MDPI

Article

A Graph Query Framework for Relational Learning

Pedro Almagro Blanco [1], Fernando Sancho Caparrini [1] and Joaquín Borrego Díaz [1]*

[1] Dept. Ciencias de la Computación e inteligencia Artificial, Universidad de Sevilla
* Correspondence: jborrego@us.es

Abstract: Relational learning has attracted much attention of the machine learning community in recent years and many real-world applications have been successfully formulated as relational learning problems. Over the past few years, several relational learning algorithms have been presented following pattern-based approach. However, this kind of learning models suffers from two fundamental problems: the computational complexity arising from relational queries, and the lack of robust and general frameworks to serve as basis for relational learning methods. In this paper, we propose an efficient graph query framework allowing cyclic querying in polynomial time and ready to be used in pattern-based learning procedures. This solution uses logical predicates instead of graph isomorphisms for query evaluation, reducing the complexity and allowing query refinements through atomic operations. Examples of application show that the proposed framework allows to learn relational classifiers in a data-efficient manner with high expressive capacities. Specifically, relational decision trees are learned from sets of tagged subnetworks providing both classifiers and characteristic patterns for the identified classes.

Keywords: symbolic artificial intelligence; graph pattern matching; graph query; node classification ; subgraph classification; relational machine learning

1. Introduction

Usually, machine learning algorithms take a set of objects as input, each object described through a vector with numerical or categorical attributes, and produce (learn) a mapping from the input to output predictions: a class label, a regression value, an associated cluster or a latent representation, among others. In Relational Learning, relationships between objects are also taken into account during the learning process and data is represented as a graph composed by nodes (entities) and links (relationships), both with possible associated properties.

The fact that relational learning methods are able to learn from the connections between data makes them very powerful in different areas [10,13,17]. Learning to classify profiles in social networks based on their relationships with other items [30], to characterize proteins based on the functional connections that arise in organisms [18] and to identify molecules or molecular fragments with potential to produce toxic effects [6] represent some prominent examples of relational machine learning applications.

Non-relational machine learning has proven its effectiveness in many different tasks obtaining results close to, or even higher than, those obtained by humans [5]. However, by comparision, less success has been achieved with relational machine learning. Two of the most important reasons for this fact are the computational complexity arising from relational queries and the lack of robust and general frameworks to serve as basis for relational learning methods. On the one hand, most of the existing relational query systems are based on graph isomorphisms and their computational complexity is NP-complete, affecting efficiency of learning methods that use them [21]. On the other hand, most of existing query systems do not allow atomic operations to expand queries in a partitioned way, preventing learning systems from making efficient searches in the query space [20].

Citation: Almagro Blanco, P.; Sancho Caparrini, F.; Borrego Díaz, J. A Graph Query Framework for Relational Learning.. *Information* **2023**, *1*, 0. https://doi.org/

Received:
Revised:
Accepted:
Published:

The novel graph query framework presented in this paper tries to solve these two fundamental problems. The goal is to obtain a query system allowing graph pattern matching under controlled complexity and providing step-by-step pattern expansion using well defined operations. A framework that meets these requirements is suitable for use in relational machine learning procedures because, combined with the appropriate exploration techniques, it allows the automatic extraction of characteristic relational patterns from data.

The rest of this paper is organized as follows. In Section 2 we give an overview of some work related to this research. Section 3 presents the novel graph query framework, showing main definitions and properties that ensure its utility, and also some representative query examples and analysis of the computational complexity arising from the model. We continue by describing the application of this framework to perform relational machine learning in Section 4. Finally, Section 5 presents some conclusions that can be derived from this work and some future lines worth to be looking at.

2. Related work

A usual way to perform relational queries is through the definition of patterns in some abstract representation of the data and then looking for their occurrences in real datasets. This working method falls within Graph Pattern Matching, an area that has been active for more than 30 years. Depending on different aspects to consider, some usual distinctions on pattern matching methods are: (a) *Structural vs. Semantic*, (b) *Exact vs. Inexact*, and (c) *Optimal vs. Approximate* [12]. One more distinction can be made depending on the relation to be matched by the pattern and the subgraphs to be considered as its occurrences: (d) graph pattern matching based on *Isomorphisms*, *Graph Simulation*, and *Bounded Simulation*, among others [11,23,31]. Our proposal falls within semantic, exact and optimal graph pattern matching. While query systems based on graph isomorphism present NP complexity, those based on simulations present polynomial complexity [16]. However, both types are based on relations between the set of elements in the query and the set of elements in the graph data, which prevents the evaluation of non-existence of elements.

There exists two fundamentally different kinds of relational learning models [25]. The first one, *the latent feature approach*, is based on latent feature learning, such as tensor factorization and neural models, and usually works well with uncertainty in a probabilistic approximation [4,7,17]. The second one, *the graph-pattern based approach*, is based on automatically extracting characteristic relational patterns (or observable graph patterns) from data [14,22]. As the framework presented in this work belongs to the second one, in the next lines we will focus on reviewing relational learning methods with a graph-pattern based approach as well as the query systems on which they are based.

The majority of pattern-based relational learning procedures derive from Inductive Logic Programming (ILP) [26]. While ILP itself (without a proper transformation of data relationships into logical predicates) does not provide relational classifiers, it does allow the automatic generation of logical decision trees that could handle relational predicates. Logical decision trees are binary decision trees where all tests in internal nodes are expressed as conjunctions of literals of a prefixed first-order language. Top-down Induction of Logical Decision Trees (TILDE) is a representative algorithm able to learn this kind of decision trees from a set of examples [3]. TILDE represents a general framework for the induction of logical decision trees which, after appropriate transformations, can be used for the induction of relational decision trees. However, as it is not oriented to relational learning, it lacks a set of operators that allow to refine relational queries properly.

Multi-relational decision tree learning (MRDTL [22]) is a relational learning algorithm supported in *Selection Graphs* [20], a graph representation of SQL queries that selects records from a relational database matching some defined constraints. Selection graphs allows atomic operations to refine queries, but can only evaluate nodes, and lacks a robust formalization allowing reasoning about their properties of refinement and complementarity. MRDTL represents a relational decision tree induction method allowing aotmic specializations of queries (*selection graphs*) but it can only learn classifications of nodes and can not

detect cyclic patterns. Our graph query framework can be used in this kind of learning tasks, allowing the learning of the classification of any structure through patterns that may contain cycles.

Another noteworthy pattern-based relational learning technique is Graph-Based Induction of Decision Trees (DT-GBI [24]), a decision tree construction algorithm for learning graph classifiers using graph-based induction (GBI), a data mining technique to extract network motifs from labeled graphs by joining pairs of connected nodes. In DT-GBI the attributes (called patterns or substructures) are generated during the execution of the algorithm [14].

As we have seen, some pattern-based approaches are able to learn to classify complete graphs and some others construct node classifiers, our proposal supports to learn from general subgraphs as base cases. In addition, our approach is able to perform cyclic queries allowing learning procedures to extract cyclic patterns from data.

3. Graph query framework

We are interested in graph queries that allow atomic specializations. Given a set of elements matching certain relational pattern, we want pattern specializations that select only a subset of those elements. As mentioned above, Selection Graphs is an example of this type of query tool and it was also created to be used in relational learning procedures, but it presents a fundamental handicap limiting the expressiveness of queries: its patterns can not contain cycles. In addition, as a graphical representation of SQL queries, it inherits the efficiency problems of the systems based on SQL technology when applied on high relational data. Our proposal takes some ideas from this approach, avoiding its limitations.

In addition to query specialization, we look for generating complementary queries when new conditions are applied. It will be useful to explore the pattern space and to characterize elements in a top-down manner. More specifically, given one query, we look for a set of specialized queries that forms embedded partitions of it.

We want to emphasize that our main goal is to provide a formalization and application examples of the model, but with the secondary goal of providing a real implementation to be usable from a practical point of view[1].

3.1. Preliminaries

Some preliminary concepts in the definition of graph queries are presented here. The reader can find a more complete review in [1].

Let us start with a graph definition covering several ones common in literature (directed/undirected graphs, multi-relational graphs, hypergraphs, etc.) and that will be useful as a common base both for stating structures of general graph datasets and queries on them.

Definition 3.1. *A* Graph *is a tuple* $G = (V, E, \mu)$ *where:*

- V *and* E *are sets, called, respectively,* set of nodes *and* set of edges *of* G.
- μ *associates each node/edge in the graph with a set of properties,* $\mu : (V \cup E) \times R \to J$, *where* R *represents the set of* keys *for properties, and* J *the set of* values.

In addition, we require the existence of a special key for the edges of the graph, called incidences *and denoted by* γ, *which associates every edge in* E *to a set of nodes in* V.

Usually, for each $\alpha \in R$ and $x \in V \cup E$, we write $\alpha(x)$ instead of $\mu(x, \alpha)$, handling properties as maps from nodes/edges to values. Note that, unlike traditional definitions, the elements in E are symbols representing the edges, and not pairs of elements from V, and γ is the function that associates every edge to the tuple (ordered or not) of nodes that it connects.

[1] https://github.com/palmagro/ggq

We will also write $\gamma(v)$ to denote the edges where node $v \in V$ participates, and the *neighborhood* of v is the set of nodes, including itself, connected with it, i.e.: $\mathcal{N}(v) = \bigcup_{e \in \gamma(v)} \gamma(e)$.

We should give some notions about the position of a node in an edge. We present a simpler definition of *position*, but a more complete one can be given to accomplish between *directed* and *undirected* edges:

Definition 3.2. *If $e \in E$ and $\gamma(e) = (v_1, \ldots, v_n) \in V^n$, then for each $v_i \in \gamma(e)$ we define its* position *in e as $ord_e(v_i) = i$. We denote $u \leq_e v$ to indicate that $ord_e(u) \leq ord_e(v)$.*

From this order between the nodes in an edge we can define *paths* in a graph:

Definition 3.3. *Given $G = (V, E, \mu)$, the set of* paths *in G, denoted by $\mathcal{P}_G$, is defined as the minimal set verifying:*

1. *If $e \in E$ and $u,\ v \in \gamma(e)$ with $u \leq_e v$, then $\rho = u \xrightarrow{e} v \in \mathcal{P}_G$. We will say that ρ* connects *the nodes u and v of G, and we will denote it by $u \overset{\rho}{\rightsquigarrow} v$.*
2. *If $\rho_1,\ \rho_2 \in \mathcal{P}_G$, with $u \overset{\rho_1}{\rightsquigarrow} v$ and $v \overset{\rho_2}{\rightsquigarrow} w$ then $\rho_1 \cdot \rho_2 \in \mathcal{P}_G$, with $u \overset{\rho_1 \cdot \rho_2}{\rightsquigarrow} w$.*

Some useful notations are:

- If $u \overset{\rho}{\rightsquigarrow} v$, then we write $\rho^o = u$ and $\rho^i = v$.
- We denote the paths *through u*, *starting in u*, and *ending in u*, respectively, by:

$$\mathcal{P}_u(G) = \{\rho \in \mathcal{P}(G) :\ u \in \rho\}$$

$$\mathcal{P}^o_u(G) = \{\rho \in \mathcal{P}(G) :\ \rho^o = u\}$$

$$\mathcal{P}^i_u(G) = \{\rho \in \mathcal{P}(G) :\ \rho^i = u\}$$

The notion of subgraph is obtained as usual by imposing that the properties are also maintained in the common elements.

Definition 3.4. *A subgraph of $G = (V, E, \mu)$ is a graph $S = (V_S, E_S, \mu_S)$ where $V_S \subseteq V$, $E_S \subseteq E$ and $\mu_S \subseteq \mu_{|V_S \cup E_S}$. We denote $S \subseteq G$.*

3.2. Graph queries

As it was stated, one of the aims of our graph query framework is the ability to obtain complementary queries to a given one. This means that if a subgraph does not verify a query it must always verify one of its complementaries. Since projection prevents from evaluating non existence of elements (something that we will need to reach complementarity), our proposal will be based on logical predicates instead of projections.

In the following we consider a prefixed graph $G = (V, E, \mu)$. We will briefly formalize what we understand concretely by a predicate for G.

Consider Θ, a collection of function, predicate, and constant symbols, containing all the properties from μ together with constants associated with elements of G and possibly some additional symbols (for example, metrics defined on G, such as *degree*). We can consider the first-order language with equality, L, making use of Θ as a set of non logical symbols. In this situation, a *predicate* on G is an element of the set of first-order formulas of L ($Form(L)$). $Form^2(L)$ will denote the binary predicates on G.

Definition 3.5. *A* Query *for G is a graph, $Q = (V_Q, E_Q, \mu_Q)$, with α and θ properties in μ_Q, such that:*

- $\alpha : V_Q \cup E_Q \to \{+, -\}$.
- $\theta : V_Q \cup E_Q \to Form^2(L)$.

Formally, Q depends on L and G, but as we will consider L and G to be prefixed, we will write $Q \in \mathcal{Q}$ (instead of $Q \in \mathcal{Q}(L, G)$) to denote that Q is a query for G using L. Note that, once a query is defined, it can be applied to several graphs sharing the same language.

Intuitively, in the semantics associated with a query we will use the second input of the binary predicates to impose membership restrictions on subgraphs of G, whereas the first input must receive elements of the corresponding type to which it is associated.

For example, if $a, b \in V_Q$ and $e \in E_Q$, and (we will denote $\theta_x := \theta(x)$):

$$\theta_a(v, S) := v \in S$$
$$\theta_b(v, S) := \exists z \in S \ (z \rightsquigarrow v)$$
$$\theta_e(\rho, S) := \exists y, z (y \overset{\rho}{\rightsquigarrow} z \wedge y \notin S \wedge z \in S)$$

$\theta_a(v, S)$ is defined for nodes, and it checks if $v \in V$ is contained in the subgraph S under evaluation. $\theta_b(v, S)$ is defined for nodes too, and it is verified when there is a path in G connecting a node of S with $v \in G$. Finally, $\theta_e(\rho, S)$ is defined for paths, and it is verified when the evaluated path, $\rho \in \mathcal{P}_G$, connects S with its outward (in G).

Given a query under the above conditions, x^+ (resp. x^-) will denote $\alpha(x) = +$ (resp. $\alpha(x) = -$), and V_Q^+ / V_Q^- (resp. E_Q^+ / E_Q^-) are the set of positive/negative nodes (resp. edges). If θ_x is not explicitly defined for an element, we assume it to be a tautology.

As next definition states, positive elements add existence constraints to queries, while negative elements add non-existence constraints. More specifically, every positive/negative node in a query requests the existence/non-existence of a node in G fulfilling its conditions (imposed by θ_x and its edges):

Definition 3.6. *Given $S \subseteq G$, and $Q \in \mathcal{Q}$, we say that S* matches *Q ($S \vDash Q$), if the next formula is verified:*

$$Q(S) = \bigwedge_{n \in V_Q} Q_n^{\alpha(n)}(S)$$

where, for each node, $n \in V_Q$:

$$Q_n^+ = Q_n, \qquad Q_n^- = \neg Q_n$$
$$Q_n(S) = \exists v \in V \left(\bigwedge_{e \in \gamma(n)} Q_{e^*}^{\alpha(e)}(v, S) \right)$$

and, for each edge, $e \in E_Q$, $* \in \{o, i\}$:

$$Q_{e^*}^+ = Q_{e^*}, Q_{e^*}^- = \neg Q_{e^*}$$
$$Q_{e^o}(v, S) = \exists \rho \in \mathcal{P}_v^o(G) \left(\theta_e(\rho, S) \wedge \theta_{e^o}(\rho^o, S) \wedge \theta_{e^i}(\rho^i, S) \right)$$
$$Q_{e^i}(v, S) = \exists \rho \in \mathcal{P}_v^i(G) \left(\theta_e(\rho, S) \wedge \theta_{e^o}(\rho^o, S) \wedge \theta_{e^i}(\rho^i, S) \right)$$

A generic query example is shown in Figure 1.

Unlike other previous graph query systems we can highlight that these queries satisfy: (1) may contain cycles; (2) can evaluate subgraphs; (3) edges in the query can be projected onto paths in the graph; (4) can evaluate structural and/or semantic characteristics; and, furthermore, (5) allow specialization through atomic operations (as we will see in the next section).

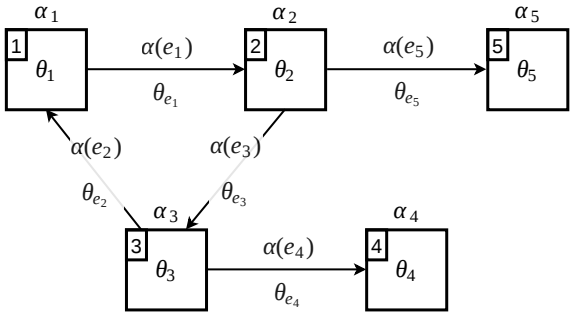

Figure 1. Graph Query Example

3.3. Refinement sets

In order to characterize elements in a graph, we need to obtain computationally effective ways for building queries by using basic operations. In this section we will present a query construction method ready to be used in relational learning tasks. First, let's define a concept about relative refinements between queries.

Definition 3.7. *Given* $Q_1, Q_2 \in \mathcal{Q}$*, we say:*

1. Q_1 refines Q_2 *in* G *(*$Q_1 \preceq_G Q_2$*) if:* $\forall S \subseteq G\ (S \vDash Q_1 \Rightarrow S \vDash Q_2)$.
2. *They are* equivalent *in* G *(*$Q_1 \equiv_G Q_2$*) if:* $Q_1 \preceq_G Q_2$ *and* $Q_2 \preceq_G Q_1$.

Two queries are equivalent when both are verified exactly by the same subgraphs. It is easy to prove the following result, which tells that $\preceq_G$ generates a partial order relationship on $\mathcal{Q}/\equiv_G$:

Theorem 1. $\preceq_G$ *is a partial order in* $\mathcal{Q}$*. That is, for every* $Q_1, Q_2, Q_3 \in \mathcal{Q}$*:*

1. $Q_1 \preceq_G Q_1$.
2. $Q_1 \preceq_G Q_2 \wedge Q_2 \preceq_G Q_1 \Rightarrow Q_1 \equiv_G Q_2$.
3. $Q_1 \preceq_G Q_2 \wedge Q_2 \preceq_G Q_3 \Rightarrow Q_1 \preceq_G Q_3$.

Next, we study the relationship between the topological structure of a query and its functionality as a predicate on subgraphs. In general, it is hard trying to extract logical properties of the predicate from the structural properties of the graph that represents it, but we can obtain some useful conditions to manipulate the structures and modify the semantics of the query in a controlled way.

Definition 3.8. *Given* $Q_1, Q_2 \in \mathcal{Q}$*, we say that* Q_1 *is a* Q^--conservative extension *of* Q_2 *(*$Q_2 \subseteq^- Q_1$*) if:*

1. $Q_2 \subseteq Q_1$.
2. $\forall n \in V^-_{Q_2}\ \forall e \in \gamma_{Q_1}(n)\ \exists e' \in \gamma_{Q_2}(n)\ (Q_e \equiv Q_{e'})$.

Figure 2 shows an example of a Q^--conservative extension. The new element in the right query imposes new constraints on the positive node but does not add any further constraints to the negative one.

Since negative nodes add non-existence constraints to subgraph verification, Q^--conservative extensions ensure that no new constraints are added to them. Hence:

Theorem 2. *If* $Q_2 \subseteq^- Q_1$ *then* $Q_1 \preceq Q_2$.

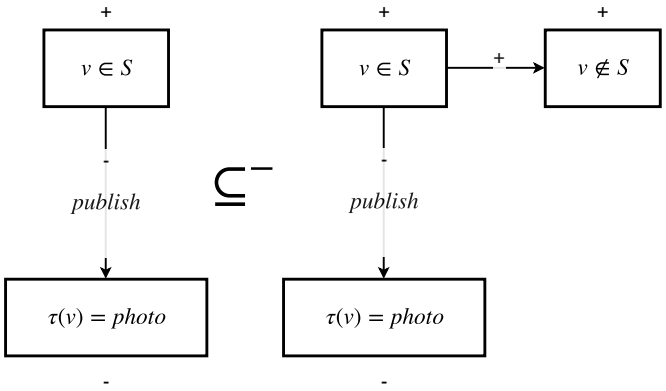

Figure 2. Q^--conservative extension

Proof. Since predicates associated to edges depend only on the information in the edge itself (which considers the value of θ in its incident nodes, regardless of their α value), we can state that:

$$\forall e \in E_{Q_2} \; (Q_{1e}^{\alpha(e)} = Q_{2e}^{\alpha(e)})$$

Considering this fact, we analyze how the predicates associated with the nodes for both queries behave:

- If $n \in V_{Q_2}^-$, since $Q_2 \subseteq^- Q_1$, then $Q_{1n}^- = Q_{2n}^-$.
- If $n \in V_{Q_2}^+$, then $Q_{1n}^+ \rightarrow Q_{2n}^+$, because ($\gamma_1$, γ_2 are the incidence functions of Q_1 and Q_2, respectively):

$$\begin{aligned} Q_{1n}^+ &= \exists v \in V \left(\bigwedge_{e \in \gamma_1(n)} Q_{1e}^{\alpha(e)} \right) \\ &= \exists v \in V \left(\bigwedge_{e \in \gamma_1(n) \cap E_{Q_2}} Q_{1e}^{\alpha(e)} \; \wedge \bigwedge_{e \in \gamma_1(n) \setminus E_{Q_2}} Q_{1e}^{\alpha(e)} \right) \\ &= \exists v \in V \left(\bigwedge_{e \in \gamma_2(n) \cap E_{Q_2}} Q_{2e}^{\alpha(e)} \; \wedge \bigwedge_{e \in \gamma_1(n) \setminus E_{Q_2}} Q_{1e}^{\alpha(e)} \right) \\ &\rightarrow Q_{2n}^+ \end{aligned}$$

Hence:

$$\begin{aligned} Q_1 &= \bigwedge_{n \in V_{Q_1}} Q_{1n}^{\alpha(n)} = \bigwedge_{n \in V_{Q_2}} Q_{1n}^{\alpha(n)} \; \wedge \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} Q_{1n}^{\alpha(n)} \\ &= \bigwedge_{n \in V_{Q_2}^+} Q_{1n}^{\alpha(n)} \; \wedge \bigwedge_{n \in V_{Q_2}^-} Q_{1n}^{\alpha(n)} \; \wedge \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} Q_{1n}^{\alpha(n)} \\ &\rightarrow \bigwedge_{n \in V_{Q_2}^+} Q_{2n}^{\alpha(n)} \; \wedge \bigwedge_{n \in V_{Q_2}^-} Q_{2n}^{\alpha(n)} \; \wedge \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} Q_{1n}^{\alpha(n)} \\ &= \bigwedge_{n \in V_{Q_2}} Q_{2n}^{\alpha(n)} \; \wedge \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} Q_{1n}^{\alpha(n)} \\ &\rightarrow Q_2 \end{aligned}$$

□

Previous result suggests that a query can be refined by adding nodes (of any sign) and edges to the existing positive nodes, but because of the (negated) interpretation of predicates associated with negative nodes, care must be taken to maintain their neighborhood to be sure that adding more edges does not weaken the imposed conditions (that, consequently, will not provide refined predicates).

In order to obtain controlled methods for query generation, in the following we will provide operations to refine queries by unit steps. To do this, we define the cloning operation, that makes copies of existing nodes and clones all the edges incident on them (and between them, when several nodes connected in the original graph are cloned):

Definition 3.9. *Given $G = (V, E, \mu)$, and $W \subseteq V$, we define the* clone of G by duplication of W, Cl_G^W, *as:*

$$Cl_G^W = (V \cup W', E \cup E', \mu \cup \{(n', \mu(n))\}_{n \in W} \cup \{(e', \mu(e))\}_{e' \in E'})$$

where $W' = \{n' \ : \ n \in W\}$ are new cloned nodes, and E' is a set of new edges obtained from incident edges on nodes of W where nodes of W are replaced by copies of W' (edges connecting original nodes with cloned nodes and edges connecting cloned nodes, are cloned).

Figure 3 shows an example of a cloned graph by duplicating two nodes (in the original graph, left side, the set of nodes to be duplicated is highlighted).

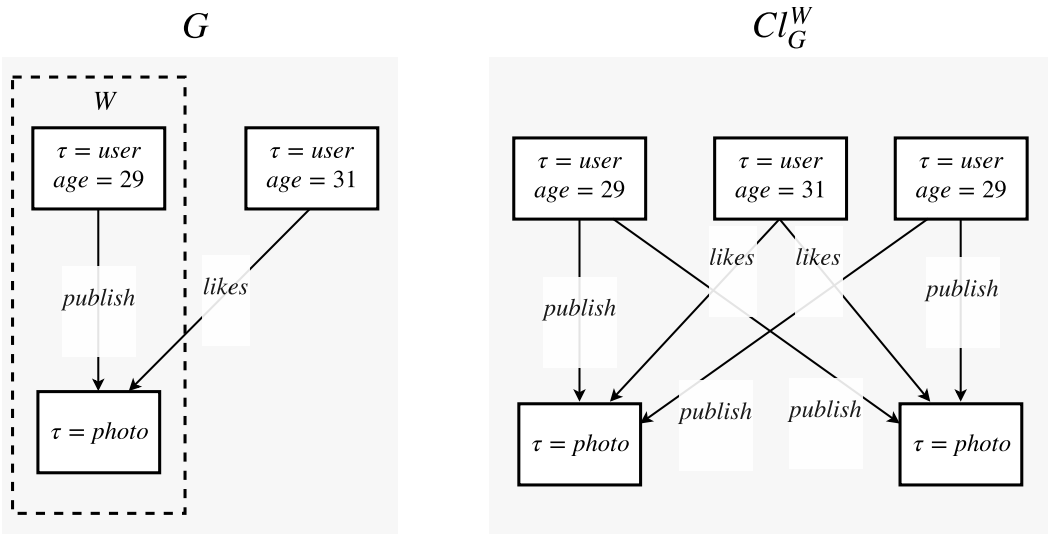

Figure 3. Clone of a graph by duplication

Next result shows that cloning positive nodes does not alter the *meaning* of the queries.

Theorem 3. *If $W \subseteq V_Q^+$, then $Cl_Q^W \equiv Q$.*

Proof. Let us denote $Q_1 = CI_Q^W$. Then:

$$\begin{aligned} Q_1 &= \bigwedge_{n \in V_{Q_1}} Q_{1n}^{\alpha(n)} = \bigwedge_{n \in V_Q} Q_{1n}^{\alpha(n)} \wedge \bigwedge_{n \in W} Q_{1n'}^{\alpha(n')} \\ &= \bigwedge_{n \in V_Q \setminus \gamma_Q(W)} Q_{1n}^{\alpha(n)} \wedge \bigwedge_{n \in \gamma_Q(W)} Q_{1n}^{\alpha(n)} \wedge \bigwedge_{n \in W} Q_{1n'}^{\alpha(n')} \\ &= \bigwedge_{n \in V_Q \setminus \gamma_Q(W)} Q_n^{\alpha(n)} \wedge \bigwedge_{n \in \gamma_Q(W)} Q_n^{\alpha(n)} \wedge \bigwedge_{n \in W} Q_n^{\alpha(n)} \\ &= Q \end{aligned}$$

□

Looking for complementary sets of selected subgraphs when refining a query, we define the central concept of refinement set:

Definition 3.10. *Given $Q \in \mathcal{Q}$, $R \subseteq \mathcal{Q}$ is a* refinement set *of Q in G if:*

1. $\forall\, Q' \in R\ (Q' \preceq_G Q)$
2. $\forall\, S \subseteq G\ (S \vDash Q \Rightarrow \exists!\, Q' \in R\ (S \vDash Q'))$

Let us present now refinement sets to obtain expressive queries from simpler ones . $Q \in \mathcal{Q}$ is prefixed, and $\top$ states for tautology:

Theorem 4. *(Add new node) If $m \notin V_Q$, the set $Q + \{m\}$, formed by:*

$$\begin{aligned} Q_1 &= (V_Q \cup \{m\},\ E_Q,\ \alpha_Q \cup (m, +),\ \theta_Q \cup (m, \top)) \\ Q_2 &= (V_Q \cup \{m\},\ E_Q,\ \alpha_Q \cup (m, -),\ \theta_Q \cup (m, \top)) \end{aligned}$$

is a refinement set of Q in G (Fig. 4).

Proof. We must verify the two necessary conditions for refinement sets:

1. Since $Q \subseteq^- Q_1$ and $Q \subseteq^- Q_2$, thus $Q_1 \preceq Q$ and $Q_2 \preceq Q$.
2. Given $S \subseteq G$ such that $S \vDash Q$. Then:

$$\begin{aligned} Q_1 &= Q \wedge Q_m \\ Q_2 &= Q \wedge \neg Q_m \end{aligned}$$

where $Q_m = \exists v \in V\ (\top)$.
If $G \neq \emptyset$, then $S \vDash Q_1$ and $S \nvDash Q_2$.
If $G = \emptyset$, then $S \nvDash Q_1$ and $S \vDash Q_2$.

□

Since $G \neq \emptyset$ (usually), $Q_1 \equiv Q$. However, although we obtain an equivalent query, this operation will be useful to add new nodes and adding new restrictions later.

The second refinement allows to create edges between existing query nodes. In order to get a valid refinement set the addition of edges is restricted to positive nodes. In the following, node marked with positive/negative sign represents cloned node whose α property has been assigned to positive/negative.

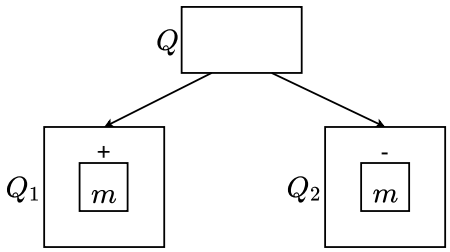

Figure 4. *Add node* refinement

Theorem 5. *(Add new edge between + nodes) If* $n, m \in V_Q^+$*, the set* $Q + \{n \xrightarrow{e^*} m\}$ $(* \in \{+, -\})$*, formed by:*

$$Q_1 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e^*} m^+\},\ \theta_{Q'} \cup (e, \top))$$
$$Q_2 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e^*} m^-\},\ \theta_{Q'} \cup (e, \top))$$
$$Q_3 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e^*} m^+\},\ \theta_{Q'} \cup (e, \top))$$
$$Q_4 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e^*} m^-\},\ \theta_{Q'} \cup (e, \top))$$

(where $Q' = Cl_Q^{\{n,m\}}$*) is a refinement set of* Q *in* G *(Fig. 5).*

Proof.

1. Since Q' is a clone of Q, then $Q \equiv Q'$. In addition, $Q' \subseteq^- Q_1, Q_2, Q_3, Q_4$, thus $Q_1, Q_2, Q_3, Q_4 \preceq Q' \equiv Q$.
2. Let us consider the predicates:

$$P_n = \exists v \in V \left(\bigwedge_{a \in \gamma(n)} Q_a^{\alpha(a)} \ \wedge Q_{e^o}^{\alpha(e)} \right)$$
$$P_m = \exists v \in V \left(\bigwedge_{a \in \gamma(m)} Q_a^{\alpha(a)} \ \wedge Q_{e^i}^{\alpha(e)} \right)$$

If $S \vDash Q_n$ and $S \vDash Q_m$, then we have four mutually complementary options:

- $S \vDash P_n \ \wedge \ S \vDash P_m \Rightarrow S \vDash Q_1$
- $S \vDash P_n \ \wedge \ S \nvDash P_m \Rightarrow S \vDash Q_2$
- $S \nvDash P_n \ \wedge \ S \vDash P_m \Rightarrow S \vDash Q_3$
- $S \nvDash P_n \ \wedge \ S \nvDash P_m \Rightarrow S \vDash Q_4$

□

Next operation adds an additional predicate to an existing edge. This operation is restricted to positive edges connecting positive nodes.

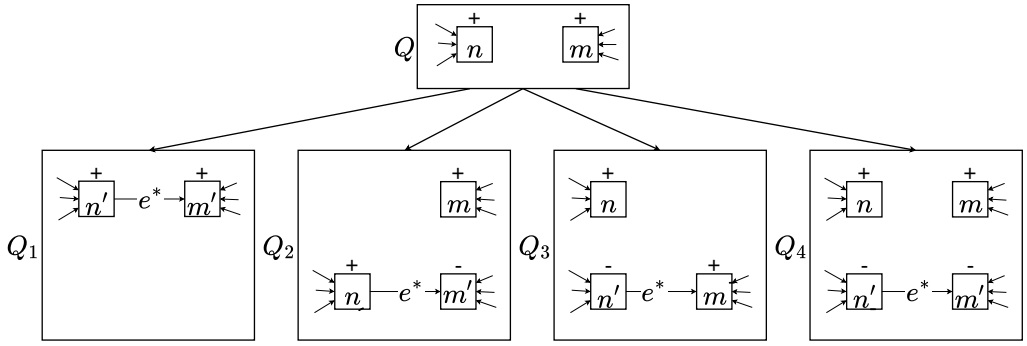

Figure 5. *Add edge* refinement (simplified)

Theorem 6. *(Add predicate to + edge between + nodes) If* $n, m \in V_Q^+$, *with* $n \xrightarrow{e^+} m$, *and* $\varphi \in Form^2(L)$, *the set* $Q + \{n \xrightarrow{e \wedge \varphi} m\}$, *formed by:*

$$Q_1 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e'} m^+\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$
$$Q_2 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e'} m^-\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$
$$Q_3 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e'} m^+\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$
$$Q_4 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e'} m^-\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$

(where $Q' = Cl_Q^{\{n,m\}}$*) is a refinement set of* Q *in* G *(Fig. 6).*

Proof. The proof is similar to previous ones. □

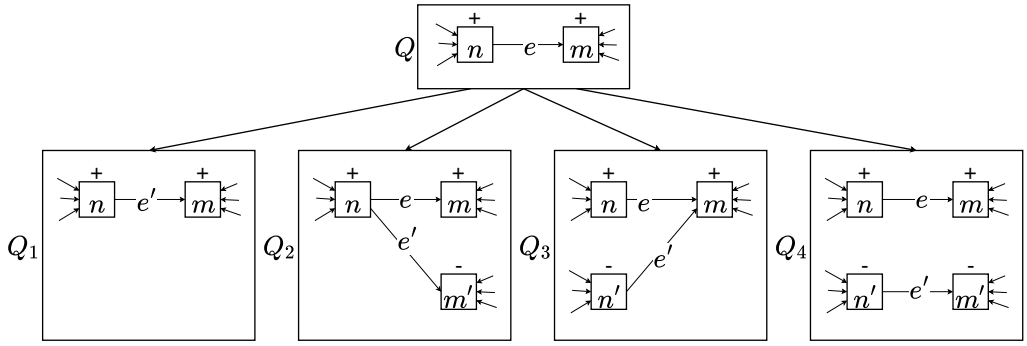

Figure 6. *Add predicate to edge* refinement (simplified)

Finally, the last operation adds predicates to existing nodes. Again, we restrict this operation to cases when the affected nodes are positive (the node where the predicate is added, and those connected to it).

Theorem 7. *(Add predicate to + node with + neighborhood) If* $\varphi \in Form^2(L)$, *and* $n \in V_Q^+$ *with* $\mathcal{N}_Q(n) \subseteq V_Q^+$, *then the set* $Q + \{n \wedge \varphi\}$ *formed by:*

$$\{Q_\sigma = (V_{Q'}, E_{Q'}, \alpha_{Q'} \cup \sigma, \theta_{Q'} \cup (n', \theta_n \wedge \varphi)) \ :\ \sigma \in \{+,-\}^{\mathcal{N}_Q(n)}\}$$

(where $Q' = Cl_Q^{\mathcal{N}_Q(n)}$, and $\{+,-\}^{\mathcal{N}_Q(n)}$ is the set of all possible assignments of signs to elements in $\mathcal{N}_Q(n)$) is a refinement set of Q in G (Fig. 7).

Proof. The proof is similar to previous ones. It is only necessary to take into account that, when modifying the node n, not only the predicate associated with it is modified but also those from all its adjacent nodes, and the set of functions $\{+,-\}^{\mathcal{N}_Q(n)}$ cover all possible sign assignments for the nodes in the neighborhood. □

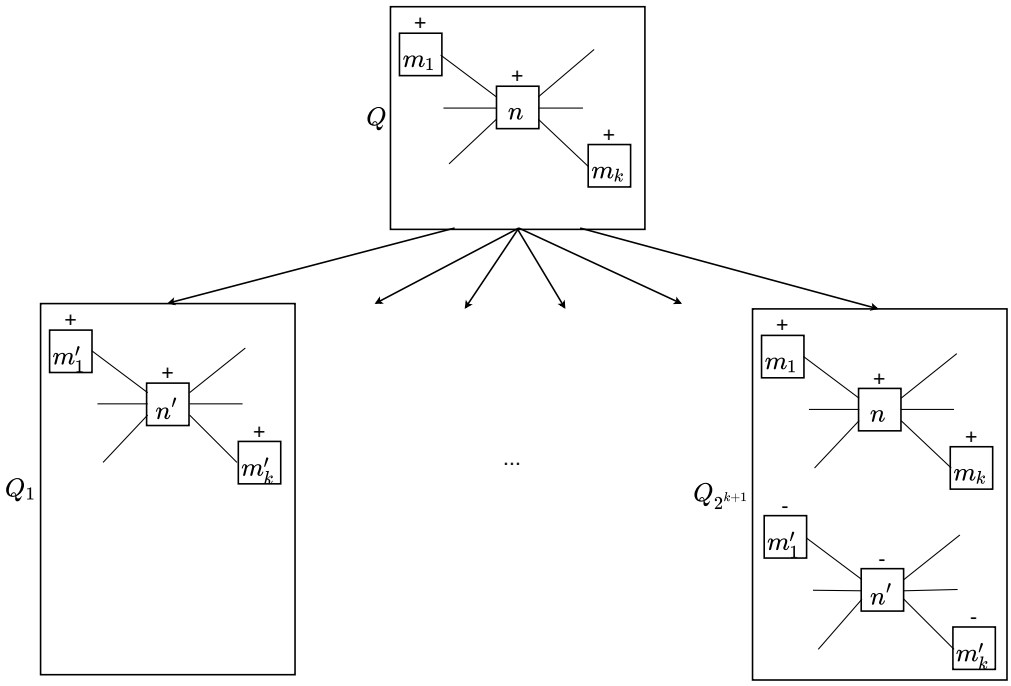

Figure 7. *Add predicate to node* refinement (simplified)

Also note that Figures 4-7 show simplified versions of the refinement sets. Section 3.4 explains how these simplifications can be obtained.

It is not easy to obtain a complementary query from its structure. However, there are many analysis on graphs where sequences of queries verifying some properties about refinement and complementarity are needed. The refinement operations presented in this section come to cover this gap and allow, for example, the construction of an embedded partition tree with the nodes labeled as follows (Fig. 8):

- The root node is labeled with Q_0 (some initial query).
- If a tree node is labeled with Q, and $R = \{Q_1, \ldots, Q_n\}$ is a refinement set of Q, then its child nodes are labeled with the elements of R.

Refinement sets defined here represent only one option, but not the only one. For example, we could consider refinements that, instead of adding constraints to positive elements, lighten the conditions over negative elements, for example by using disjunction of predicates instead of conjunction of them.

3.4. Simplified refinement sets

Let us provide some operations to simplify a query into an equivalent one.

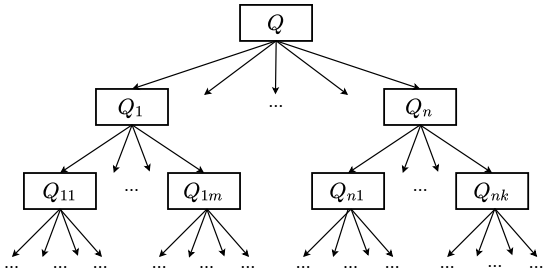

Figure 8. Refinements tree

Definition 3.11. *We say that $Q' \subseteq Q$ is* redundant *in Q if $Q \equiv Q - Q'$. Where $Q - Q'$ is the subgraph of Q given by:*

$$(V_Q \setminus V_{Q'}, E_Q \setminus (E_{Q'} \cup \{\gamma(n) : n \in V_{Q'}\}), \mu_Q)$$

A first result that allows to obtain simplified versions of a query by removing redundant nodes is:

Theorem 8. *Given a query Q, and $n, m \in V_Q$ verifying:*

- $\alpha(n) = \alpha(m)$
- $\theta_n \equiv \theta_m$
- *For each $e \in \gamma(n)$, exists $e' \in \gamma(m)$, with $\alpha(e) = \alpha(e')$, $\theta_e \equiv \theta_{e'}$ and $\gamma(e) \setminus \{n\} = \gamma(e') \setminus \{m\}$*

Then, n is redundant in Q.

Essentially, m is a clone of n, but possibly with more connected edges. We can obtain a similar result for edges:

Theorem 9. *Given a query Q, and two edges, $e, e' \in E_Q$, such that $\alpha(e) = \alpha(e')$, $n \xrightarrow{e} m$ and $n \xrightarrow{e'} m$ with $n, m \in V_Q^+$. If $\theta_e \to \theta_{e'}$ then e' is redundant in Q.*

From these two results we can simplify (by sequentially removing redundant elements after cloning) the refinement sets defined in section 3.3.

3.5. Graph query examples

For illustrative purposes only, in this section we show some queries on a toy graph dataset. Figure 9 shows a section of Starwars graph[2].

In order to simplify the representation of queries and graphs, one of the properties in μ, which we will call τ and represents types on nodes and edges, will be expressed as labels (in edges) or icons (in nodes). Also, *name* property of nodes is written on them, and undirected edges are represented with bidirectional arrows. Property α will be directly represented on query elements using $+/-$ symbols, and we will write the binary predicate θ directly on elements (except in the case of tautologies). When expressions like $\tau(\rho) = X$ are in the predicate of an edge, X is written directly and interpreted as a regular expression that must be verified by the sequence of τ properties of the links in the associated graph path.

Query 1 (Figure 10) can be read as: *Two characters connected with TEACHES relationship where the master is older than 500 years and both are devout of the Jedi*. In this case, structural constraints are imposed through the presence of edges and predicates using τ, *name*, and *age*

[2] http://console.neo4j.org/?id=StarWars

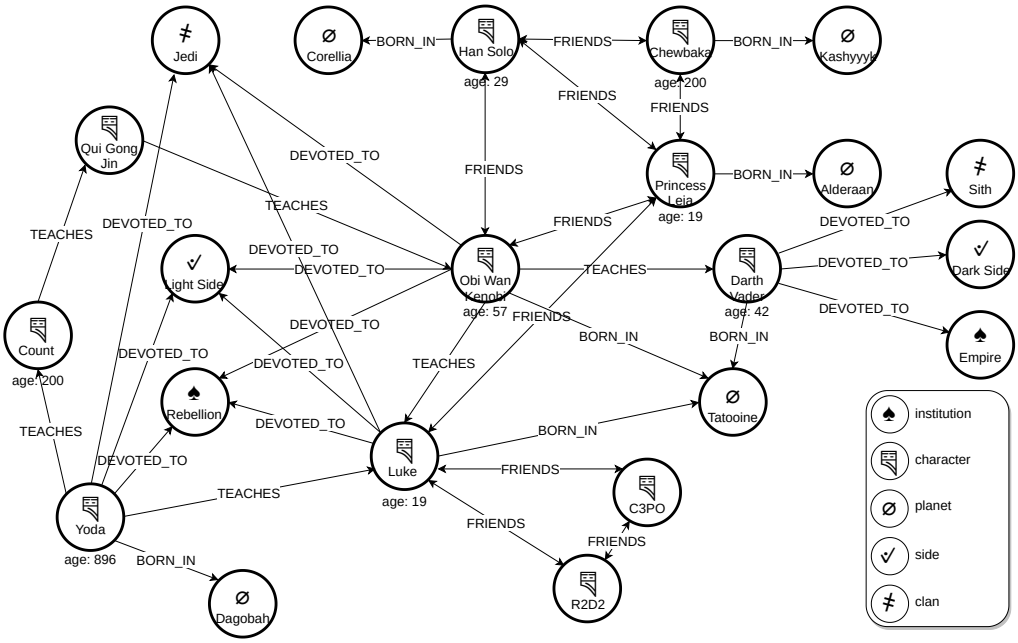

Figure 9. Section of Starwars Graph

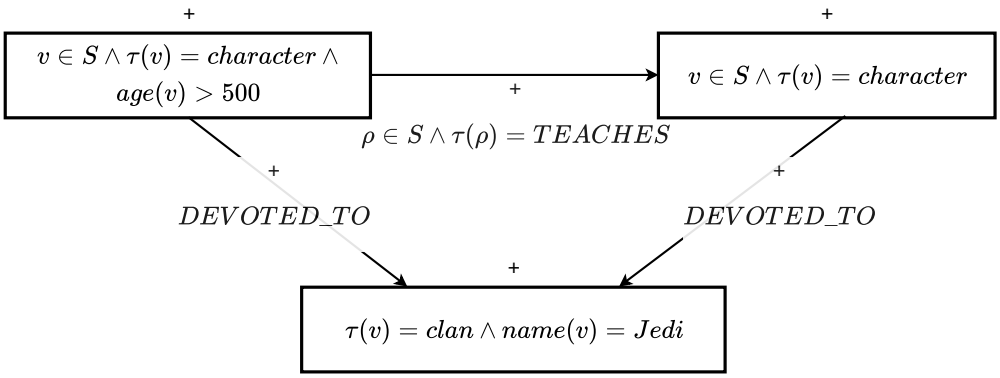

Figure 10. Query 1

properties. As an example, subgraph formed by `Yoda`, `Luke` and the `TEACHES` relationship between them in Figure 9 matches this query.

Query 2 (Figure 11) presents a cyclic query using `FRIENDS` relationships. Any subgraph containing three characters that are friends with each other will verify it (for example, subgraph formed by `Hans Solo`, `Chewbaka`, `Princess Leia` and `FRIENDS` relationships between them in Figure 9).

Query 3 (Figure 12) can be read as: *Character with an outgoing degree greater than 3, devout of a clan other than Sith and connected through a path composed by any number of FRIENDS and TEACHES relationships with someone who comes from Alderaan*. In this case, a regular expression has been used to express a path composed by an undefined number of `FRIENDS` and `TEACHES` relationships, and an auxiliary function, $gr_s(v) \in L$, has been used to refer to the outgoing degree of node v. Any subgraph containing `Luke` or `Obi Wan Kenobi` will verify this query.

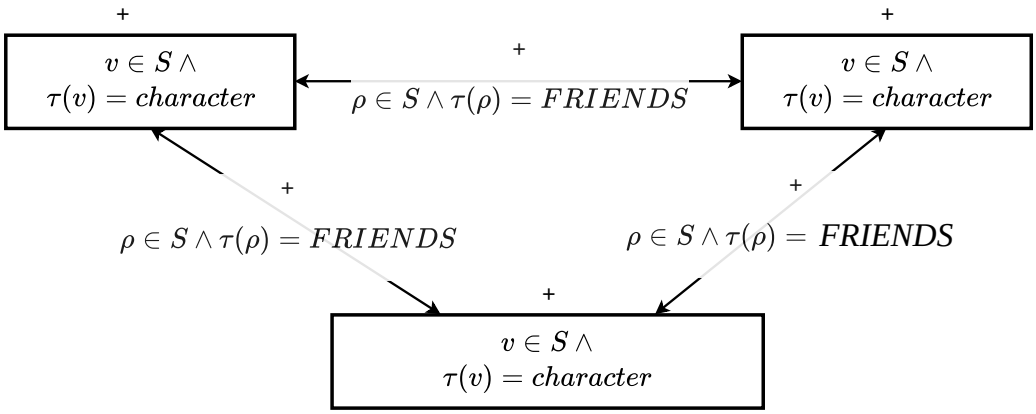

Figure 11. Query 2

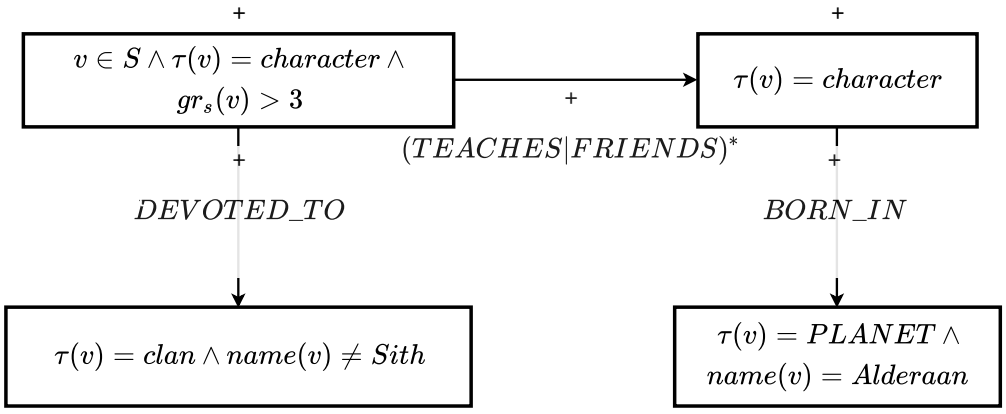

Figure 12. Query 3

3.6. Computational complexity

Query systems based on graph isomorphisms (most of the existing ones) face NP-complete complexity [9,19].

Previous section shows our graph query framework based on logical predicates, and now we will justify that the complexity of evaluating queries is polynomial (even in the case of cyclic queries) when two constraints are imposed: length of paths represented by links in the query are bounded by a constant, k, and the complexity of predicates used in nodes and edges are polynomial.

To check $S \models Q$, we need to check every $Q_n(S)$ predicate associated to each node $n \in V_Q$. Moreover, every predicate $Q_n(S)$ associated to a node in Q requires the evaluation of one $Q_e(v, S)$ predicate for every link $e \in \gamma_Q(n)$. Thus, first we will focus on the analysis of the computational complexity associated to link predicates $Q_e(v, S)$, then we will level up to analyze the complexity of node predicates $Q_n(S)$, and finally we will demonstrate that the complexity of evaluating the query is polynomial.

As set out previous constraints, the complexity of evaluating predicates associated to nodes and edges in a query Q is polynomial, $\mathcal{O}(p)$. Predicates $Q_e(v, S)$ associated to edges in a query verify the existence of a path ρ in the graph G starting/ending in v fulfilling its own predicate $\theta_e(\rho, S)$ and with source and target nodes fulfilling predicates $\theta_{e^o}(\rho^o, S)$

and $\theta_{e^i}(\rho^i, S)$, respectively. Thus, complexity associated to the evaluation of a given path is $\mathcal{O}(3p) = \mathcal{O}(p)$.

Complexity associated to check the existence of a path starting/ending in a node $v \in V$ and fulfilling above conditions is $\mathcal{O}(p \times |V|^k)$, where $|V|^k$ is the number of paths starting/ending in v with length bounded by k. Since the number of links starting/ending in a node $n \in V_Q$ is bounded by $|E_Q|$, complexity associated to a node predicate $Q_n(S)$ belongs to $\mathcal{O}(p \times |V|^k \times |E_Q|)$.

Finally, if the query is composed by $|V_Q|$ nodes, complexity of checking a query $Q(S)$ belongs to $\mathcal{O}(p \times |V|^k \times |E_Q| \times |V_Q|)$. As we can see, constant k (path length bound) plays a main role when executing this kind of queries, due to its condition of exponent of the complexity.

The efficiency of a graph query framework, performing query operations in polynomial time, even when considering cyclic queries, is a fundamental aspect when faced with large-scale data sets (common in real-world applications) and, moreover, when using a query system as the kernel of relational machine learning algorithms, as we show in the next section.

4. Relational machine learning

In this section, we will use the benefits of the presented framework to learn relational classifiers on graph data sets. Specifically, starting from a set of labeled subgraphs in a graph data set, we will define a pattern search method based on information gain to obtain characteristic patterns for each class of subgraphs.

4.1. Information-gain pattern mining

In order to use previous graph query framework to obtain characteristic patterns of subgraph classes, we will perform top-down induction of decision trees to explore the pattern space, using graph queries as test tools in internal nodes of the trees. Along the tree construction process, the best refinement sets are obtained, providing queries that characterize classes in the graph data set.

The training set, $\mathcal{L}$, consists of pairs (S_i, y_i), where S_i is a subgraph of G, and y_i its associated class. Each node n of the resulting decision tree is associated with:

- a subset of the training set: $\mathcal{L}_n \subseteq \mathcal{L}$,
- a query Q_n such that: $\forall S \in \mathcal{L}_n (S \vDash Q_n)$.

The algorithm follows the usual tree learning procedure: It starts creating a tree containing a single node (the root) associated with the complete training set, $\mathcal{L}$, and with an initial query Q_0 matching all its elements ($\forall S \in \mathcal{L}, S \vDash Q_0$). Then, the algorithm evaluates which refinement set maximizes information gain when dividing $\mathcal{L}$ and applies it to Q_0. Next, one child node is created for every query in the refinement set and samples from $\mathcal{L}$ are transmitted through the child with the matching query (as it is a refinement set of Q_0, there is one and only one child with a matching associated query). The process is repeated recursively for every new node until some stop condition is satisfied, then the node will become a leaf associated to a class. Note that decision trees obtained with this method are not necessarily binary, unlike the most common in literature.

4.2. Relational tree learning examples

Here we present some application examples to illustrate how relational learning can be performed using this query framework and refinement sets. The refinement operations will be those seen in section 3.3. Our stop condition requires that all subgraphs in a decision node belong to the same class. First we will address node classification problems and then we will move to classify more complex structures.

Consider the toy graph shown in Figure 13, which represents a small social network with users and items. Our goal is to learn to classify nodes according to their types from the patterns observed in the data set.

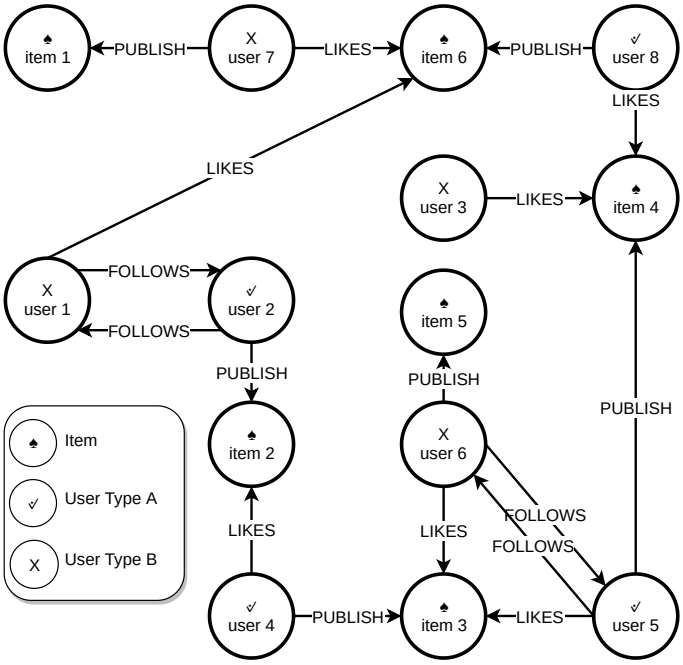

Figure 13. Social Network toy

Starting with a training set formed by all nodes in the graph, Figure 14 shows the relational decision tree obtained through the process described in section 4.1. Negative nodes/edges are marked with a cross and nodes with predicate $\theta(v, S) := v \in S$ are bigger and white colored. This tree correctly classifies all nodes in the graph according to their type (`User A`, `User B`, or `Item`) using relational information from the network. In addition, characteristic patterns for every node type are obtained in the leafs of the tree that can be used to directly evaluate nodes and explain future classifications.

In a similar way, by using every character node in Starwars toy graph (Figure 9) and its corresponding `specie` property as training set, relational decision tree in Figure 15 discriminates and explain every character specie in the graph. Patterns on the leafs of the tree characterize each specie: born friends of `Luke` are humans but unborn friends of `Luke` are droids, wookies are those born in `Kashyyk`, etc.

5. Conclusions and future work

The main contribution of this paper consists in a novel graph query framework that enables polynomial cyclic evaluation of queries and refinements between queries based on atomic operations. As it was shown, refinements can be used in relational learning processes. Moreover, the presented framework accomplishes several pretty requirements: (1) uses the very same grammar for the queries and for the structures under evaluation, (2) allows evaluation of structures (subgraphs) beyond single nodes, (3) allows cyclic queries in polynomial time (when query path length is bounded), (4) provides a controlled and automatic query construction by using refinements, and (5) refinement sets form embedded partitions of the set of structures they evaluate, providing ideal tools for top-down learning procedures.

Query systems based on graph isomorphism present exponential complexity when facing cyclic queries. In addition, if a projection is required for pattern verification, the task of evaluating the non-existence of certain elements becomes hard or impossible. Specifically, the query graph framework presented here evaluates the existence/non-existence of paths and nodes in a graph instead of requiring isomorphisms, allowing the evaluation of cyclic patterns in polynomial time.

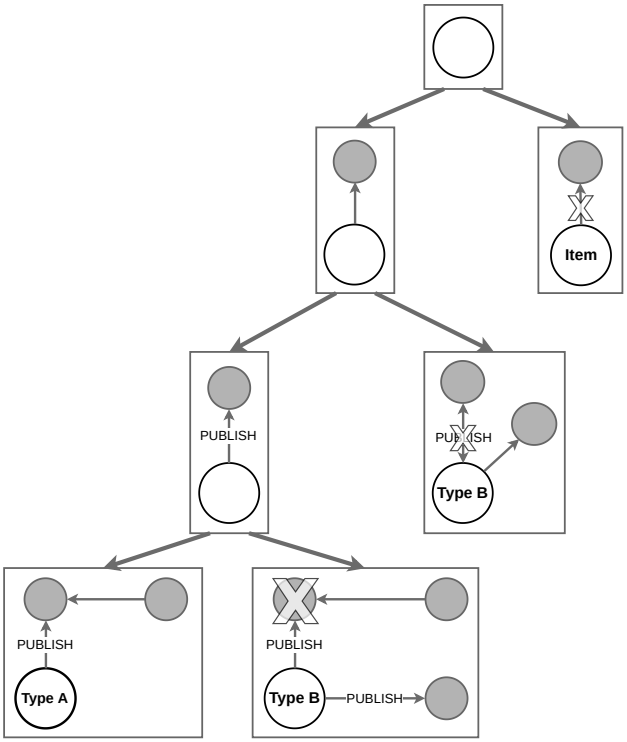

Figure 14. Node type classifier

After carrying out a first and fully functional proof-of-concept implementation, the capabilities of this graph query framework have been experimentally demonstrated. An explicit use in relational learning procedures, as those shown in section 4.2, has been carried out, and the experimental results show that this methodology extracts interesting patterns from relational data, something of major importance in explainable learning and automatic feature extraction tasks.

Despite presented query definition makes use of binary graph data sets (not hypergraphs), they can be applied on hypergraph data too, since the concept of path that connects pairs of nodes is defined independently of the arity of the edges involved. For the sake of simplicity, and because of the lack of real hypergraph databases, we have restricted queries to the binary case, but they are ready to be extended to more general cases when the use of hypergraphs will be widespread.

Also, a minimal and robust set of refinement operations have been offered in section 3.3, but they are not intended to be optimal for every learning task. More complex refinement families can be created (for example, combining the refinement *add edge* with *adding property to an edge* into a single step) to reduce the number of steps needed to reach complex queries and to avoid plateaus (flat regions in the pattern space). If this option is carried out properly (unifying the refinements according to the frequency of occurrence of structures in a graph, for example) faster versions of learning algorithms can be obtained by sacrificing the covering of a wider query space. Theoretical tools have been provided in this work to support the correctness of new refinement families, and it is a future work to develop automatic methods for generating refinement sets according to a given learning task and the specific characteristics on the graph data set. Automatic generation of such sets from statistics extracted from the graph data can lead to considerable optimizations.

Patterns associated to leafs in the obtained decision trees can be used to characterize subgraph classes. In addition, as usual in decision trees, the path from the root node to the corresponding leaf of the decision tree for a given input can be used to justify decisions, something useful in many sensitive applications. Moreover, patterns obtained from the presented graph learning procedure can be used as features in other machine learning

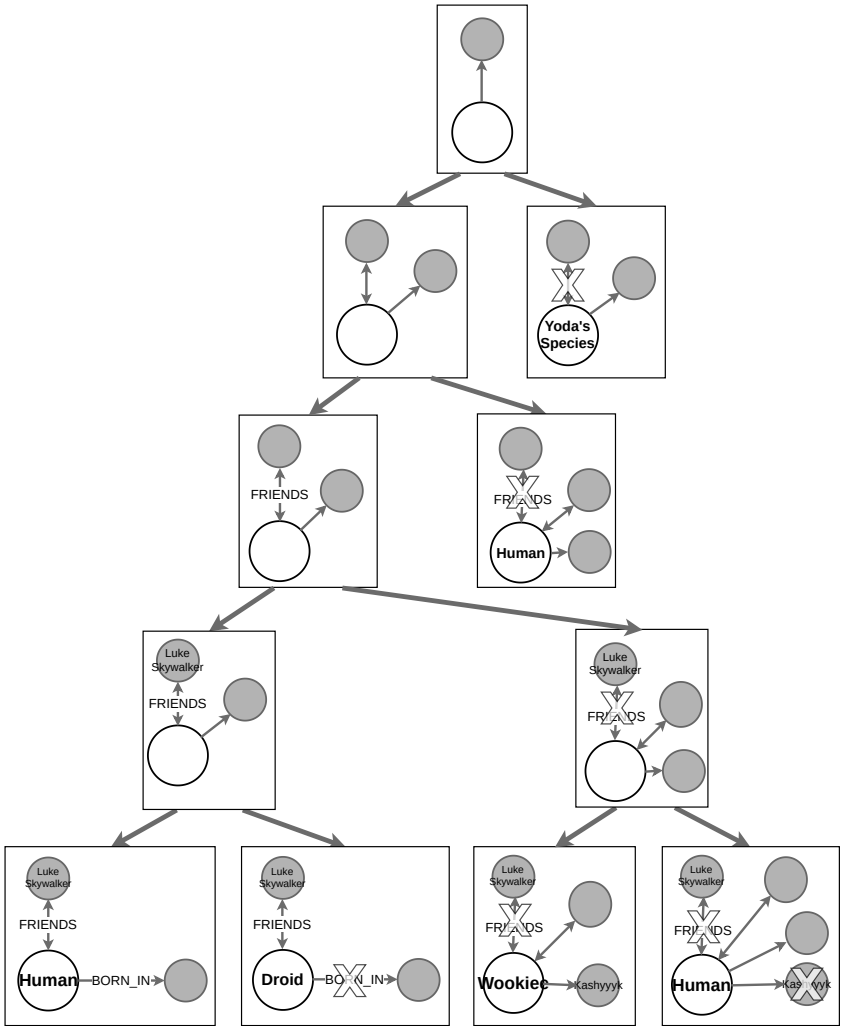

Figure 15. Character specie classifier

methods. Once the patterns have been obtained, they can be used as boolean features to model subgraphs and allow non-relational machine learning methods learn from them.

Relational decision tree learning can be exploited by *ensemble* methods (like Random Forest) and, although explainability is diluted when several trees are combined, their predictive capacity can be greatly expanded. Probabilistic aggregation of queries to produce patterns that can be interpreted as probabilistic decision tools must be explored.

In addition, although a relational decision tree learning method has been used, other machine learning algorithms could be tested in conjunction with this query framework to explore further possibilities of relational learning.

Acknowledgements

Acknowledgments: The work was supported in part by the GLVEZUS Project of Universidad Central de Ecuador. This research is also partially supported by the project TIN2013-41086-P (LOCOCIDA Project), from Ministerio de Economía, Industria y Competitivad of Spain with FEDER funds from EU, and by the project "Metodologías de datos aplicadas al análisis de las exposiciones artísticas en Andalucía para el desarrollo de la economía creativa", from Fundación Pública Centro de Estudios Andaluces (2017-2019).

References

1. Almagro-Blanco, P., & Sancho-Caparrini, F. (2017). Generalized graph pattern matching. *CoRR*, *abs/1708.03734*. URL: http://arxiv.org/abs/1708.03734. arXiv:1708.03734.
2. Barceló, P., Libkin, L., & Reutter, J. L. (2011). Querying graph patterns. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* PODS '11 (pp. 199–210). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/1989284.1989307. doi:10.1145/1989284.1989307.
3. Blockeel, H., & Raedt, L. D. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, *101*, 285–297. URL: http://www.sciencedirect.com/science/article/pii/S0004370298000344. doi:10.1016/S0004-3702(98)00034-4.
4. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26* (pp. 2787–2795). Curran Associates, Inc. URL: http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf.
5. Brynjolfsson, E., & Mitchell, T. (2017). What can machine learning do? workforce implications. *Science*, *358*, 1530–1534.
6. Camacho, R., Pereira, M., Costa, V. S., Fonseca, N. A., Adriano, C., Simões, C. J., & Brito, R. M. (2011). A relational learning approach to structure-activity relationships in drug design toxicity studies. *Journal of integrative bioinformatics*, *8*, 176–194.
7. Chang, K.-W., Yih, S. W.-t., Yang, B., & Meek, C. (2014). Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. ACL – Association for Computational Linguistics. URL: https://www.microsoft.com/en-us/research/publication/typed-tensor-decomposition-of-knowledge-bases-for-relation-extraction/.
8. Consens, M. P., & Mendelzon, A. O. (1990). Graphlog: A visual formalism for real life recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* PODS '90 (pp. 404–416). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/298514.298591. doi:10.1145/298514.298591.
9. Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* STOC '71 (pp. 151–158). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/800157.805047. doi:10.1145/800157.805047.
10. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., & Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '14 (pp. 601–610). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/2623330.2623623. doi:10.1145/2623330.2623623.
11. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., & Wu, Y. (2010). Graph pattern matching: From intractable to polynomial time. *Proc. VLDB Endow.*, *3*, 264–275. URL: http://dx.doi.org/10.14778/1920841.1920878. doi:10.14778/1920841.1920878.
12. Gallagher, B. (2006). Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, *6*, 45–53.
13. García-Jiménez, B., Pons, T., Sanchis, A., & Valencia, A. (2014). Predicting protein relationships to human pathways through a relational learning approach based on simple sequence features. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *11*, 753–765. doi:10.1109/TCBB.2014.2318730.
14. Geamsakul, W., Matsuda, T., Yoshida, T., Motoda, H., & Washio, T. (2003). Classifier construction by graph-based induction for graph-structured data. In K.-Y. Whang, J. Jeon, K. Shim, & J. Srivastava (Eds.), *Advances in Knowledge Discovery and Data Mining: 7th Pacific-Asia Conference, PAKDD 2003, Seoul, Korea, April 30 – May 2, 2003 Proceedings* (pp. 52–62). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/3-540-36175-8_6. doi:10.1007/3-540-36175-8_6.
15. Gupta, S. (2015). *Neo4j Essentials*. Community experience distilled. Packt Publishing. URL: https://books.google.es/books?id=WJ7NBgAAQBAJ.
16. Henzinger, M. R., Henzinger, T. A., & Kopke, P. W. (1995). Computing simulations on finite and infinite graphs. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on* (pp. 453–462). IEEE.
17. Jacob, Y., Denoyer, L., & Gallinari, P. (2014). Learning latent representations of nodes for classifying in heterogeneous social networks. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining* WSDM '14 (pp. 373–382). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/2556195.2556225. doi:10.1145/2556195.2556225.
18. Jiang, J. Q. (2011). Learning protein functions from bi-relational graph of proteins and function annotations. In T. M. Przytycka, & M.-F. Sagot (Eds.), *Algorithms in Bioinformatics* (pp. 128–138). Berlin, Heidelberg: Springer Berlin Heidelberg.
19. Karp, R. M. (1975). On the computational complexity of combinatorial problems. *Networks*, *5*, 45–68.
20. Knobbe, A. J., Siebes, A., Wallen, D. V. D., & Syllogic B., V. (1999). Multi-relational decision tree induction. In *In Proceedings of PKDD' 99, Prague, Czech Republic, Septembre* (pp. 378–383). Springer.
21. Latouche, P., & Rossi, F. (2015). Graphs in machine learning: an introduction.
22. Leiva, H. A., Gadia, S., & Dobbs, D. (2002). Mrdtl: A multi-relational decision tree learning algorithm. In *Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003* (pp. 38–56). Springer-Verlag.
23. Milner, R. (1989). *Communication and Concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.
24. Nguyen, P. C., Ohara, K., Motoda, H., & Washio, T. (2005). Cl-gbi: A novel approach for extracting typical patterns from graph-structured data. In T. B. Ho, D. Cheung, & H. Liu (Eds.), *Advances in Knowledge Discovery and Data Mining: 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005. Proceedings* (pp. 639–649). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/11430919_74. doi:10.1007/11430919_74.

25. Nickel, M., Murphy, K., Tresp, V., & Gabrilovich, E. (2016). A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, *104*, 11–33.
26. Plotkin, G. (1972). Automatic methods of inductive inference, .
27. van Rest, O., Hong, S., Kim, J., Meng, X., & Chafi, H. (2016). Pgql: a property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems* (p. 7). ACM.
28. Reutter, J. L. (2013). *Graph Patterns: Structure, Query Answering and Applications in Schema Mappings and Formal Language Theory*. Ph.D. thesis The school where the thesis was written Laboratory for Foundations of Computer Science School of Informatics University of Edinburgh.
29. Segaran, T., Evans, C., Taylor, J., Toby, S., Colin, E., & Jamie, T. (2009). *Programming the Semantic Web*. (1st ed.). O'Reilly Media, Inc.
30. Tang, L., & Liu, H. (2009). Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 817–826). ACM.
31. Zou, L., Chen, L., & Özsu, M. T. (2009). Distance-join: Pattern match query in a large graph database. *Proc. VLDB Endow.*, *2*, 886–897. URL: http://dx.doi.org/10.14778/1687627.1687727. doi:10.14778/1687627.1687727.