow a level-wise approach to mining, using 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 [4, 5]. While this principle results in connected clauses for unary patterns, frequent 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. In the context of cover-set based ILP systems such as Progol [9] and Aleph [10], this problem has been addressed by relational pathfinding [11, 12] and function learning [13]. 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 path clauses in our approach are similarly anchored in the data, but are directly selected based on their frequency across all examples. Query mining has also been extended to association rules with conjunctive heads [14], which can be seen as associations between conjunctive redescriptions, and to flexible numbers of query variables [15], which provides an interesting direction for finding redescriptions of arbitrary arity.

4 Algorithm

As outlined in Section 2, we take a first step towards relational redescription mining by addressing the following subproblem: Given a network and a set of positive object pairs, that can be obtained, in particular, by fixing one side of a redescription, find a relational pattern that accurately describes these examples.

We propose a two-phase approach that only considers linked patterns and reduces the number of costly subgraph isomorphism-based coverage tests. The first phase mines those path clauses that cover at least a given number of positive examples. The second phase constructs a relational pattern by combining path clauses with identical support into more general graph clauses and choosing a set of such clauses that accurately covers the examples.

We next discuss in turn how we obtain path clauses, how we create graph clauses from path clauses, and how we select graph clauses for the final pattern.

4.1 Mining Frequent Path Clauses

Our first intermediate goal is to obtain the set of path clauses that are frequent among the training examples, as any frequent graph pattern connecting nodes of interest has to be a combination of such paths. To do so, we extract paths (in terms of edge features) of increasing length that connect example pairs in

```
Input: A network $N$ with a set of positive example object pairs $\mathcal{O}^+$, a frequency threshold $\gamma$, and a maximum number of trials $\kappa$.
Output: A set of frequent paths clauses $\mathcal{C}$.
1:  $k \leftarrow 0$
2:  $\mathcal{P}_k \leftarrow$ paths of length 0, i.e., starting nodes in $\mathcal{O}^+$
3:  while $\mathcal{P}_k \neq \emptyset$ do
4:      $k \leftarrow k + 1$; $\mathcal{V} \leftarrow \emptyset$; $\mathcal{U} \leftarrow \emptyset$
5:      for each $P' \in \mathcal{P}_{k-1}$ do                  ▷ $P[i]$ is node at position $i$ in path $P$
6:          for each $n \in neighbors(P'[k-1])$ do
7:              $P \leftarrow P'$
8:              if $n \notin P$ then
9:                  $P[k] \leftarrow n$
10:                 if $(P[0], P[k]) \in \mathcal{O}^+$ then            ▷ example pair connected
11:                     $\mathcal{V} \leftarrow \mathcal{V} \cup \{P\}$
12:                 else
13:                     $\mathcal{U} \leftarrow \mathcal{U} \cup \{P\}$
14:     $\mathcal{F} \leftarrow$ FreqClauses$(\mathcal{V}, \gamma, \mathcal{O}^+, N)$
15:     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{F}$
16:     $\mathcal{O}_k \leftarrow \mathcal{O}_{k-1} \cup \bigcup_{C \in \mathcal{F}} \text{supp}^+(C)$
17:     if $k > \kappa$ and $\mathcal{O}_{k-\kappa} = \mathcal{O}_k$ then     ▷ no new example pair covered for $\kappa$ steps
18:         $\mathcal{P}_k \leftarrow \emptyset$
19:     else
20:         $\mathcal{P}_k \leftarrow \mathcal{V} \cup \mathcal{U}$
21: return $\mathcal{C}$
```

Fig. 2. FreqPaths: Mining frequent path clauses from a network.

the network, and add node features for nodes on the path as well as comparison features between nodes on the path. For each length, we align paths of that length and mine for frequent predicate sequences that maintain connectedness. Procedure FreqPaths in Figure 2 details this process.

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 5-13 extends paths in $\mathcal{P}_{k-1}$ to paths of length k, discards cyclic paths, and sorts the resulting paths into the sets $\mathcal{V}$ and $\mathcal{U}$ of paths connecting some example and other paths, respectively. The procedure FreqClauses in line 14 then produces frequent path clauses based on $\mathcal{V}$. Those are added to the set of clauses to be returned, and we keep track of the set $\mathcal{O}_k$ of covered examples for the termination criterion.

The inputs of FreqClauses are the set $\mathcal{V}$ of paths of length k connecting some example, the frequency threshold γ, the set $\mathcal{O}^+$ of example pairs, and the network N. Each path in $\mathcal{V}$ connecting nodes $o_0, o_1, \ldots, o_k$ can be represented as an ordered list of features according to the following principle:

$$\begin{array}{lll} & & (F_N(o_0), \\ F_E(o_0,o_1), F_E(o_1,o_0), & F_C(o_0,o_1), F_C(o_1,o_0), & F_N(o_1), \\ F_E(o_1,o_2), F_E(o_2,o_1), & F_C(o_0,o_2), F_C(o_2,o_0), F_C(o_1,o_2), F_C(o_2,o_1), & F_N(o_2), \\ & \ldots, & F_N(o_k)) \end{array}$$

That is, we start with the node features of the starting node, and then add for each following node o_i in order the edge features (in both directions) for the node o_i and its predecessor o_{i-1}, the comparison features (again in both directions) for o_i and all earlier nodes o_j, and the node features of o_i. For instance, the path $(\#A, \#1, \#Z)$ in the central graph of Figure 1 is represented as follows (we abbreviate m(ale), f(emale), p(arent), s(pouse), a(ge), and index sets by predicate type for better readability):

$$(\{m\}^N, \{s\}^E, \{s\}^E, \{\}^C, \{\}^C, \{f\}^N, \{\}^E, \{p\}^E, \{\}^C, \{a^<\}^C, \{\}^C, \{\}^C, \{\}^N)$$

Another example of a path of length two is

$$(\{\}^N, \{s\}^E, \{s\}^E, \{a^<\}^C, \{\}^C, \{f\}^N, \{\}^E, \{p\}^E, \{\}^C, \{a^<\}^C, \{\}^C, \{a^<\}^C, \{m\}^N)$$

Due to the simple example setting with few attributes, feature sets are small here; they can contain more elements in general. Given such path representations, FreqClauses mines for sequences of predicate sets respecting the linking constraint that cover more than γ pairs in $\mathcal{O}^+$. For instance,

$$(\{\}^N, \{s\}^E, \{s\}^E, \{\}^C, \{\}^C, \{f\}^N, \{\}^E, \{p\}^E, \{\}^C, \{a^<\}^C, \{\}^C, \{\}^C, \{\}^N)$$

covers both examples above, resulting in the path clause

$$\begin{aligned} c(\#A, \#Z) \quad :- \quad & \epsilon_{gen}^{spouse}(\#A, \#1), \epsilon_{gen}^{spouse}(\#1, \#A), \nu_{sex}^{female}(\#1), \\ & \epsilon_{gen}^{parent}(\#Z, \#1), \phi_{age}^{<}(\#Z, \#A). \end{aligned}$$

On the other hand, dropping the *parent* feature results in

$$(\{\}^N, \{s\}^E, \{s\}^E, \{\}^C, \{\}^C, \{f\}^N, \{\}^E, \{\}^E, \{\}^C, \{a^<\}^C, \{\}^C, \{\}^C, \{\}^N)$$

which does not qualify as a path clause, as it is no longer linked.

To solve this constrained sequence mining problem, one could for instance combine an off-the-shelf sequence mining tool with postprocessing to enforce linkage, design a special purpose algorithm, or exploit a declarative approach to mining patterns under constraints [16]. We use the latter approach, which allows us to obtain an efficient specialized miner without implementing it from scratch.

4.2 Combining Path Clauses into Graph Clauses

Our second intermediate goal is to combine several path clauses into one graph clause, which allows for more expressive patterns.

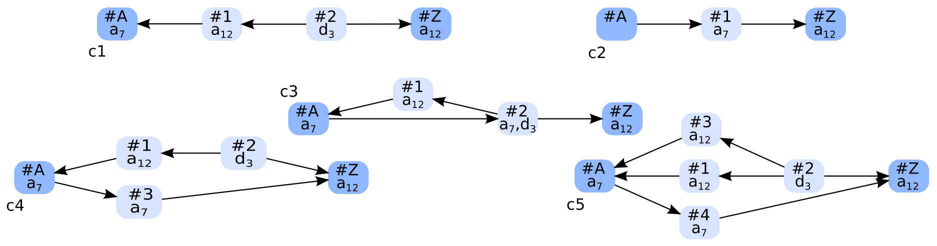

Fig. 3. Example of three graph clauses (c3-c5) combing path clauses c1 and c2.

As an illustration, Figure 3 depicts two path clauses c1 and c2 as well as three example graph clauses 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 clauses, and only finitely many among the valid clauses are supported by the data. Therefore, we merge paths based on their instantiations in the data rather than based on their clause representation. This ensures that we only construct valid clauses with non-empty support.

More specifically, given a set $\mathcal{K}$ of clauses with query variables $(\#A, \#Z)$ and a positive example (o_1, o_2) in the intersection of their supports, let $\{b_1, \ldots, b_n\}$ be the union of all instantiations of bodies of all clauses in $\mathcal{K}$ that map $(\#A, \#Z)$ to (o_1, o_2). Replacing each object in the ground clause $c(o_1, o_2) : -b_1, \ldots, b_n$ by a unique variable results in the unique *maximal clause* for $\mathcal{K}$.

Figure 4 illustrates this process. The graph with rectangular nodes represents the relevant part of the data network. We start with path clauses $c1$ and $c2$:

$$\begin{aligned} c1(\#A, \#Z) \quad &: - \quad \nu_a^7(\#A), \epsilon(\#1, \#A), \nu_a^{12}(\#1), \epsilon(\#2, \#1), \nu_d^3(\#2), \\ &\qquad \epsilon(\#2, \#Z), \nu_a^{12}(\#Z) \\ c2(\#A, \#Z) \quad &: - \quad \epsilon(\#A, \#1), \nu_a^7(\#1), \epsilon(\#1, \#Z), \nu_a^{12}(\#Z) \end{aligned}$$

The clause instantiations for object pair $(13, 82)$ are:

$$\begin{aligned} c1(13, 82) \quad &: - \quad \nu_a^7(13), \epsilon(44, 13), \nu_a^{12}(44), \epsilon(52, 44), \nu_d^3(52), \epsilon(52, 82), \nu_a^{12}(82) \\ c1(13, 82) \quad &: - \quad \nu_a^7(13), \epsilon(5, 13), \nu_a^{12}(5), \epsilon(52, 5), \nu_d^3(52), \epsilon(52, 82), \nu_a^{12}(82) \\ c2(13, 82) \quad &: - \quad \epsilon(13, 81), \nu_a^7(81), \epsilon(81, 82), \nu_a^{12}(82) \end{aligned}$$

Taking the union of clause bodies results in the ground clause

$$\begin{aligned} c(13, 82) \quad &: - \quad \nu_a^7(13), \epsilon(44, 13), \nu_a^{12}(44), \epsilon(52, 44), \nu_d^3(52), \epsilon(52, 82), \nu_a^{12}(82), \\ &\qquad \epsilon(5, 13), \nu_a^{12}(5), \epsilon(52, 5), \epsilon(13, 81), \nu_a^7(81), \epsilon(81, 82) \end{aligned}$$

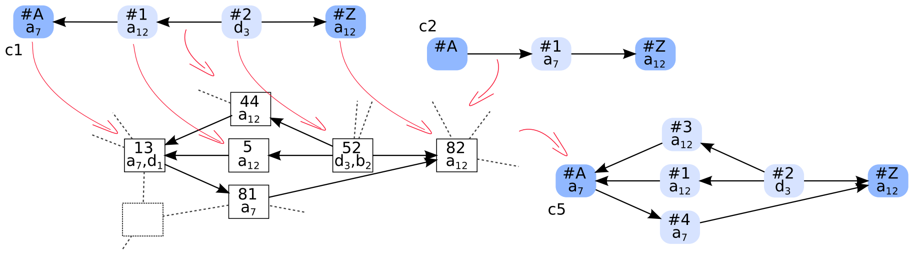

Fig. 4. Example of maximal clause construction; see Section 4.2 for details.

Finally, replacing constants by unique variables results in the maximal clause $c5$:

$$\begin{aligned} c5(\#A,\#Z) \quad :- \quad & \nu_a^7(\#A), \epsilon(\#3,\#A), \nu_a^{12}(\#3), \epsilon(\#2,\#3), \nu_d^3(\#2), \\ & \epsilon(\#2,\#Z), \nu_a^{12}(\#Z), \epsilon(\#1,\#A), \nu_a^{12}(\#1), \epsilon(\#2,\#1), \\ & \epsilon(\#A,\#4), \nu_a^7(\#4), \epsilon(\#4,\#Z) \end{aligned}$$

Note that #3 and #1 in $C5$ are *duplicate variables*, that is, they appear in the same node predicates and the same edge and comparison predicates with the same neighbors. While this is interesting from an expressivity point of view (as under OI subsumption, it implements counting), it also results in multiple clause 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 and always reduce maximal clauses to their *singular*. The singular of a clause C, denoted $\mathrm{sc}(C)$, is the clause obtained by keeping only one node from each set of duplicates in C. In our example, $\mathrm{sc}(C5) = C4$, cf. Figure 3.

4.3 Building Relational Patterns

Our third and last intermediate goal is to select an accurate set of graph clauses as the final description. The basic idea is to add clauses to the pattern that improve the coverage of at least a given number of examples, while reducing coverage of non-example pairs. We start by outlining some key concepts.

We partition the set of path clauses $\mathcal{C}$ into *equivalence classes* with respect to positive support, that is, we group together all clauses that cover the same set of positive examples. We only consider graph clauses constructed for paths in the same equivalence class, as these will cover the same examples.

We define an order $\prec$ on clauses as follows: $c_1 \prec c_2$ if and only if either $|\mathrm{supp}^-(c_1)| < |\mathrm{supp}^-(c_2)|$ or $|\mathrm{supp}^-(c_1)| = |\mathrm{supp}^-(c_2)| \wedge |\mathrm{supp}^+(c_1)| > |\mathrm{supp}^+(c_2)|$, that is, c_1 covers less negatives, or, if negative support is equal, more positives than c_2. For a given example O and a set of clauses S, the *best clause* $\mathrm{best}(O,S)$ is the $\prec$-minimal clause in S covering O. For a clause $C \in \mathcal{S}$, we define its *best support* as the set of objects for which C is the best clause in $\mathcal{S}$, that is, $\mathrm{supp}^*_{\mathcal{S}}(C) = \{O \in \mathcal{O} \mid \mathrm{best}(O,\mathcal{S}) = C\}$.

The key idea behind BuildPattern as outlined in Figure 5 is to add a clause C to the current pattern $\mathcal{S}$ if C is the best clause in $\mathcal{S} \cup \{C\}$ for at

```
    Input: A network with a set of positive examples $\mathcal{O}^+$, a set of path clauses $\mathcal{C}$, a minimum support threshold $\sigma$ and minimum support ratio $\theta$.
    Output: A relational pattern $P$.
 1: $\mathcal{S} \leftarrow \{C_2^{\emptyset}\}$
 2: $\mathcal{E} \leftarrow \{(\mathcal{O}_{\mathcal{M}}, \mathcal{M}) \mid \emptyset \subset \mathcal{M} \subseteq \mathcal{C} \wedge \forall\, C \in \mathcal{C} : (C \in \mathcal{M} \leftrightarrow \mathrm{supp}^+(C) = \mathcal{O}_{\mathcal{M}})\}$
 3: while $\mathcal{E} \neq \emptyset$ do
 4:     $(\mathcal{O}_{\mathcal{K}}, \mathcal{K}) \leftarrow \arg\max_{(\mathcal{O},C)\in\mathcal{E}} \sum_{O\in\mathcal{O}} \left|\mathrm{supp}^-(\mathrm{best}(O,\mathcal{S}))\right|$   ▷ most promising class
 5:     Remove $(\mathcal{O}_{\mathcal{K}}, \mathcal{K})$ from $\mathcal{E}$
 6:     if $\mathcal{K}$ has potential to improve the cover of more than $\sigma$ examples then
 7:         for $K \in \mathrm{sc}(\mathcal{K})$, in order of decreasing support do
 8:             if $\sigma \leq \left|\mathrm{supp}^*_{\mathcal{S}\cup\{K\}}(K)\right|$ then   ▷ sufficient improvement
 9:                 $\mathcal{S} \leftarrow \mathcal{S} \cup \{K\}$
10: $P \leftarrow \{K \in \mathcal{S} \mid (\sigma \leq |\mathrm{supp}^*_{\mathcal{S}}(K)|) \wedge (\theta \leq |\mathrm{supp}^*_{\mathcal{S}}(K)| \,/\, |\mathrm{supp}^-(K)|)\}$
11: return $P$
```

Fig. 5. BuildPattern: Selecting clauses to build a relational pattern.

least σ examples. Note that this criterion depends on the current pattern, and clauses added later on may decrease the importance of already present clauses. The algorithm maintains two sets: the set $\mathcal{S}$ of graph clauses in the current pattern (initially containing $C_2^{\emptyset}$, the empty clause of arity two), and the set $\mathcal{E}$ of clause equivalence classes that have not yet been processed. Once $\mathcal{E}$ is empty, $\mathcal{S}$ is post-processed to remove clauses that no longer pass the minimum support threshold σ, or that cover too many negative examples compared to their overall contribution, as measured by the minimum support ratio θ.

Constructing and evaluating graph clauses (line 7) is costly. To limit the number of processed equivalence classes, the algorithm therefore chooses the most promising equivalence class in $\mathcal{E}$, that is, the one with highest accumulated negative support for its examples' best clauses in the current pattern (line 4). Graph clauses constructed from this class are candidates to replace the current poor clauses associated to these examples.

Furthermore, we want to add those clauses to S that will be new best clauses for at least σ examples (line 8). Hence, we only process classes $(\mathcal{O}_{\mathcal{K}}, \mathcal{K})$ with sufficient potential for improvement, that is, if there are at least σ examples in $\mathcal{O}_{\mathcal{K}}$ whose current best clause covers negative examples or covers fewer than $|\mathcal{K}|$ pairs (line 6). If this is the case, we use the method discussed in the previous subsection to construct all singular clauses, order them by decreasing positive support, and add those clauses to $\mathcal{S}$ that actually improve the covers of more than σ examples (lines 7-9).

With this, we have all components of the overall algorithm in place. First, path clauses are mined using FreqPaths, cf. Section 4.1. These form the input to the selection procedure BuildPattern as detailed above. During selection, graph clauses are constructed as outlined in Section 4.2.

5 Experiments

We now evaluate our algorithm on the example task of finding genealogical patterns to explain kinship terminology, investigating the following two questions:

Q1 Does the proposed algorithm find accurate redescriptions?
Q2 How does our path based approach compare to a relational query mining approach, both in terms of pattern quality and running time?

We extracted data from the *Alyawarra Ethnographic Database*[4], which provides 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. As kinship terms involving deceased individuals included in the genealogy are unavailable, we restrict the evaluation to the 104 individuals with complete information, excluding kinship terms with less than three examples (identifiers 25 and 27) as well as "self" (24) and "dead" (28). For each kinship term in turn, the pairs $(\#A, \#Z)$ of individuals such that $\#A$ refers to $\#Z$ using the given term constitute the positive examples $\mathcal{O}^+$.

We implemented the algorithm in Python, using Fim Cp [16] to mine path clauses (FreqClauses). We consider edge predicates for the genealogical relation with values *spouse* and *parent*, node predicates for attribute *sex*, and relation $<$ for comparing values of node attribute *age*. We use frequency threshold $\gamma = 0.2$ and maximum number of trials $\kappa = 5$ for mining paths, while building patterns with minimum support ratio $\theta = 1$ and support threshold $\sigma = 3$.

Table 1 presents running times and quantitative results. The set $\mathcal{O}_k$ contains those positive examples that support some frequent path clause. For each final pattern P, we report the number of graph clauses included ($|P|$), the positive and negative support (supp^+ and supp^-, cf. Section 2), as well as the precision, recall and Jaccard coefficient, defined as $|\mathrm{supp}^+| / (|\mathrm{supp}^+| + |\mathrm{supp}^-|)$, $|\mathrm{supp}^+| / |\mathcal{O}^+|$ and $|\mathrm{supp}^+| / (|\mathcal{O}^+| + |\mathrm{supp}^-|)$, respectively. Note that we evaluate the quality of patterns on the training data because redescription mining is a descriptive approach that aims at characterizing as precisely as possible the data at hand using expressive and interpretable patterns, not at learning predictive patterns.

We observe that the algorithm found disjunctions of up to seven patterns, only failing to identify a discriminative pattern in four cases. Most of the patterns consist of a small number of clauses and reach relatively high precision. Running times vary from a couple of seconds up to about four minutes. They seem to depend on the number of frequent paths found and possible symmetries involved, that is, on the complexity of the pattern, more than on the number of examples. While the precision of the patterns is high, their recall is low, resulting in relatively poor Jaccard coefficient. This is likely due to terms not being restricted to pure kinship relations, but also taking a broader meaning, such as referring to a man older than oneself as *uncle*, in general. We created a filtered dataset by removing kinship terms between individuals who are further

[4] http://habc.eu/csac/wiki/knsrc/KinSources/AU01Alyawarra1971

Table 1. Quality statistics for redescriptions P of Alyawarra kinship terms: number of positive examples, of positive examples covered by a frequent path, of positive examples covered by P, and of negative examples covered by P, precision, recall, Jaccard coefficient, number of clauses in P, and running time.

	Kinship relation	$\lvert\mathcal{O}^+\rvert$	$\lvert\mathcal{O}_k\rvert$	$\lvert\text{supp}^+\rvert$	$\lvert\text{supp}^-\rvert$	Prec.	Rec.	Jacc.	$\lvert P\rvert$	Time (s)
(1)	Arengiya	228	15	0	0	0	0	0	0	1.60
(2)	Anyainya	489	243	123	4	0.968	0.251	0.249	3	40.45
(3)	Aidmeniya	231	113	30	4	0.882	0.129	0.127	4	21.55
(4)	Aburliya	379	59	24	7	0.774	0.063	0.062	3	3.44
(5)	Adardiya	493	91	21	1	0.954	0.042	0.042	2	2.64
(6)	Agngiya	508	199	138	2	0.985	0.271	0.27	3	56.02
(7)	Aweniya	453	231	127	12	0.913	0.28	0.273	5	67.43
(8)	Amaidya	817	92	92	1	0.989	0.112	0.112	2	1.85
(9)	Abmarliya	805	172	79	7	0.918	0.098	0.097	3	19.90
(10)	Awaadya	462	49	43	1	0.977	0.093	0.092	2	4.39
(11)	Anguriya	505	43	37	2	0.948	0.073	0.072	2	4.01
(12)	Adiadya	739	83	72	4	0.947	0.097	0.096	5	19.39
(13)	Angeliya	299	220	40	9	0.816	0.133	0.129	5	260.80
(14)	Algyeliya	447	205	36	4	0.9	0.08	0.079	2	180.09
(15)	Adniadya	43	30	9	3	0.75	0.209	0.195	1	55.51
(16)	Aleriya	943	384	277	26	0.914	0.293	0.285	5	153.23
(17)	Umbaidya	1256	364	276	7	0.975	0.219	0.218	3	163.26
(18)	Anowadya	392	61	61	3	0.953	0.155	0.154	2	0.55
(19)	Muriya	569	181	20	0	1	0.035	0.035	4	30.51
(20)	Agenduriya	13	9	0	0	0	0	0	0	18.68
(21)	Amburniya	272	118	94	19	0.831	0.345	0.323	7	8.19
(22)	Andungiya	142	58	20	8	0.714	0.14	0.133	3	3.88
(23)	Aneriya	193	85	0	0	0	0	0	0	10.56
(26)	Undyaidya	6	3	0	0	0	0	0	0	1.03

than 4 degrees apart in the genealogy. The Jaccard coefficient of patterns mined from this filtered data set was significantly higher, mostly above 0.7, supporting our conjecture. However, such filtering prevents the algorithm from learning descriptions involving longer genealogical chains, and is thus undesirable. Furthermore, in an alternating scheme addressing the full redescription mining task, the current set of examples is not necessarily the final one, but will be refined in subsequent iterations, which should eventually lead to higher accuracy.

Some examples of patterns found are shown in Figure 6. The *Awaadya* term (Figure 6 (a)) is simply equivalent to *Older Brother*, i.e., an older male sibling. The middle clause, $g10.1$, is an example that requires a graph clause, as a path clause connecting $\#A$ and $\#Z$ cannot express that they share both parents. The other clause, $g10.2$, describes the alternative case where the siblings have a common father but different mothers.

Two clauses were mined for the *Algyeliya* term (Figure 6 (b)). While both describe the relation to the daughter of one's paternal aunt, $g14.1$ focuses on the cases of female speakers, and $g14.2$ on the cases where the speaker is younger than the addressee. By distinguishing these special cases, the pattern outperforms the single graph that leaves out the restrictions on the speaker, which covers proportionally notably more negative examples. This is an example of the difficulty to select good representative patterns among numerous variants.

As a final example, our algorithm found three definitions for the *Umbaidya* term (Figure 6 (c)), suggesting that this term is used by mothers to refer to their

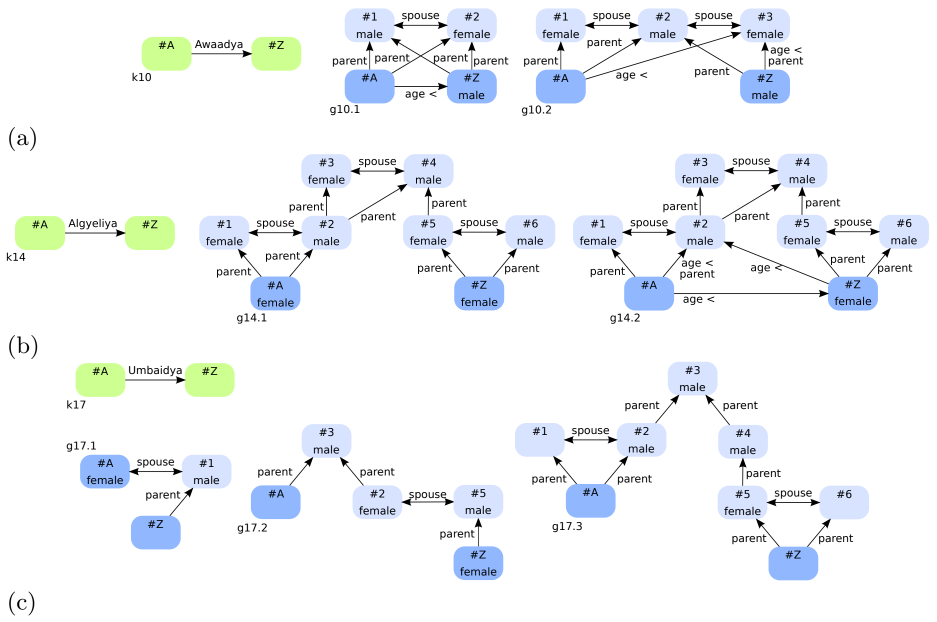

Fig. 6. Examples of kinship terms (graphs labeled k10, k14, k17) for pairs $(\#A, \#Z)$ described in terms of attributes and genealogical relations (remaining graphs).

child ($g17.1$), and by male and female speakers alike to refer to daughters of their sister ($g17.2$) or the children of their maternal uncle's daughter ($g17.3$). The first clause matches the ethnographic explanation provided for this term. The second clause differs from the second glossary entry, which restricts this structure to male speakers. The third clause has the same level of complexity as the last glossary entry, but a different structure. For most terms, our algorithm returned a pattern containing one or several clauses corresponding to the main definition provided for the term. In some cases, it found matching supplementary usage. In other cases, the additional usage found deviated from the provided explanation. Frequently, the deviation was an intermediate genealogical level or a difference in gender of some individual in the relation, as in the second clause above.

To summarize, this experiment affirmatively answers **Q1**, showing that our algorithm is able to find satisfactory patterns in this setting where various kinship usages of each term occur in the data, mixed with other, broader, usages that can not be explained in terms of genealogical links.

We now turn to the second question, the comparison to a relational query miner. More specifically, we use a modified version of `c-armr` [5] (implemented in Prolog) that mines top-k clauses with respect to the difference in support on positive and negative examples, that is, $score(p) = |supp^{+}(p)| - |supp^{-}(p)|$. We mine for top-5 clauses, using the same positive examples as above and all other pairs of nodes with full information as negative examples. As discussed in Section 3, the implementation does not ensure that query variables are linked.

To address this problem, we refine unlinked clauses 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 [11–13], but avoids the need to adapt the canonical refinement operator used in our implementation.[5]

As running times quickly become prohibitive due to large numbers of unlinked or non-discriminative clauses, we restrict the number of body literals to at most five. Under this restriction, running times per kinship term range from 5 to 11 minutes, thus illustrating that our path-based approach can provide a much faster alternative for mining relational patterns. Furthermore, as a direct consequence of this restriction, no pattern with positive score was found for six of the kinship terms, including *Algyeliya*, for which the path based approach finds the complex pattern illustrated in Figure 6 (b). Here, query nodes are four edge predicates apart, forcing linked clauses with at most five literals to be very general and thus not able to discriminate between positive and negative pairs.

In general, if the query miner finds clauses, those with highest score are of comparable quality to best single graph patterns found by the path-based approach. For instance, for *Awaadya*, the query miner finds seven best clauses, each covering 41 positives and 3 negatives. Six of them are subgraphs of g10.1 in Figure 6 (a), the last one adds an extra link to such a subgraph. For *Umbaidya*, the two highest scoring clauses, covering each 120 positives and 3 negatives, correspond to g17.2 in Figure 6 (c), but omitting one direction of the spouse link and all sex attributes except for either #2 or #5, respectively.

Concerning our second question **Q2**, these experiments thus indicate that, compared to a standard query mining approach, our path-based approach can find more complex descriptions of kinship terms much faster.

6 Conclusions and Future Work

We have introduced the problem of relational redescription mining as well as a first step towards a solution, an efficient algorithm for finding a description for a given set of node pairs. Disjunctions of graph clauses are constructed based on frequent path clauses, which guarantees that patterns are linked and reduces the need for expensive subgraph isomorphism tests. We demonstrated the effectiveness of our approach in the Alyawarra kinship domain, obtaining redescriptions of kinship terms through genealogical links that matched ethnographic explanations. Compared to a relational query miner, our algorithm found more complex descriptions much faster.

Important directions for future work include the thorough exploration of the algorithm's applicability and performance on various datasets, also in comparison to other approaches, the realization of an alternating scheme for relational redescription mining based on the algorithm developed here, and the extension to arbitrary numbers of query variables.

[5] We could not compare to GILPS [13], due to a bug in its function learning module.

Acknowledgments AK is a postdoctoral fellow of the Research Foundation Flanders (FWO Vlaanderen). Part of the work was done while EG was visiting KU Leuven. We thank Tias Guns and Matti Järvisalo for their help with Fim Cp.

References

1. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In Spaccapietra, S., ed.: Journal on Data Semantics IV. Volume 3730 of LNCS., Springer Berlin / Heidelberg (2005) 146–171
2. Ramakrishnan, N., Kumar, D., Mishra, B., Potts, M., Helm, R.F.: Turning CARTwheels: An alternating algorithm for mining redescriptions. In: 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM (2004) 266–275
3. Galbrun, E., Miettinen, P.: From Black and White to Full Colour: Extending Redescription Mining Outside the Boolean World. Statistical Analysis and Data Mining (2012) in press
4. Dehaspe, L., Toivonen, H.: Discovery of frequent DATALOG patterns. Data Min. Knowl. Discov. **3**(1) (1999) 7–36
5. De Raedt, L., Ramon, J.: Condensed representations for inductive logic programming. In: 9th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press (2004) 438–446
6. Esposito, F., Malerba, D., Semeraro, G., Brunk, C., Pazzani, M.: Traps and pitfalls when learning logical definitions from relations. In Ras, Z.W., Zemankova, M., eds.: ISMIS. Volume 869 of LNCS., Springer (1994) 376–385
7. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: 2nd IEEE International Conference on Data Mining, IEEE Computer Society (2002) 721–724
8. Kong, X., Yu, P.S.: Semi-supervised feature selection for graph classification. In: 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM (2010) 793–802
9. Muggleton, S.: Inverse entailment and Progol. New Generation Computing **13** (1995) 245–286
10. Srinivasan, A.: The Aleph Manual. University of Oxford. (2007)
11. Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In: 10th National Conference on Artificial Intelligenc, AAAI Press / The MIT Press (1992) 50–55
12. Ong, I.M., de Castro Dutra, I., Page, D., Santos Costa, V.: Mode directed path finding. In Gama, J., Camacho, R., Brazdil, P., Jorge, A., Torgo, L., eds.: ECML. Volume 3720 of LNCS., Springer (2005) 673–681
13. Santos, J.C.A., Tamaddoni-Nezhad, A., Muggleton, S.: An ILP system for learning head output connected predicates. In Lopes, L.S., Lau, N., Mariano, P., Rocha, L.M., eds.: EPIA. Volume 5816 of LNCS., Springer (2009) 150–159
14. Goethals, B., Van den Bussche, J.: Relational association rules: Getting WARMeR. In Hand, D.J., Adams, N.M., Bolton, R.J., eds.: Pattern Detection and Discovery. Volume 2447 of LNCS., Springer (2002) 125–139
15. Goethals, B., Hoekx, E., Van den Bussche, J.: Mining tree queries in a graph. In: 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM (2005) 61–69
16. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: A constraint programming perspective. Artif. Intell. **175**(12-13) (2011) 1951–1983