mathematics

MDPI

Article

Logical–Mathematical Foundations of a Graph Query Framework for Relational Learning

Pedro Almagro-Blanco, Fernando Sancho-Caparrini and Joaquín Borrego-Díaz *

Departamento Ciencias de la Computación e Inteligencia Artificial, E. T. S. Ingeniería Informática, Universidad de Sevilla, 41012 Sevilla, Spain; palmagro@us.es (P.A.-B.)
* Correspondence: jborrego@us.es

Abstract: Relational learning has attracted much attention from the machine learning community in recent years, and many real-world applications have been successfully formulated as relational learning problems. In recent years, several relational learning algorithms have been introduced that follow a pattern-based approach. However, this type of learning model suffers from two fundamental problems: the computational complexity arising from relational queries and the lack of a robust and general framework to serve as the basis for relational learning methods. In this paper, we propose an efficient graph query framework that allows for cyclic queries in polynomial time and is ready to be used in pattern-based learning methods. This solution uses logical predicates instead of graph isomorphisms for query evaluation, reducing complexity and allowing for query refinement through atomic operations. The main differences between our method and other previous pattern-based graph query approaches are the ability to evaluate arbitrary subgraphs instead of nodes or complete graphs, the fact that it is based on mathematical formalization that allows the study of refinements and their complementarity, and the ability to detect cyclic patterns in polynomial time. Application examples show that the proposed framework allows learning relational classifiers to be efficient in generating data with high expressiveness capacities. Specifically, relational decision trees are learned from sets of tagged subnetworks that provide both classifiers and characteristic patterns for the identified classes.

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

MSC: 03B70; 68P15; 68P20

Citation: Almagro-Blanco, P.; Sancho-Caparrini, F.; Borrego-Díaz, J. Logical–Mathematical Foundations of a Graph Query Framework for Relational Learning. *Mathematics* **2023**, *11*, 4672. https://doi.org/10.3390/math11224672

Academic Editors: Jie Meng, Xiaowei Huang, Minghui Qian and Zhixuan Xu

Received: 18 September 2023
Revised: 8 November 2023
Accepted: 11 November 2023
Published: 16 November 2023

1. Introduction

Typically, machine learning algorithms take as input a set of objects, each described by a vector of numerical or categorical attributes, and produce (learn) a mapping from the input to the output predictions: a class label, a regression score, 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 are represented as graphs composed of nodes (entities) and links (relationships), both with possible associated properties.

The fact that relational learning methods can learn from the connections between data makes them very powerful in different domains [1–4]. Learning to classify profiles in social networks based on their relationships with other objects [5,6], characterizing proteins based on functional connections that arise in organisms [7], and identifying molecules or molecular fragments with the potential to produce toxic effects [8] are some prominent examples of relational machine learning applications.

There are two basic approaches to relational learning, the *latent feature* or *connectionist approach* and the *graph pattern-based approach* or *symbolic approach* [9]. The connectionist approach has proven its effectiveness in many different tasks [10–15]. In comparison,

the pattern-based approach has been less successful. Two of the most important reasons for this fact are the computational complexities arising from relational queries and the lack of robust and general frameworks that serve as the basis for this kind of symbolic relational learning method. On the one hand, most existing relational query systems are based on graph isomorphisms, and their computational complexity is NP-complete, which affects the efficiency of learning methods using them [16]. On the other hand, most existing query systems do not allow for atomic operations to expand queries in a partitioned manner, preventing learning systems from efficiently searching the query space [17].

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

The computational capacity needed to assess the performance of graph query methods is significant. Our study focuses on formalizing an efficient graph query system and defining a set of operations to refine queries. However, we do not conduct an extensive analysis of performance or efficiency in comparison to other methods. The primary result of our study is a mathematical formalization of a graph query system. The graph query system must fulfill three characteristics: (1) Conduct atomic operations (refinements) to expand queries in a partitioned manner, (2) assess any substructure in a graph (beyond isolated nodes or complete graphs), and (3) evaluate cyclic patterns in polynomial time. To the best of our knowledge, no other approach meets these requirements.

The paper is structured as follows. Section 2 provides an overview of related research. Section 3 introduces a novel graph query framework, outlining its main definitions and properties that guarantee its utility. Representative query examples and an analysis of the computational complexity arising from the model are also presented. Section 4 describes the implementation of the framework in performing relational machine learning. Finally, Section 5 presents the conclusions that can be drawn from this investigation and identifies potential avenues for future research.

2. Related Work

A common approach to executing relational queries entails developing patterns in an abstract representation of data and searching for their occurrences in actual datasets. This working method falls under the scope of graph pattern matching, an area of study that has been actively researched for more than three decades. Depending on various aspects to consider, there are customary distinctions in pattern-matching methods. (a) Structural, semantic, exact, inexact, optimal, and approximate are distinctions that can be made in matching relations between patterns and subgraphs [18]. Additionally, graph pattern matching can be based on isomorphisms, graph simulation, and bounded simulation, among other methods [19–21]. While systems for querying based on graph isomorphism present NP complexity, those based on simulations present polynomial complexity [22,23]. 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 the non-existence of elements. Our proposal is within the scope of semantic, exact, and optimal graph pattern matching implemented with an approach similar to simulations.

As stated above, there are two fundamentally different types of relational learning models [24]. The first type, known as 'the latent feature approach', is founded upon latent feature learning, for example, tensor factorization and neural models, and generally performs well when handling uncertainty via probabilistic approximation [3,25,26]. The second approach, known as the graph-pattern-based approach, automatically extracts relational patterns, also called observable graph patterns, from data [27,28]. Since this work pertains to the second approach, our focus in the subsequent discussion will be on the

review of relational learning techniques that utilize the graph-pattern-based approach and the query systems upon which they rely.

Most of the pattern-based relational learning methods are derived from Inductive Logic Programming (ILP) [29]. ILP does not inherently offer relational classifiers, though it does permit the automatic creation of logical decision trees capable of managing relational predicates, provided that data relationships have been properly transformed into logical predicates. Binary decision trees are logical decision trees, in which all tests in internal nodes are expressed as conjunctions of literals of a prefixed first-order language. TILDE (Top-down Induction of Logical Decision Trees) is one of the representative algorithms that can learn this type of decision tree from a given set of examples [30]. TILDE provides a framework for generating logical decision trees that can be further adapted for relational decision trees. Nevertheless, it does not cater to relational learning and, therefore, fails to offer certain operations for refining relational queries. We refer to atomic operations as those that bring about minor structural modifications to the query (typically the addition or deletion of a node or an edge, or some of their characteristics).

Multi-relational decision tree learning (MRDTL [28]) is a learning algorithm for relationships and is supported by selection graphs [17], a graph representation of SQL queries that selects records from a relational database based on certain constraints. Selection graphs enable atomic operations to enhance queries, but they lack the ability to distinguish between query elements that constitute the query result and those that relate to objects that should or should not be linked to the query result. Consequently, queries performed using selection graphs yield records that satisfy the given selection graph conditions but cannot identify subgraphs. The refinement operations presented on the selection graphs are as follows: *adding positive conditions*, *adding negative conditions*, *adding present edges and opening nodes*, *and adding absent edges* and *closing nodes*. This set of operations does not allow for the construction of cyclic patterns.

Another noteworthy pattern-based method for relational learning is graph-based induction of decision trees (DT-GBI [31]), which is a decision tree construction algorithm for learning graph classifiers using graph-based induction (GBI), a data mining technique for extracting network motifs from labeled graphs by connecting pairs of nodes. In DT-GBI, the attributes (referred to as patterns or substructures) are generated during the execution of the algorithm [27].

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 learning from general subgraphs as base cases. Moreover, our technique can execute cyclic queries, hence allowing for the extraction of cyclic patterns from data during the learning process.

3. Graph Query Framework

In graph pattern matching, precise definitions are fundamental for research. They create a shared terminology, whilst theorems illustrated by mathematical proofs reveal essential characteristics and direct the development of algorithms. This paper presents a mathematical tool for conducting graph pattern matching. We will do this by utilizing these mathematical tools.

We are exploring graph queries that enable atomic specializations. We aim to produce pattern specializations that select only a particular subset of elements by utilizing a set of elements that satisfy a specific relational pattern. Selection Graphs serves as an instance of this query tool and has been developed for use in relational learning procedures. However, it has a fundamental limitation, as its patterns are unable to include cycles. Furthermore, when dealing with high relational data, incorporating a graphical representation of SQL queries can lead to efficiency issues. Our proposal draws inspiration from this method while bypassing its possible constraints.

When searching for query specialization, we aim to also create complementary queries to cover new conditions. Exploring the pattern space and characterizing elements in a

top-down method would be helpful. In particular, we seek a group of specialized queries that create embedded partitions of a single query.

We would like to emphasize that our primary aim is to offer formalization and examples of how the model can be applied, with the added goal of producing a real implementation that is practical for use (https://github.com/palmagro/ggq, accessed on 1 September 2023).

3.1. Preliminaries

This passage presents preliminary concepts for defining graph queries. For a more complete review, refer to [32].

We will begin with a graph definition that encompasses several common types found in the literature, such as directed/undirected graphs, multi-relational graphs, and hypergraphs. This definition serves as a foundational basis for general graph dataset structures and queries.

Definition 1. *A* Graph *is a tuple* $G = (V, E, \mu)$*, where*

- V *and* E *are sets, called, respectively, the* 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 *represents the set of* values*.*

Furthermore, it is necessary to have a distinct key for the edges of the graph, called incidences *and denoted by* γ*, which associates each edge in* E *with a set of vertices in* V*.*

The domain of μ is the Cartesian product of the sets $V \cup E$ and R. Generally, we denote $\alpha(x)$ instead of $\mu(x, \alpha)$ for each $x \in V \cup E$ and $\alpha \in R$, treating properties as maps from nodes/edges to values. Unlike standard definitions, the items in E are symbols that indicate the edges, rather than pairs of elements from V. Additionally, gamma is the function that matches each edge to the group of nodes—ordered or otherwise—that it connects.

We will use $\gamma(v)$ to denote the edges in which node $v \in V$ participates. The *neighborhood* of v is the set of nodes, including itself, connected to it; that is, $\mathcal{N}(v) = \bigcup_{e \in \gamma(v)} \gamma(e)$.

For instance, we could depict a binary social graph G, which encompasses a set of nodes V, a set of edges E, and a function μ that associates each node/edge in the graph with a set of properties R. Our social graph would comprise the attribute $\tau \in R$ that may assume the values *person* and *photo* for the nodes and $follows$ and *like* for edges. The attribute $gamma \in R$ would be responsible for associating a pair of nodes in V with each edge. Furthermore, nodes and edges may possess additional attributes, such as *age* for nodes with $\tau = person$ or *date* for edges with $\tau = like$.

We need to provide an understanding of the position of a node in an edge. We offer a basic definition of *position*, but a more comprehensive one can be provided to distinguish between *directed* and *undirected* edges:

Definition 2. *If* $e \in E$ *and* $\gamma(e) = (v_1, \ldots, v_n) \in V^n$*, then we define the* position *of each* $v_i \in \gamma(e)$ *in* e *as* $ord_e(v_i) = i$*. We denote* $u \leq_e v$ *to indicate* $ord_e(u) \leq ord_e(v)$*.*

From this ordering of the nodes on an edge, we can establish *paths* within a graph.

Definition 3. *Given a graph* $G = (V, E, \mu)$*, we define the set of paths in* G *as* $\mathcal{P}_G$*, which is the smallest set that satisfies the conditions:*

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 as follows:

- 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:

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

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

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

For example, for a graph G with $V = \{v_1, v_2, v_3\}$ and $E = \{e_1, e_2\}$, $\gamma(e_1) = \{v_1, v_2\}$ and $\gamma(e_2) = \{v_2, v_3\}$, the set of paths comprises $\mathcal{P}_G = \{v_1 \overset{e_1}{\rightsquigarrow} v_2, v_2 \overset{e2}{\rightsquigarrow} v_3, v_1 \overset{e1 \cdot e2}{\rightsquigarrow} v_3\}$ with $\mathcal{P}^o_{v_1}(G) = \{v_1 \overset{e1}{\rightsquigarrow} v_2, v_1 \overset{e1 \cdot e_2}{\rightsquigarrow} v_3\}$, and $\mathcal{P}^i_{v_1}(G) = \emptyset$. The concept of a subgraph is acquired by employing the customary procedure of enforcing that the features are sustained within the intersecting elements.

Definition 4. *A subgraph of* $G = (V, E, \mu)$ *is defined as a graph* $S = (V_S, E_S, \mu_S)$ *where* V_S *is a subset of* V, E_S *is a subset of* E, *and* μ_S *is a subset of* $\mu_{|V_S \cup E_S}$. *We denote* $S \subseteq G$.

An instance of a subgraph from the graph stated earlier could be constituted by $V_S = \{v_1, v_2\}$, $E_S = \{e_1\}$, and $\gamma | V_S \cup E_S$.

3.2. Graph Queries

As mentioned previously, our graph query framework aims to enable the generation of complementary queries based on a given query. This entails ensuring that if a subgraph does not comply with a query, it must always comply with one of its complements. However, since projection hinders the evaluation of non-existent elements, which is necessary for achieving complementarity, we propose the use of logical predicates instead of projections.

In the following, we examine a graph that is prefixed, denoted by $G = (V, E, \mu)$. We will provide a brief formalization of our understanding of a predicate for G. More details on this topic can be found in [33].

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

Definition 5. *A* query *for* G *is a graph, specifically* $Q = (V_Q, E_Q, \mu_Q)$, *possessing* α *and* θ *properties in* μ_Q, *and satisfying the following conditions:*

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

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

Intuitively, when examining a query, we utilize the second input of binary predicates to place limitations on the membership of subgraphs within G. Conversely, the first input should receive elements of the corresponding type with which it is associated.

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

$$\theta_a(v, S) := v \in S,$$

$$\theta_b(v, S) := \exists z \in S \ (z \rightsquigarrow v),$$

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

The node-based $\theta_a(v, S)$ is defined to check whether the subgraph evaluation of S contains $v \in V$. The node-based $\theta_b(v, S)$ is verified only when a path in G connects a node of S with $v \in G$. Lastly, the path-based $\theta_e(\rho, S)$ is defined to verify if the evaluated path $\rho \in \mathcal{P}_G$ connects S with its outward in G.

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

According to the following definition, positive elements impose constraints on the presence of queries, while negative elements impose constraints on their absence. To be more specific, each positive/negative node in a query requires the existence/non-existence of a node in G that satisfies its conditions (imposed by θ_x and its edges):

Definition 6. *Given $S \subseteq G$, and $Q \in \mathcal{Q}$, we say that S* matches *Q ($S \vDash Q$), if the following formula holds:*

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

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.

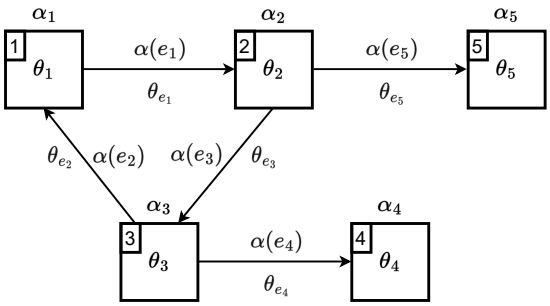

Figure 1. Graph query example.

Unlike other previous graph query systems, this system can efficiently satisfy the following requirements: (1) The ability to contain cycles; (2) the capability to evaluate subgraphs; (3) projecting edges in the query onto paths in the graph; (4) evaluating structural and/or semantic characteristics; and (5) the added benefit of specialization through atomic operations (as will be discussed in the next section).

3.3. Refinement Sets

To properly characterize the elements within a graph, it is crucial to utilize computationally effective methods when constructing queries based on basic operations. This

section will introduce a query construction method optimized for use in relational learning tasks. To begin, let us first define the concept of relative refinements between queries.

Definition 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 deemed equivalent when they are confirmed to be exactly the same by identical subgraphs. From this definition, it is straightforward the following result is straightforward (the proof of which may be omitted):

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 examine the relationship between the topological structure of a query and its functionality as a predicate on subgraphs. Generally, extracting the logical properties of the predicate from the structural properties of the graph that represents it is difficult. However, we can obtain useful conditions to manipulate the structures and modify the query's semantics in a controlled manner.

Definition 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 illustrates an example of a Q^--conservative extension. The novel element in the right query mandates fresh constraints on the positive node, but it does not introduce any additional constraints to the negative one.

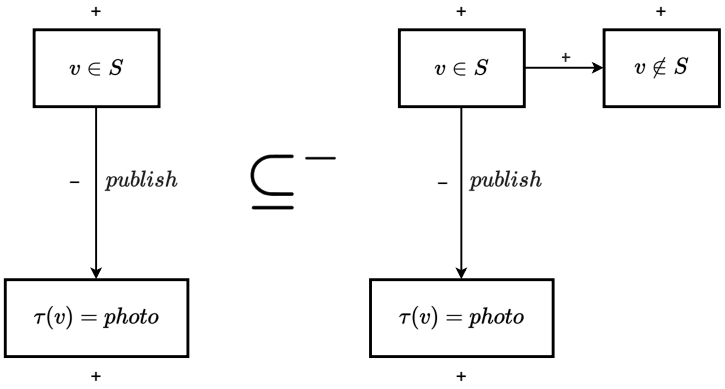

Figure 2. Q^--conservative extension.

Since negative nodes introduce non-existence constraints to subgraph verification, Q^--conservative extensions guarantee that no new constraints are added to them. Therefore,

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

Proof. Since predicates associated with edges are solely based on the information within the edge itself (which takes into account the value of θ in its incident nodes, irrespective of their α value), we can assert that:

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

Considering this fact, we examine the behavior of predicates associated with the nodes for both queries:

- 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 results suggest 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 (which, consequently, will not provide refined predicates).

To achieve controlled methods of query generation, we will outline processes for refining queries through unit steps. We will accomplish this by defining the cloning operation, whereby existing nodes are duplicated, and all incident edges (including those between the nodes) on the original graph are also cloned:

Definition 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 from W, 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 node set to be duplicated is highlighted).

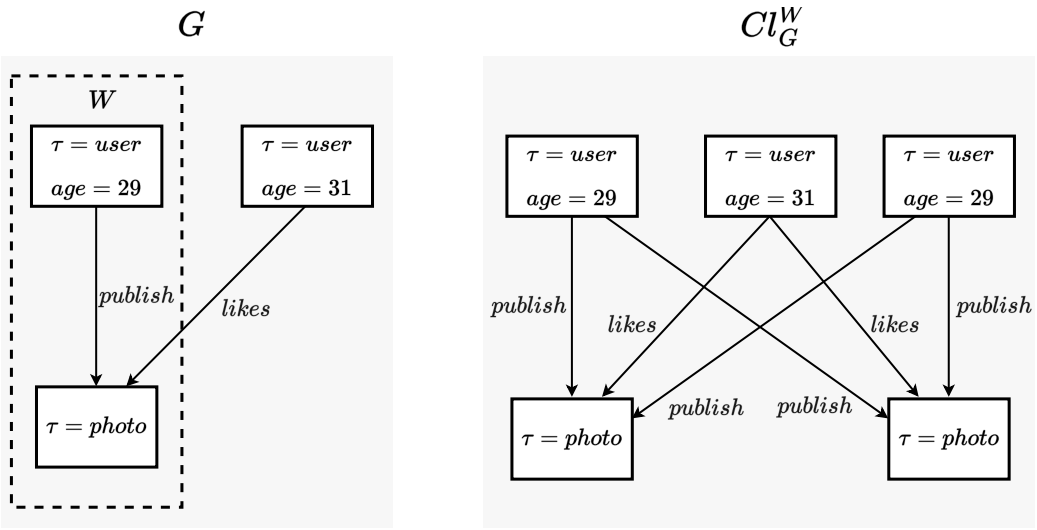

Figure 3. Clone of a graph by duplication.

The next result indicates that duplicating positive nodes does not change the *meaning* of the queries.

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

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

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

□

When refining a query to find complementary sets of selected subgraphs, we define the concept of a refinement set as central:

Definition 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 now introduce refinement sets to enhance simpler queries for expressiveness. $Q \in \mathcal{Q}$ is prefixed, and $\top$ represents a 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 (Figure 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:

$$Q_1 = Q \wedge Q_m,$$
$$Q_2 = Q \wedge \neg Q_m$$

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 is beneficial for adding new nodes and restrictions in the future.

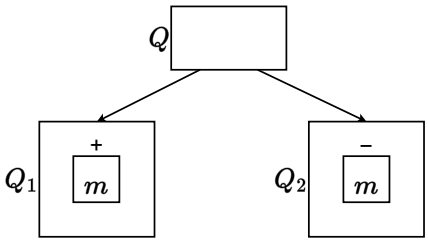

Figure 4. The *add node* refinement.

The second refinement allows for the establishment of edges between query nodes that already exist. To obtain a valid refinement set, the inclusion of edges is limited to positive nodes. Subsequently, the nodes marked with a positive/negative sign represent cloned nodes whose α property has been designated as positive/negative.

Theorem 5. *(Add a 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 (Figure 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:

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

□

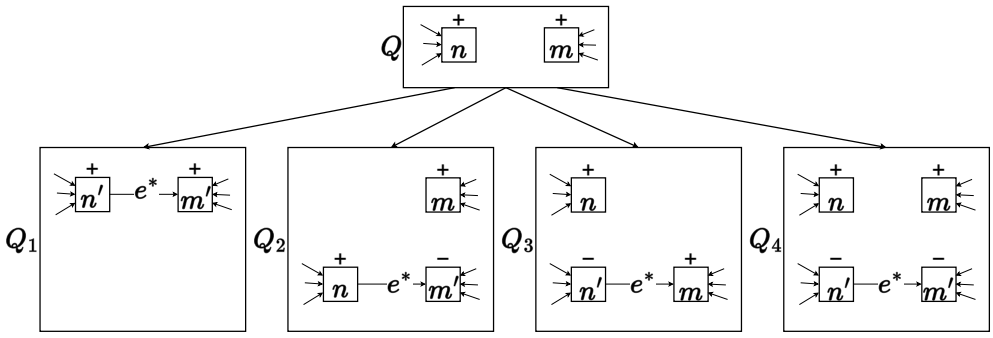

Figure 5. The *add edge* refinement (simplified).

Next, an additional predicate is added to an existing edge through the following operation, limited to positive edges connecting positive nodes.

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 *(Figure 6).*

Proof. The proof is similar to the previous ones. □

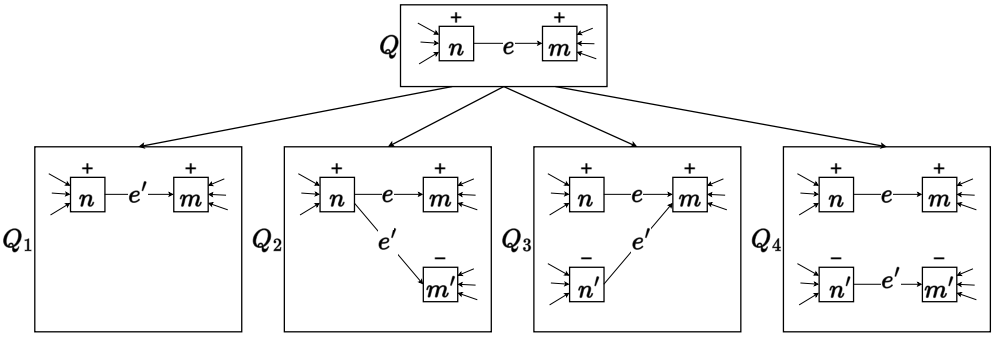

Figure 6. The *add the predicate to edge* refinement (simplified).

Finally, the last step involves adding predicates to existing nodes. This operation is only permitted when the affected nodes are positive, including 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 (Figure 7).*

Proof. The proof resembles earlier ones. It is important to consider that modifying node n not only alters the associated predicate but also those of its neighboring nodes. Additionally, the set of functions $\{+,-\}^{\mathcal{N}_Q(n)}$ encompasses all feasible sign assignments for the nodes within the neighborhood. □

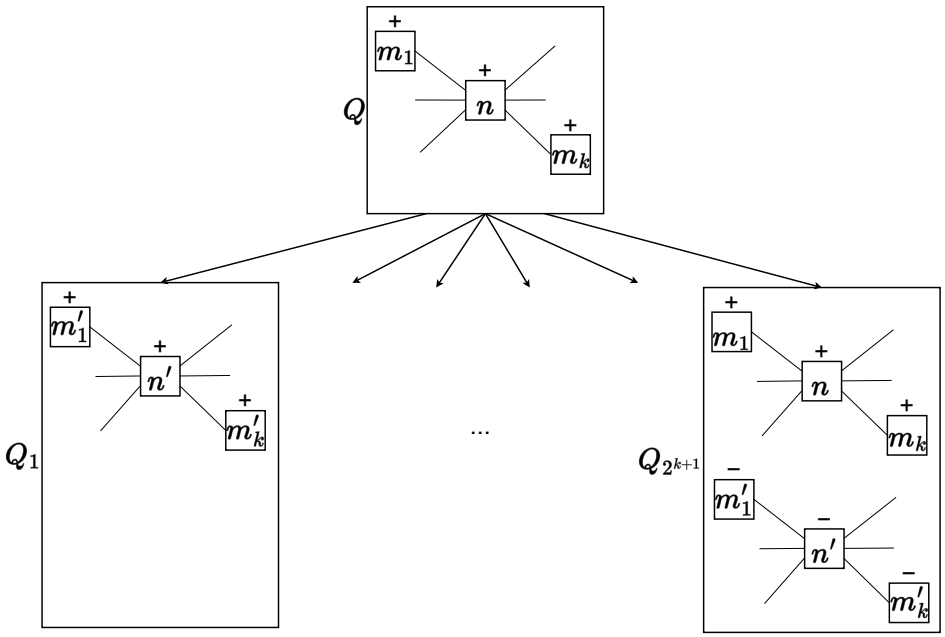

Figure 7. The *add the predicate to node* refinement (simplified).

Also note that simplified versions of the refinement sets are shown in Figures 4–7. Section 3.4 explains how to obtain these simplifications.

Obtaining a complementary query from the structure is a challenging task. Nonetheless, graph analyses often require sequences of queries to verify properties related to refinement and complementarity. To bridge this gap, this section introduces refinement operations. These operations facilitate the construction of an embedded partition tree, where nodes are labeled as illustrated in Figure 8:

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

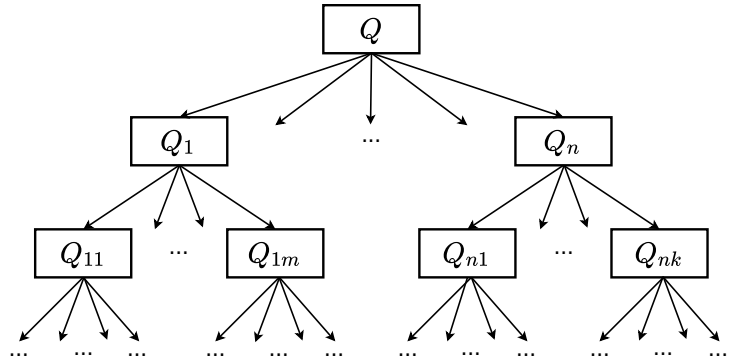

Figure 8. Refinement tree.

Refinement sets presented herein offer one approach, rather than the sole approach. 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 simplify a query into an equivalent one by applying certain operations.

Definition 11. *We define $Q' \subseteq Q$ as* redundant *in Q if $Q \equiv Q - Q'$. Here, $Q - Q'$ represents the subgraph of Q given by*

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

One initial finding that enables the acquisition of simplified versions of a query by eliminating superfluous nodes is as follows (note that from the following two results, we only give an idea of the proof, which can be very laborious but straightforward from the above constructions):

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.

Proof. A query (Q) comprises nodes and their relationships. Each query node imposes constraints on the subgraph that is evaluated, including the presence or absence of nodes and the paths in which they participate. These restrictions should be considered during the evaluation. If there are two nodes, $n, m \in V_Q$, with $\alpha(n) = \alpha(m)$ and $\theta_n \equiv \theta_m$, and for each $e \in \gamma(n)$, there exists $e' \in \gamma(m)$, with $\alpha(e) = \alpha(e')$, $\theta_e \equiv \theta_{e'}$ and $\gamma(e) \setminus \{n\} = \gamma(e') \setminus \{m\}$, both nodes apply identical restrictions to the subgraph being evaluated. Therefore, deleting one of them will not change the assessment on the subgraph. □

In particular, m is a duplicate of n, but potentially with additional connected edges. A comparable outcome for the edges can be achieved:

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.*

Proof. Following the same reasoning as the previous theorem, if there are two edges, e and e', in E_Q, which connect the same two nodes in a query and $\alpha(e) = \alpha(e')$ and $\theta_e \to \theta_{e'}$, the constraint imposed by e implies the constraint imposed by e'. Therefore, eliminating e' would maintain the set of matching subgraphs.

□

From these two findings, we can streamline the refinement sets that were established in Section 3.3 by removing redundant elements in succession after cloning.

3.5. Graph Query Examples

For illustrative purposes, this section presents a series of queries on a toy graph dataset. Figure 9 illustrates a segment of the Star Wars graph (http://console.neo4j.org/?id=StarWars, accessed on 1 September 2023).

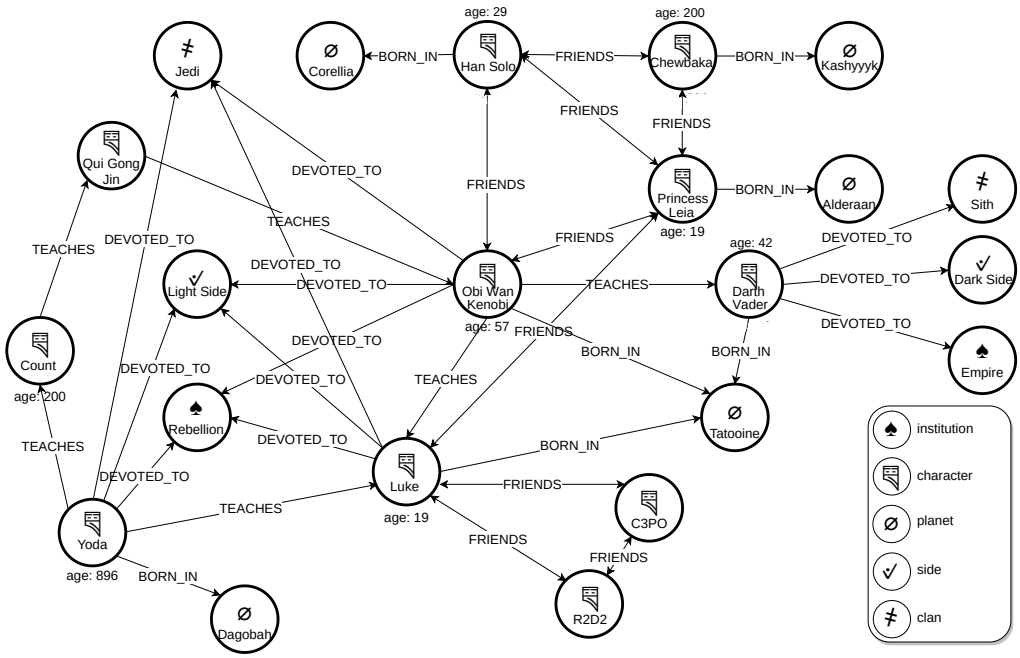

Figure 9. Section of the Star Wars graph.

To streamline query and graph representation, we will convert τ, a property denoting node and edge types, into labels for edges or icons for nodes. Additionally, the node properties denoted by *name* will be written on them, and the undirected edges will be represented by bidirectional arrows. The property α will be represented directly on the query elements using $+/-$ symbols, and we will write the binary predicate θ directly on the elements (except for tautologies). When expressions such as $\tau(\rho) = X$ are in the predicate of an edge, X is written directly and interpreted as a regular expression to be verified by the sequence of τ properties of the links in the associated graph path.

Query 1 (Figure 10) can be interpreted as follows: *Two characters are connected by a TEACHES relationship, where the master is over 500 years old and both are devoted to the Jedi.* This query utilizes structural constraints through the presence of edges and predicates with properties such as τ, *name*, and *age*. For example, in Figure 9, the subgraph comprising `Yoda`, `Luke`, and their `TEACHES` relationship satisfies this query.

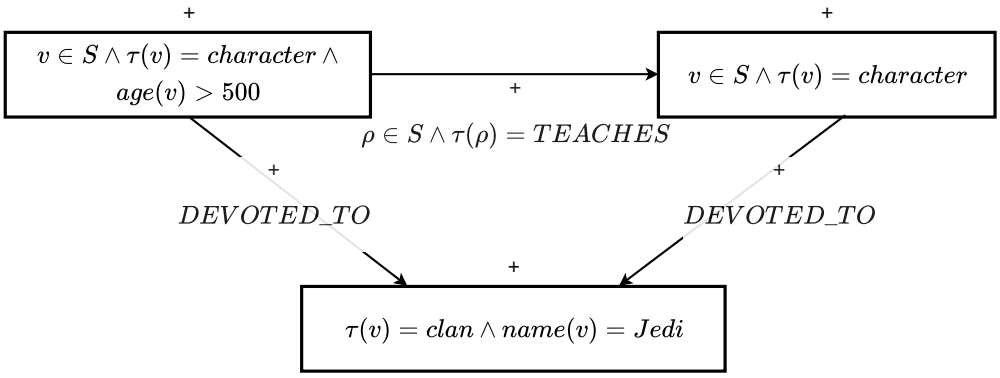

Figure 10. Query 1.

Query 2 (Figure 11) outlines a cyclic query that utilizes the `FRIENDS` relationships. It can be verified on any subgraph containing three characters who are friends with each other (for example, the subgraph formed by `Han Solo`, `Chewbacca`, `Princess Leia`, and the `FRIENDS` relationships in Figure 9).

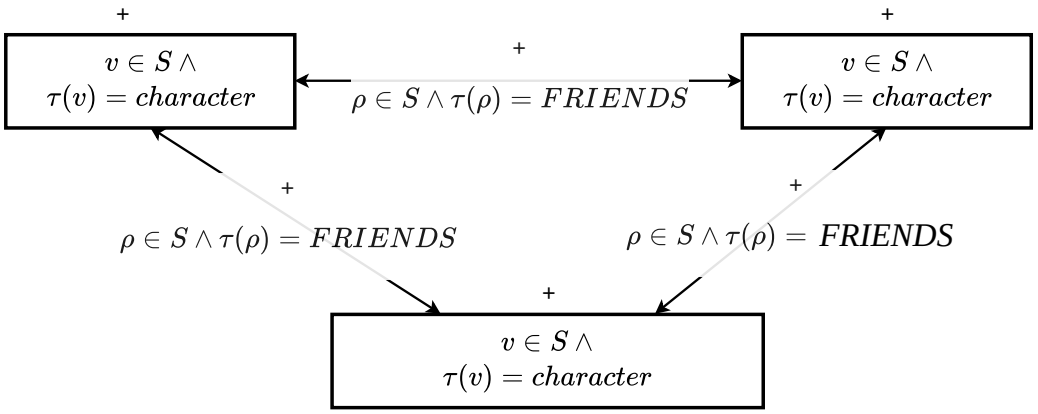

Figure 11. Query 2.

Query 3 (Figure 12) can be interpreted as follows: *A character with more than three outgoing relationships, not belonging to the Sith clan, connected through a path consisting of any number of FRIENDS and TEACHES relationships with an individual from Alderaan.* In this scenario, a regular expression is employed to denote a path consisting of an unspecified amount of `FRIENDS` and `TEACHES` relationships. Additionally, an auxiliary function, $gr_s(v) \in L$, is utilized to reference the outgoing degree of node v. This query will be validated by any subgraph that contains `Luke` or `Obi-Wan Kenobi`.

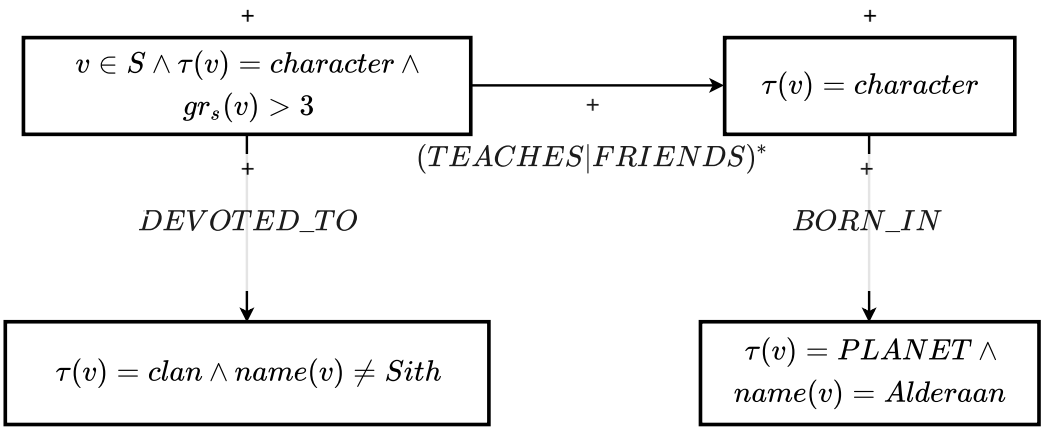

Figure 12. Query 3. Where $(a|b)^*$ denotes any number of edges of type a or b.

3.6. Computational Complexity

Query systems based on graph isomorphisms (most of the existing ones) face NP-complete complexity [34,35].

The preceding section presented our graph query framework that is reliant on logical predicates. In this section, we will demonstrate that the assessment of queries is polynomial, even in the case of cyclic queries. This is achievable by imposing two constraints. First, the length of paths that are illustrated by links in the query is restricted by the constant k, and second, the complexity of the predicates used in nodes and edges is polynomial.

To verify $S \models Q$, it is necessary to examine each predicate $Q_n(S)$ linked to all nodes $n \in V_Q$. Furthermore, each predicate $Q_n(S)$ linked to a node in Q requires assessing one $Q_e(v, S)$ predicate for every link $e \in \gamma_Q(n)$. Thus, initially, we focus on evaluating the computational complexity linked with the link predicates $Q_e(v, S)$. Subsequently, we will examine the complexity associated with the node predicates $Q_n(S)$. Ultimately, we will show that the query complexity is polynomial.

As previously defined, the computational complexity of evaluating predicates attributed to both nodes and edges within a query (Q) is polynomial, denoted by $\mathcal{O}(p)$. The predicates $Q_e(v, S)$ related to edges in a query verify the existence of a path ρ in the graph G starting/ending in v that satisfies its own predicate $\theta_e(\rho, S)$ and with the source and destination nodes that satisfy the predicates $\theta_{e^o}(\rho^o, S)$ and $\theta_{e^i}(\rho^i, S)$, respectively. Thus, the complexity involved in evaluating a particular path is $\mathcal{O}(3p) = \mathcal{O}(p)$.

The computational complexity to verify the existence of a path beginning or ending at a node v in V, satisfying the aforementioned criteria, is $\mathcal{O}(p \times |V|^k)$. Here, $|V|^k$ denotes the number of paths that start or end at v under the condition that they are no longer than k in length. As the number of links commencing or ending at a node $n \in V_Q$ is bound by $|E_Q|$, the computational complexity involved in the node predicate $Q_n(S)$ is $\mathcal{O}(p \times |V|^k \times |E_Q|)$.

Finally, if the query consists of $|V_Q|$ nodes, the complexity of checking the query $Q(S)$ is $\mathcal{O}(p \times |V|^k \times |E_Q| \times |V_Q|)$. It is evident that the constant k (path length bound) significantly impacts the execution of such queries, as it determines the exponent of the complexity.

The efficient operation of a graph query framework is crucial when dealing with large-scale datasets that are commonly found in real-world applications. Notably, when such a system is employed as the kernel of relational machine learning algorithms, as we will demonstrate in the following section, the ability to perform query operations in polynomial time, even for cyclic queries, is fundamental.

4. Relational Machine Learning

In this section, we will leverage the advantages of the framework presented to acquire relational classifiers on graph datasets. To elaborate, we will initiate from a labeled subgraph set within a graph dataset and then develop a pattern search technique founded on information gain to obtain typical patterns for each subgraph class.

4.1. Information-Gain Pattern Mining

To obtain characteristic patterns of subgraph classes using the previous graph query framework, a top-down decision tree induction will be conducted to explore the pattern space. Within the trees' internal nodes, graph queries will serve as test tools. The best refinement sets will be identified during the tree construction process, resulting in queries that define classes within the graph dataset.

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

- 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 tree learning procedure is standard: A tree is initialized comprising one node (the root) linked to the entire set of training, $\mathcal{L}$. The initial query, Q_0, corresponds to all its constituents ($\forall S \in \mathcal{L}, S \vDash Q_0$). The subsequent stage involves determining which refinement set generates the maximum information gain while separating $\mathcal{L}$, and applying it to Q_0. For each query in the refinement set, a corresponding child node is created, and $\mathcal{L}$ samples are transmitted through it. A child with a matching associated query is guaranteed to exist since it is a refinement set of Q_0. The recursive process continues for each new node until a stop condition is met. At that point, the node becomes a leaf associated with a class. Note that the decision trees derived from this approach are not predominantly binary, unlike the prevalent trees in the literature.

4.2. Relational Tree Learning Examples

Here, we introduce some practical instances to demonstrate the process of performing relational learning by using the query framework and refinement sets. The refinement operations will be as mentioned in Section 3.3. A critical factor is that all subgraphs in a decision node belong to the same class, which we require as the stopping condition. Initially, we will focus on node classification problems before proceeding to classify more intricate structures.

Consider the small social network illustrated in Figure 13, portraying users and items in a graph. The objective is to classify the nodes based on the patterns extracted from the dataset.

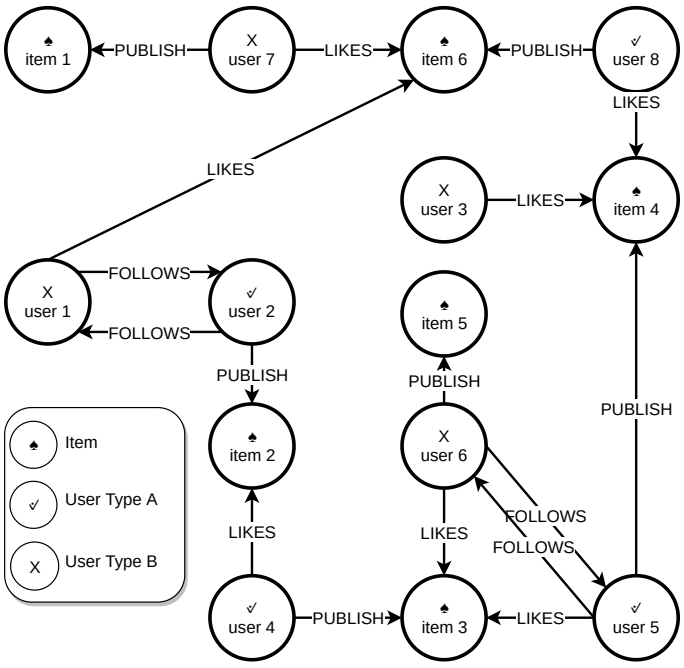

Figure 13. Social network toy.

Beginning with a training set composed of all nodes in the graph, Figure 14 displays the relational decision tree acquired through the process elucidated in Section 4.1. Negative nodes/edges are identified with a cross, while nodes with predicate $\theta(v, S) := v \in S$ are larger and white in hue. This tree accurately assigns types (`User A`, `User B`, or `Item`) to all nodes in the graph by exploiting relational information from the network. Furthermore, on the leaves of the tree, distinctive patterns are acquired for each node type, which can be used to directly assess nodes and clarify future classifications.

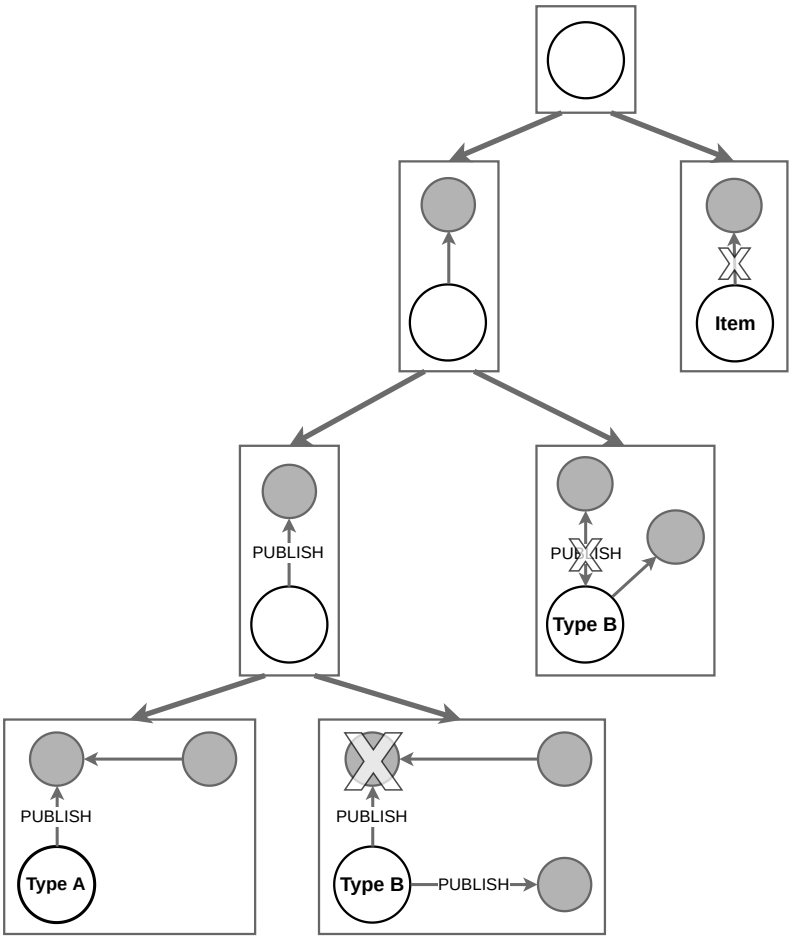

Figure 14. Node type classifier.

Similarly, by utilizing each character node in the Star Wars toy graph (Figure 9) and the corresponding `species` property as a training dataset, the relational decision tree shown in Figure 15 categorizes and explains each character's species in the graph. The leaf patterns of the tree characterize each species: human characters are born friends of `Luke`, droids are unborn friends of `Luke`, Wookiees are those born in `Kashyyyk`, etc.

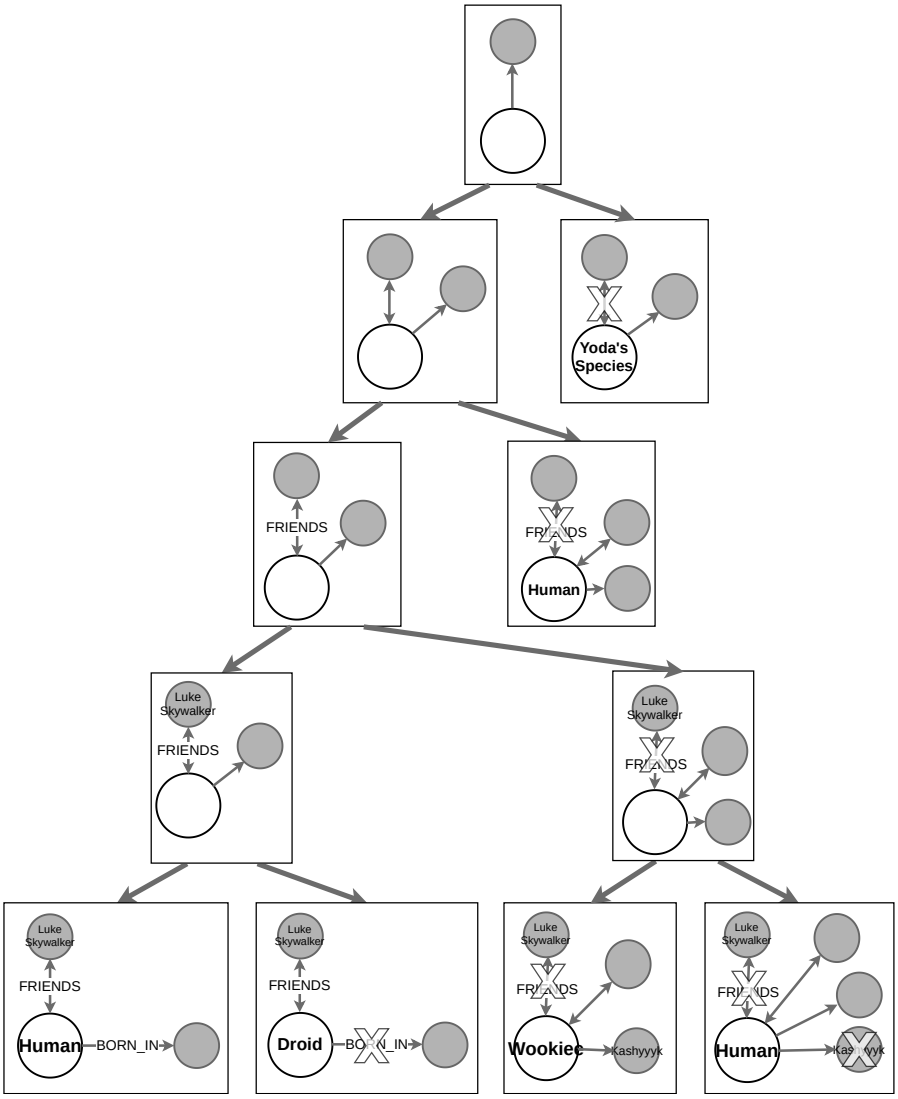

Figure 15. Character species classifier.

5. Conclusions and Future Work

This paper's main contribution consists of a novel framework for graph queries that permits the polynomial cyclic assessment of queries and refinements based on atomic operations. The framework's ability to apply refinements in relational learning processes was also demonstrated. In addition, the presented framework fulfills several essential requirements. The system utilizes consistent grammar for both queries and evaluated structures. It allows the assessment of subgraphs beyond individual nodes and supports cyclic queries within polynomial time (where the length of the query path is limited). The system offers a controlled and automated query construction via refinements, and the refinement sets constitute embedded partitions of the evaluated structure set, making them effective tools for top-down learning techniques.

Graph isomorphism-based query systems exhibit exponential complexity when presented with cyclic queries. Additionally, if a projection is necessary for pattern verification, evaluating the non-existence of specific elements becomes difficult or even impossible. However, the query graph framework offered here assesses the existence/non-existence of paths and nodes in a graph rather than demanding isomorphisms, thus enabling the evaluation of cyclic patterns in polynomial time.

After conducting an initial and fully functional proof-of-concept implementation, the graph query framework's capabilities have been demonstrated through experimen-

tation. This methodology has been explicitly applied in relational learning procedures, as demonstrated in Section 4.2, and the results of these experiments have shown that interesting patterns can be extracted from relational data. This is of great significance in both explainable learning and automatic feature extraction tasks. The results' graphs were obtained via our proof-of-concept implementation on a graph database and by employing the *matplotlib* library [36].

Despite the presented query definition utilizing binary graph datasets (rather than hypergraphs), it can be implemented on hypergraph data as well. This is due to the fact that the concept of a path, which connects pairs of nodes, is independent of the edge arity involved. For the sake of simplicity, and due to the absence of true hypergraph databases, our queries are limited to the binary case. Nevertheless, they can be adapted to more universal cases once the usage of hypergraphs becomes more widespread.

Also, in Section 3.3, a basic and reliable set of refinement operations is provided. However, these operations should not be considered as the most suitable solution for all types of learning tasks. To achieve complex queries and to prevent plateaus in the pattern space, more complex refinement families can be established. For example, it is possible to combine the *add edge* and *adding property to an edge* operations into one step, thereby reducing the number of steps required. If executed properly, unifying the refinements based on the frequency of structural occurrences in a graph, for instance, can lead to faster versions of learning algorithms at the expense of covering a broader query space. This work provides theoretical tools to support the accuracy of new refinement families. Future research will focus on developing automated methods to generate refinement sets based on a given learning task and the specific characteristics of the graph dataset. Extracting statistics from the graph data for the automatic generation of such sets can result in significant optimizations.

It is concluded that establishing effective techniques for matching graph patterns and learning symbolic relationships is feasible, resulting in a systematic exploration of the pattern space, a high degree of expressiveness in queries, and computational costs of implementations kept within reasonable order.

Patterns associated with the leaves in obtained decision trees can be used to characterize subgraph categories. Moreover, the path from the root node to the corresponding leaf of the decision tree for a particular input can be used to justify decisions; this is a beneficial feature in various sensitive applications, such as decision trees. In addition, patterns obtained from the graph learning procedure presented can serve as features in other machine learning methods. Once the patterns are acquired, they can serve as Boolean features for subgraph modeling, enabling non-relational machine learning methods to learn from them.

Learning of relational decision trees can be utilized by *ensemble* methods (such as random forest), and although the explanatory power is diluted when multiple trees are combined, their predictive power can be greatly enhanced. Therefore, it is essential to investigate the probabilistic amalgamation of queries to generate patterns that can be interpreted as probabilistic decision tools.

Furthermore, while a relational decision tree learning technique has been employed, additional machine learning algorithms can be evaluated alongside this query framework to investigate more opportunities for relational learning.

Author Contributions: Conceptualization, P.A.-B. and F.S.-C.; methodology, P.A.-B. and F.S.-C.; software, P.A.-B.; validation, P.A.-B., F.S.-C. and J.B.-D.; formal analysis, P.A.-B., F.S.-C. and J.B.-D.; investigation, P.A.-B. and F.S.-C.; resources, P.A.-B. and F.S.-C.; data curation, P.A.-B. and F.S.-C.; writing—original draft preparation, P.A.-B., F.S.-C. and J.B.-D.; writing—review and editing, P.A.-B., F.S.-C. and J.B.-D.; visualization, P.A.-B.; supervision, F.S.-C.; project administration, P.A.-B. and F.S.-C.; funding acquisition, J.B.-D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Project PID2019-109152G, funded by the State Investigation Agency (Agencia Estatal de Investigación), MCIN/AEI/10.13039/501100011033.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dong, X.; Gabrilovich, E.; Heitz, G.; Horn, W.; Lao, N.; Murphy, K.; Strohmann, T.; Sun, S.; Zhang, W. 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, New York, NY, USA, 24–27 August 2014; ACM: New York, NY, USA, 2014; pp. 601–610. [CrossRef]
2. García-Jiménez, B.; Pons, T.; Sanchis, A.; Valencia, A. Predicting protein relationships to human pathways through a relational learning approach based on simple sequence features. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2014**, *11*, 753–765. [CrossRef] [PubMed]
3. Jacob, Y.; Denoyer, L.; Gallinari, P. 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, New York, NY, USA, 24–28 February 2014; ACM: New York, NY, USA, 2014; pp. 373–382. [CrossRef]
4. Lee, N.; Hyun, D.; Na, G.S.; Kim, S.; Lee, J.; Park, C. Conditional Graph Information Bottleneck for Molecular Relational Learning. *arXiv* **2023**, arXiv:2305.01520.
5. Fan, W. Graph Pattern Matching Revised for Social Network Analysis. In Proceedings of the 15th International Conference on Database Theory (ICDT '12), Berlin, Germany, 26–29 March 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 8–21. ISBN 9781450307918. [CrossRef]
6. Tang, L.; Liu, H. Relational learning via latent social dimensions. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; ACM: New York, NY, USA, 2009; pp. 817–826.
7. Jiang, J.Q. Learning protein functions from bi-relational graph of proteins and function annotations. In *Algorithms in Bioinformatics;* Przytycka, T.M., Sagot, M.-F., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 128–138.
8. Camacho, R.; Pereira, M.; Costa, V.S.; Fonseca, N.A.; Adriano, C.; Simões, C.J.; Brito, R.M. A relational learning approach to structure-activity relationships in drug design toxicity studies. *J. Integr. Bioinform.* **2011**, *8*, 176–194. [CrossRef]
9. De Raedt, L.; Dumančić, S.; Manhaeve, R.; Marra, G. From Statistical Relational to Neural-Symbolic Artificial Intelligence. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20), Yokohama, Japan, 7–15 January 2021.
10. Wang, Y.; Wang, W.; Liang, Y.; Cai, Y.; Liu, J.; Hooi, B. NodeAug: Semi-Supervised Node Classification with Data Augmentation. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20), Virtual Event, 23–27 August 2020; pp. 207–217. [CrossRef]
11. Kazemi, S.M.; Poole, D. RelNN: A Deep Neural Model for Relational Learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), New Orleans, LA, USA, 2–7 February 2018.
12. Pacheco, M.L.; Goldwasser, D. Modeling Content and Context with Deep Relational Learning. *Trans. Assoc. Comput. Linguist.* **2021**, *9*, 100–119. [CrossRef]
13. Ahmed, K.; Altaf, A.; Jamail, N.S.M.; Iqbal, F.; Latif, R. ADAL-NN: Anomaly Detection and Localization Using Deep Relational Learning in Distributed Systems. *Appl. Sci.* **2023**, *13*, 7297. [CrossRef]
14. Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; Sun, M. Graph Neural Networks: A Review of Methods and Applications. *AI Open* **2020**, *1*, 57–81. [CrossRef]
15. Wu, L.; Cui, P.; Pei, J.; Zhao, L.; Guo, X. Graph Neural Networks: Foundation, Frontiers and Applications. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), Washington, DC, USA, 14–18 August 2022; pp. 4840–4841. [CrossRef]
16. Latouche, P.; Rossi, F. Graphs in machine learning: An introduction. *arXiv* **2015**, arXiv:1506.06962.
17. Knobbe, A.J.; Siebes, A.; Wallen, D.V.D.; Syllogic, B.V. Multi-relational decision tree induction. In Proceedings of the PKDD'99, Prague, Czech Republic, 15–18 September 1999; Springer: Berlin/Heidelberg, Germany, 1999; pp. 378–383.
18. Gallagher, B. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI Fall Symp.* **2006**, *6*, 45–53.
19. Fan, W.; Li, J.; Ma, S.; Tang, N.; Wu, Y.; Wu, Y. Graph pattern matching: From intractable to polynomial time. *Proc. VLDB Endow.* **2010**, *3*, 264–275. [CrossRef]
20. Milner, R. *Communication and Concurrency*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1989.
21. Zou, L.; Chen, L.; Özsu, M.T. Distance-join: Pattern match query in a large graph database. *Proc. VLDB Endow.* **2009**, *2*, 886–897. [CrossRef]
22. Henzinger, M.R.; Henzinger, T.A.; Kopke, P.W. Computing simulations on finite and infinite graphs. In Proceedings of the Proceedings of IEEE 36th Annual Foundations of Computer Science, Milwaukee, WI, USA, 23–25 October 1995; pp. 453–462.
23. Ma, S.; Cao, Y.; Fan, W.; Huai, J.; Wo, T. Strong Simulation: Capturing Topology in Graph Pattern Matching. *ACM Trans. Database Syst.* **2014**, *39*, 1–49. [CrossRef]

24. Nickel, M.; Murphy, K.; Tresp, V.; Gabrilovich, E. A review of relational machine learning for knowledge graphs. *Proc. IEEE* **2016**, *104*, 11–33. [CrossRef]
25. Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; Yakhnenko, O. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26*; Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2013; pp. 2787–2795. Available online: http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf (accessed on 1 September 2023).
26. Chang, K.W.; Yih, W.T.; Yang, B.; Meek, C. Typed tensor decomposition of knowledge bases for relation extraction. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25–29 October 2014; ACL—Association for Computational Linguistics: Stroudsburg, PA, USA, 2014. Available online: https://www.microsoft.com/en-us/research/publication/typed-tensor-decomposition-of-knowledge-bases-for-relation-extraction/ (accessed on 1 September 2023).
27. Geamsakul, W.; Matsuda, T.; Yoshida, T.; Motoda, H.; Washio, T. Classifier construction by graph-based induction for graph-structured data. In *Advances in Knowledge Discovery and Data Mining, Proceedings of the 7th Pacific-Asia Conference, PAKDD 2003, Seoul, Repblic of Korea, 30 April–2 May 2003;* Whang, K.-Y., Jeon, J., Shim, K., Srivastava, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 52–62. [CrossRef]
28. Leiva, H.A.; Gadia, S.; Dobbs, D. Mrdtl: A multi-relational decision tree learning algorithm. In Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003), Szeged, Hungary, 29 September–1 October 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 38–56.
29. Plotkin, G. Automatic Methods of Inductive Inference. Ph.D. Thesis, University of Edinburgh, Edinburgh, UK, 1972.
30. Blockeel, H.; Raedt, L.D. Top-down induction of first-order logical decision trees. *Artif. Intell.* **1998**. [CrossRef]
31. Nguyen, P.C.; Ohara, K.; Motoda, H.; Washio, T. Cl-gbi: A novel approach for extracting typical patterns from graph-structured data. In *Advances in Knowledge Discovery and Data Mining, Proceedings of the 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, 18–20 May 2005*; Ho, T.B., Cheung, D., Liu, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 639–649. [CrossRef]
32. Almagro-Blanco, P.; Sancho-Caparrini, F. Generalized graph pattern matching. *arXiv* **2017**, arXiv:1708.03734.
33. Bonifati, A.; Fletcher, G.; Voigt, H.; Yakovets, N.; Jagadish, H.V. *Querying Graphs*; Morgan & Claypool Publishers: San Rafael, CA, USA, 2018.
34. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing STOC '71, Shaker Heights, OH, USA, 3–5 May 1971; ACM: New York, NY, USA, 1971; pp. 151–158. [CrossRef]
35. Karp, R.M. On the computational complexity of combinatorial problems. *Networks* **1975**, *5*, 45–68. [CrossRef]
36. Hunter, J.D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **2007**, *9*, 3. [CrossRef]