Datalog-based Reasoning with Heuristics over Knowledge Graphs

Teodoro Baldazzi[1,*], Davide Benedetto[1], Matteo Brandetti[2], Adriano Vlad[3,2], Luigi Bellomarini[4] and Emanuel Sallinger[2,3]

[1] *Università Roma Tre, Department of Computer Science and Engineering, Rome, Italy*
[2] *TU Wien, Faculty of Informatics, Vienna, Austria*
[3] *University of Oxford, Department of Computer Science, Oxford, UK*
[4] *Banca d'Italia, Italy*

Abstract

Datalog+/- has recently emerged as a family of powerful languages for ontological reasoning on knowledge graphs (KGs). Yet, performing reasoning tasks on real-world scenarios that feature large inputs and recursions, e.g., answering Boolean Conjunctive Queries (BCQs), can be very onerous even for modern Datalog-based systems. To tackle this problem, we introduce a novel reasoning technique that adopts heuristic search strategies to guide the evaluation of Datalog+/- programs and optimize BCQ answering. We apply our methodology to efficiently solve time- and space-demanding tasks on financial KGs.

Keywords

Datalog+/-, reasoning, knowledge graph, heuristic search

1. Introduction

Recent years have witnessed a rising interest, both in academia and industry, towards querying and exploiting large amounts of data in the form of *knowledge graphs* (KGs). This led to the development of intelligent systems that manage such data as the extensional component of KGs and infer new intensional knowledge via *ontological reasoning* mechanisms. To achieve this, modern applications require powerful logic languages for knowledge representation [1] that provide full support for recursion, joins and existential quantification, essential features for graph navigation and ontological reasoning, while sustaining tractability [2].

Datalog$^{\pm}$ Reasoning. Among them, Datalog$^{\pm}$ [3, 4, 5, 6] members (technically, *fragments*) have recently emerged, extending Datalog [7, 8, 9, 10] with existential quantification and covering, in some cases, the above requirements. Datalog$^{\pm}$ rules are function-free Horn clauses that may include existentials, i.e., *tuple-generating dependencies* (TGDs). The semantics of a Datalog$^{\pm}$ program can be defined in an operational way via the *chase* [11]: given a database D and a set

Datalog 2.0 2022: 4th International Workshop on the Resurgence of Datalog in Academia and Industry, September 05, 2022, Genova - Nervi, Italy

[*] Corresponding author.

teodoro.baldazzi@uniroma3.it (T. Baldazzi); davide.benedetto@uniroma3.it (D. Benedetto); matteo.brandetti@gmail.com (M. Brandetti); adriano.vlad@gmail.com (A. Vlad); luigi.bellomarini@bancaditalia.it (L. Bellomarini); sallinger@dbai.tuwien.ac.at (E. Sallinger)

Σ of rules in input, it adds new tuples to D until Σ is satisfied, introducing freshly generated symbols (technically, *labelled nulls*) that act as placeholders for existential quantification. The navigational capabilities offered by Datalog, empowered by recursion in combination with arbitrary joins, are vital for expressing complex reasoning tasks over KG.

Nonetheless, graph traversal remains a very time- and space-demanding task, especially in the presence of large inputs, as it potentially involves a blowup of the generated facts [12]. In this context, answering *Boolean Conjunctive Queries* (BCQs) – i.e., queries whose answer is a Boolean value – is a particularly relevant case, as often only a subset of the generated facts actually contributes to answering the query [13]. Indeed, standard BCQ evaluation approaches (both *materialization* and *streaming* ones) are *domain-unaware*, that is, they do not exploit external knowledge of the domain to guide chase and query answering. This may cause a significantly superfluous computation in the number of generated facts, heavily affecting reasoning performance. This limitation also characterizes state-of-the-art optimization techniques such as *semi-naive* evaluation [14], which prevents the discovery of duplicated facts by evaluating only the newly generated ones, and *Magic Sets* [15], a top-down strategy that, starting from the query, builds derivation trees that search for matching facts in the database.

In this work, we investigate the generation of superfluous facts in recursive settings when answering BCQs. We contribute a novel reasoning approach that tackles this problem and provides an optimized way to evaluate Datalog$^{\pm}$ rules. Let us consider the following example.

Example 1. *A traffic volume scenario on a geographic knowledge graph* [16].

$$Road(p_1, p_2, tv), p_1 = A \rightarrow Route(p_1, p_2, tv) \tag{1}$$

$$Route(p_1, p_2, tv_1), Road(p_2, p_3, tv_2), p_1 \neq p_3 \rightarrow Route(p_1, p_3, tv_1 + tv_2) \tag{2}$$

$$Route(p_1, p_2, tv), ttv = msum(tv), ttv \geq thold \rightarrow HighTraffic(p_1, p_2) \tag{3}$$

A road from city A to a city p_2 identifies a route featuring a traffic volume tv (rule 1). If there is a route from p_1 to p_2 and a road from p_2 to another city p_3, then there is a route from p_1 to p_3. The resulting traffic volume is the sum of the one from p_1 to p_2 and from p_2 to p_3 (rule 2). The overall set of itineraries from p_1 to p_2 is highly trafficated (HighTraffic) if the total traffic volume of all possible routes from p_1 to p_2 is greater than or equal to a threshold thold *(rule 3).*

The *msum* operator represents a *monotonic aggregation* [12] that sums all the traffic volumes of the routes from p_1 to p_2. Note that distinct routes may have one or more roads in common. Consider, as ontological reasoning task, the BCQ $Q = HighTraffic(A, B) \wedge HighTraffic(A, L)$ to check whether the set of routes from A to B and the one from A to L are highly trafficated, i.e., their traffic volume is greater than or equal to thold = 18. The graph in Figure 1, represents the roads (black edges, labelled with their daily average traffic volume in the order of thousands) that connect pairs of cities (nodes). Consider an input database D containing an extensional fact of the form *Road*(x,y,tv) for each road in the graph. It can be observed that the answer to Q is indeed positive. However, there are

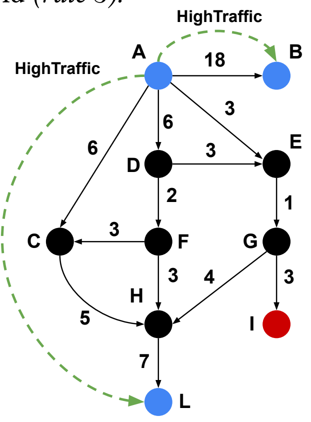


Figure 1: KG Example 1

multiple ways to activate the above rules on D to achieve such answer, causing the generation of a distinct number of facts and, consequently, an impact on the performance of the computation. For instance, by activating rule 2 from *Route*(A,E,3) and *Road*(E,G,1), we generate *Route*(A,G,4). Similarly, *Route*(A,F,8) derives from *Route*(A,D,6) and *Road*(D,F,2). Then, by triggering again rule 2 on such newly generated facts, we create *Route*(A,I,7) from *Road*(G,I,3), *Route*(A,H,8) from *Road*(G,H,4), and *Route*(A,H,11) from *Road*(F,H,3). Finally, the aggregated traffic volume of *Route*(A,L,15) and *Route*(A,L,18) exceeds 18, thus *HighTraffic*(A,L) is produced via rule 3. Indeed, such approach generates both facts that are not used to answer Q (e.g. *Route*(A,I,7), and facts that provide an unnecessary contribution to pass the threshold (e.g. *Route*(A,L,15)).

Informed Search and Reasoning. With this work, we propose a novel reasoning methodology to efficiently solve BCQ answering tasks in the presence of large inputs and recursive Datalog$^{\pm}$ settings, limiting the generation of superfluous facts in the chase. To achieve this, we consider the task as an informed search problem and we optimize the reasoning process by injecting into the program specific heuristics that prioritize the selection of facts for rule evaluation. We focus on Warded Datalog$^{\pm}$ [17], a powerful Datalog$^{\pm}$ fragment. It encompasses both a high expressive power, capturing all SPARQL queries under OWL 2 QL entailment regime and set semantics, as well as a very good trade-off with data complexity, featuring PTIME for the reasoning [2]. Moreover, thanks to its syntactic properties, the Warded fragment ensures the termination of the chase procedure [18]. With reference to Example 1, we prioritize the rule activation for roads with the highest traffic volume and whose destination node features the highest number of roads leading into it. Indeed, by applying our methodology with such heuristic, we activate rule 1 from *Road*(A,C,6), since it features 2 roads leading into C and a traffic volume of 6. Similarly, we then trigger rule 2 on *Route*(A,C,6) and *Road*(C,H,5), generating *Route*(A,H,11). Finally, *Route*(A,L,18) is created by prioritizing *Road*(H,L,7): since its traffic volume is equal to thold, *HighTraffic*(A,L) is produced via rule 3.

More in detail, our **contributions** can be summarized as follows.

- We present **heuristic-based reasoning**, a novel approach to optimize the evaluation of BCQ answering tasks. We formalize ontological reasoning to answer BCQs as a search problem and we exploit informed search techniques to evaluate Warded Datalog$^{\pm}$ TGDs in an efficient fashion, guiding rule activation according to fitting heuristics.
- We discuss **real-world KG applications** of our methodology to solve two computationally hard scenarios in the financial domain, namely *Close Link* and *Company Control*, by employing empirically-defined heuristics [19, 20].
- We provide an **experimental evaluation** of heuristic-based reasoning in recursive settings. Specifically, we compare our ad-hoc implementation, based on *Best-First* and A^* search algorithms, with state-of-the-art BCQ answering techniques for Close Link and Company Control. We show that our approach exhibits superior performance in BCQ evaluation.

Overview. The remainder of this paper is organized as follows. In Section 2 we provide the preliminary notions. In Section 3 we present our novel heuristic-based reasoning methodology. Section 4 is dedicated to the experimental evaluation. In Section 5 we discuss the related work and we draw our conclusions in Section 6.

2. Preliminaries

To guide our discussion, let us first present some preliminary notions.

Relational Foundations. Let $\mathbf{C}$, $\mathbf{N}$, and $\mathbf{V}$ be disjoint countably infinite sets of *constants*, (*labelled*) *nulls* and *variables*, respectively. A (*relational*) *schema* $\mathbf{S}$ is a finite set of relation symbols (or *predicates*) with associated arity. A *term* is either a constant or a variable. An *atom* over $\mathbf{S}$ is an expression of the form $R(\bar{v})$, where $R \in \mathbf{S}$ is of arity $n > 0$ and $\bar{v}$ is an n-tuple of terms. A *database* (*instance*) over $\mathbf{S}$ associates to each symbol in $\mathbf{S}$ a relation of the respective arity over the domain of constants and nulls. The members of the relations are called *tuples* or *facts*. Given two conjunctions of atoms ς_1 and ς_2, we define a *homomorphism* from ς_1 to ς_2 as a mapping $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \rightarrow \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ s.t. $h(t) = t$ if $t \in \mathbf{C}$, $h(t) \in \mathbf{C} \cup \mathbf{N}$ if $t \in \mathbf{N}$ and for each atom $a(t_1, \ldots, t_n) \in \varsigma_1$, then $h(a(t_1, \ldots, t_n)) = a(h(t_1), \ldots, h(t_n)) \in \varsigma_2$. Two facts are *isomorphic* if they refer to the same predicate, feature the same constants in the same positions, and there is a bijection between labelled nulls.

Rules and Existentials. Warded Datalog$^\pm$ extends Datalog by introducing existential quantifiers and other features to make it suitable for ontological reasoning, while employing specific syntactic restrictions for tractability [18]. A Warded Datalog$^\pm$ program consists of a set of facts and *existential rules*, or *tuple-generating dependencies* (TGDs), of the form $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x},\bar{y})\rightarrow\exists\bar{z}\ \psi(\bar{x},\bar{z}))$, where φ (the *body*) and ψ (the *head*) are conjunctions of atoms over the respective predicates and the arguments are vectors of variables and constants: for brevity, quantifiers can be omitted. A fact is *intensional* if it belongs to a predicate that occurs in at least one head, otherwise it is *extensional* (or *ground*).

Ontological Reasoning and Query Answering. Intuitively speaking, an ontological reasoning task consists in answering a conjunctive query (CQ) Q over a database D, augmented with a set of logical rules Σ. More formally, given a database D over $\mathbf{S}$ and a set of TGDs Σ, we name the *models* of D and Σ as the set $\mathbf{B}$ of all databases (and we write $\mathbf{B} \models D \cup \Sigma$) such that $\mathbf{B} \supseteq D$, and $\mathbf{B} \models \Sigma$. The answer to a Boolean CQ (BCQ) $Q \leftarrow \psi(\bar{x}, \bar{z})$ over D under Σ is positive iff there exists a homomorphism *h*: $C \cup V \rightarrow C \cup N$ s.t. *h*$(\psi(\bar{x}, \bar{z})) \subseteq D$.

Chase Procedure. The semantics of a Datalog$^\pm$ program can be operationally defined via chase-based procedures [11]. They enforce the satisfaction of a set of dependencies Σ over a database D, incrementally expanding D with facts derived from the application of rules $\in \Sigma$ over D into a new database *chase*(D,Σ). Such facts possibly contain labelled nulls that act as placeholders for the existentially quantified variables. We say that *chase*(D, Σ) is the *universal model* for D and Σ, i.e., for every database in $\mathbf{B}$ that is a model for D and Σ, there is a homomorphism mapping *chase*(D, Σ) to $\mathbf{B}$. In the *oblivious* [3] version of the chase, given a TGD $\varphi(\bar{x}, \bar{y})\rightarrow\exists\bar{z}\ \psi(\bar{x}, \bar{z})$, a *chase step* can be performed over D if there exists an *applicable* homomorphism *h* that maps the atoms of $\varphi(\bar{x}, \bar{y})$ to facts in D (i.e., *h*$(\varphi(\bar{x}, \bar{y})) \subseteq$ *D*). Specifically, Warded Datalog$^\pm$ adopts an *isomorphic* chase variant that ensures termination by employing an isomorphism check over the results of the chase steps. Such variant is BCQ-equivalent to the oblivious one [18, 21]. When the homomorphism h is applicable, the fact $h^{'}(\psi(\bar{x}, \bar{z})))$ is created, where $h^{'}$ is obtained by extending h so that $h^{'}(\bar{z}) \in N$ is a fresh labelled null, for each $z_i \in \bar{z}$. Such fact is added to D if there is no isomorphism with a fact $\in D$. It is worth mentioning that the result of the isomorphic chase for D and Σ, where Σ is a set of warded TGDs, is unique.

Monotonic Aggregations. Real-world applications, such as the one discussed in Example 1, may require support for aggregate functions, which can be achieved by means of *monotonic aggregations* [12] in the Datalog$^{\pm}$ context. Formally, a rule with an aggregation is a first-order sentence of the form $\forall \bar{x}(\varphi(\bar{x}), z = \text{maggr}(x, \langle \bar{c} \rangle) \rightarrow \psi(\bar{g}, z))$, where maggr is the name of an aggregation function, $x \in \bar{x}$, and $\bar{g} \subseteq \bar{x}$ is a n-uple of group-by arguments, $\bar{c} \subseteq \bar{x}$ (with $\bar{c} \cap \bar{g} = \emptyset$) is a m-uple of variables contributing to the aggregation and z a *monotonic aggregate*, that is, an existentially quantified variable whose value is computed by the aggregation. Let us consider tuples $\langle \bar{g}, \bar{c}, x_i \rangle$ of $\varphi(\bar{x})$ and let $X_{\bar{g}}$ be each of the multi-sets of tuples of $\varphi(\bar{x})$ selected by a specific n-uple of $\bar{g}$. A rule with an aggregation maps each multi-set $X_{\bar{g}}$ into an output set $Z_{\bar{g}}$ of tuples $\langle \bar{g}, z_i \rangle$ of $\psi(\bar{g}, z)$ computed as follows. For a monotonically decreasing (increasing) aggregation function, for each $\langle \bar{g}, \bar{c}, x_i \rangle \in X_g$ we have a corresponding tuple $\langle \bar{g}, z_i \rangle \in Z_{\bar{g}}$, where the value for z_i is computed as $z_i = \text{maggr}(\bar{c}, x_i)$. This function memorizes the most recently computed aggregate and returns an updated value at each invocation such that, for each value of $\bar{c}$, the minimum (maximum) value of x_i is considered in the current aggregate [18].

Heuristic Search. Heuristic Search mechanisms allow an agent to find solutions to a problem via an heuristic. In general, a search problem can be defined as a combination of the following components: (i) an *initial state* s_0 from which the agent begins the search; (ii) a set of possible *actions* that the agent may perform in a certain state s_i; (iii) a *transition model*, describing the state $\big\langle s_i, a \big\rangle$ that results from performing an action a at a given state s_i; together, such elements represent the *state space* of the problem, that is, the set of all the reachable states from an initial one and via sequences of actions, modeled as a directed acyclic graph in which nodes are states and edges are actions; (iv) a *goal test*, which determines whether a state is a *goal state*; and (v) a *path cost function*, which assigns a numeric cost to each path. An *optimal solution* to the problem is a sequence of actions, leading from the initial state to the goal state, that features the lowest cost among all the solutions. Regarding informed search algorithms, given a current node n, the next node to be considered is selected according to an *evaluation function* $f(n)$, whose output is an estimation of the lowest path cost to reach a goal state. They employ a *heuristic function* h that encapsulates additional knowledge about the state space to support the search of optimal solutions by identifying promising paths to visit: intuitively, $h(n)$ corresponds to the estimated cost of the most promising path from n to a goal state, according to a certain heuristic. The exploited heuristics are *admissible* to the search problem, that is, the estimated cost to reach the goal state is not higher than the lowest possible cost from the current node in the path [22]. Among the types of informed search algorithms, in this work we are interested in:

- *Best-First*, which selects at each step the node that appears to be the most promising one according to the heuristic, that is, the one closest to the goal. Therefore, given a current node n, the evaluation function is $f(n) = h(n)$;
- *A**, which combines the heuristic estimation of the path cost from the current node n to a goal state ($h(n)$) with the cost required to reach n itself from the initial state ($g(n)$). Thus, the evaluation function is $f(n) = g(n) + h(n)$, which corresponds to the estimated cost of the most promising solution crossing n. For A* to be optimal in case of graph search, the employed heuristic must be *consistent*, i.e., for each node n and each successor n' of n derived from an action a, the estimated cost to reach the goal state from n is never higher than the cost to reach n' plus the estimated cost from n' to the goal state.

3. Reasoning with Heuristics

In this section, we first introduce a novel formalization of BCQ answering as an informed search problem, then we delve into how informed search methodologies can be effectively adopted to optimize BCQ evaluation over Warded Datalog$^{\pm}$.

BCQ Answering as Search Problem. Consider an instance of the BCQ answering problem $\mathcal{P}$, i.e., answering a BCQ Q over a database D under a set of warded TGDs Σ. We formalize it as an informed search problem as follows. A *chase state* of $\mathcal{P}$, denoted as s_k=*chase*^{k}(D,Σ), is a set of facts comprising the extensional ones in the initial database D and the intensional ones generated after the application of a sequence of $k > 0$ chase steps. Observe that *chase*^{k}(D,Σ) is not a unique set, as it depends on the order of applicable homomorphisms triggered in the chase procedure. The *chase state space* for $\mathcal{P}$ is the power set $P(\textit{chase}(D, \Sigma))$, i.e., all the sets of facts generated after the application of all the possible sequences of chase steps. Consider a state s_k=*chase*^{k}(D,Σ). The set of applicable homomorphisms H from Σ to *chase*^{k}(D,Σ) is the set of actions that can be performed at s_k. The application of a homomorphism $h \in H$ over s_k=*chase*^{k}(D,Σ) to generate s_{k+1}=*chase*$^{k+1}$(D,Σ) represents a transition from s_k to s_{k+1}. The *initial chase state* s_0 is the state where no chase step has been applied yet, i.e., the input database D=*chase*^{0}(D,Σ). If the BCQ Q has a positive answer, then the *goal states* G of $\mathcal{P}$ are the set of states where Q has a positive answer, i.e., *chase*^{k}(D,Σ) $\models Q$. If Q has a negative answer instead, then the *goal state* s_g of $\mathcal{P}$ is the state where no further homomorphism is applicable, i.e., *chase*(D,Σ). A *solution* $S = s_0, \ldots, s_g$ for $\mathcal{P}$ represents a sequence of chase step applications from the initial database D to s_g=*chase*^{k}(D,Σ) such that *chase*^{k}(D,Σ)) $\models Q$. The *evaluation function* $f : P(\textit{chase}(D, \Sigma)) \rightarrow \mathbb{R}_{\geq 0}$ assigns to each state s a weight according to a given heuristic. The highest weight represents an estimation of the lowest cost path from s to a goal state s_g. In the presence of an ideal f, the *optimal solution* for $\mathcal{P}$ (i.e., the sequence of chase step applications with the lowest cost path) guarantees that no superfluous facts are generated in the chase procedure when answering Q. Intuitively, given a state s=*chase*^{k}(D,Σ), the evaluation function f represents the strategy to select the most suitable applicable homomorphism in H.

Heuristic-based Reasoning. The most effective approach to define the evaluation function f is to enrich the program with external knowledge of the reasoning setting, provided by a domain expert. We represent such external knowledge as a semantic constraint on D, expressed via a heuristic function defined as follows.

Definition 1 (Ground Heuristic Function). *Consider a database D over a relational schema S. A Ground Heuristic Function (GHF) $\delta : D \rightarrow \mathbb{R}_{\geq 0}$ is a user-defined function that maps each tuple of D to a weight $w \in \mathbb{R}_{\geq 0}$.*

We exploit such a priori knowledge of the ground facts in the chase procedure to produce a more general heuristic function that also considers the inferred facts and how they are derived. Intuitively, the priority assigned to ground facts can be combined and propagated to the facts that directly descend from them.

Definition 2 (Chase Heuristic Function). *Consider a database D over a relational schema S, a set Σ of warded TGDs and a GHF δ. A Chase Heuristic Function (CHF) $\gamma : \textit{chase}(D, \Sigma) \rightarrow \mathbb{R}_{\geq 0}$ is a function that maps each fact of chase(D, Σ) to a weight $w \in \mathbb{R}_{\geq 0}$ such that $\delta \subseteq \gamma$. An Extended*

Chase Heuristic Function (ECHF) $\Gamma : P(chase(D, \Sigma)) \to \mathbb{R}_{\geq 0}$ *is a function that maps a set of tuples* $T \subseteq chase(D, \Sigma)$ *to a weight* w' *such that* $w' = \sum_{\forall t \in T} \gamma(t)$*, where* w' *is the sum of the weights assigned by* γ *to each* $t \in T$.

Based on such definitions, we now introduce the *heuristic chase step*, which represents the evaluation function of the BCQ answering problem. In fact, it allows us to select and apply, at each step in the chase, the best applicable homomorphism according to the heuristic provided by the domain expert.

Definition 3 (Heuristic Chase Step Application)**.** *Consider a set* Σ *of warded TGDs, an intermediate instance chase*^{k}*(D,*Σ*) and an ECHF* Γ*. Let* H *be the set of applicable homomorphisms from* Σ *to chase*^{k}*(D,*Σ*). A Heuristic chase step consists in the application of* $h = max[\Gamma(h_i(\sigma(\bar{x}, \bar{y})))]_{\forall h_i \in\ H}$ *over chase*^{k}*(D,*Σ*), where* $\sigma(\bar{x}, \bar{y})$ *is a warded TGD* $\in \Sigma$ *and* h *is the homomorphism* $\in H$ *whose images (i.e., tuples) maximise* Γ.

Based on such a revised form of the chase step application, we devise a new reasoning methodology, which we name *heuristic-based reasoning*. In Section 4 we show how such methodology effectively enables the generation of less unavailing facts in the chase when answering BCQs and consequently optimizes the program evaluation. As stated by Definition 2, the heuristic chase step requires a γ defined for the facts generated in the chase. The process for defining γ can be performed dynamically during the chase as follows: (1) initialize $\gamma = \delta$ by employing the knowledge provided by the domain expert before starting the chase; (2) after applying a chase step, update γ with a weight for the generated fact p. The strategies to define such a weight depend on the informed search employed. Indeed, $\gamma(p)$ has to properly consider the evaluation components of the informed search method (e.g., the heuristic h and the function g for A*).

- *Best-First*: in this case, $\gamma(p)$ is the sum of the priorities assigned to *parents*(p) (i.e., the facts where p directly derived from), since best-first only considers the heuristic function h to select the next node n ($f(n) = h(n)$). This way $\gamma(p)$ inherits and combines the heuristic estimation Γ(*parents*(p)).
- *A**: in this case, $\gamma(p)$ is the combination of the heuristic estimation h from the current state to a goal state and the cost g required to reach the current state from the initial state, i.e., the ground facts where p originally derived from. Essentially, such distance is incorporated in the *derivation level* of p (denoted as *depth*(p)) which can be inductively defined as follows: (i) a fact $p \in D$ has *depth*(p) $= 0$; (ii) a fact p derived from a chase step application inherits the maximum derivation level of *parents*(p) $+ 1$. Thus, A* considers $\gamma(p)$ as the difference between the heuristic estimation Γ(*parents*(p)) of the last heuristic chase step application (i.e., $h(n)$) and *depth*(p) (i.e., $g(n)$).

Note that, to practically ensure uniformity of the dimensions considered in the ECHF, the values associated with $\gamma(p)$ are always normalized between 0 and 1.

Heuristic Chase Procedure. Algorithm 1 provides the pseudocode of the chase procedure that employs heuristics chase steps. A description of the algorithm follows.

The algorithm takes as input a database D, a set Σ of Warded TGDs and a BCQ Q. Additionally, it requires the GHF δ and the selected informed search strategy. The output of the algorithm is the answer to Q. First of all, the CHF γ is initialized to δ (line 2), since it only considers extensional facts during the first chase step. Similarly, the chase instance $chase$ that stores the

Algorithm 1 Heuristic Chase Algorithm.

```
 1: function HEURISTIC_CHASE($D$, $\Sigma$, $Q$, $\delta$, search_strategy)
 2:     $\gamma = \delta$                                          ▷ CHF is initialized to GHF
 3:     chase $= D$                                      ▷ chase instance is initialized
 4:     $H =$ init_sorted_structure$(D, \Sigma, \gamma)$        ▷ sorted homomorphisms with weights
 5:     while not $H$.is_empty() do
 6:         $h = H$.poll_first$()$
 7:         $p =$ apply$(\Sigma, h)$           ▷ apply current homomorphism and derive new fact
 8:         if check_isomorphism($p$,chase) then                 ▷ isomorphic facts check
 9:             chase $=$ chase $\cup \{p\}$
10:             if is_positive($Q$, chase) then                       ▷ BCQ answer check
11:                 return true
12:             weight $=$ extract_weight(p, search_strategy)     ▷ derive weight for new fact
13:             $\gamma[p] =$ weight                         ▷ CHF is updated with new weight
14:             $H =$ update_sorted_structure$(\Sigma, p, \gamma)$
15:     return false
```

facts generated during the procedure is initialized to D and corresponds to the initial chase state (line 3). A data structure H is employed to store, at each step of the procedure, all the applicable homomorphisms. Such homomorphisms are sorted according to a weight that is assigned via γ, representing their priority with respect to the heuristic and the search strategy (line 4). While H is not empty, the applicable homomorphism h with the highest weight is extracted (line 6). The result of applying h is a new fact p that also contains chase metadata, such as about its parents and its derivation level (line 7). If p is not isomorphic with a fact already present in $chase$, then it is added to the chase instance (lines 8-9). After such update, the algorithm checks whether the query Q has a positive answer, in which case it terminates (lines 10-11). Otherwise, it derives the weight for p by employing its chase metadata, with respect to the search strategy, and it updates γ accordingly (lines 12-13). Finally, the new applicable homomorphisms, derived from p and Σ, are weighted according to the new γ and are added to H (line 14). If H is empty, that is, all the homomorphisms have been applied, then the answer to Q is negative (line 15).

4. Experimental Evaluation

In this section, we adopt and validate our novel methodology to efficiently solve real-world KG tasks known to be computationally demanding even for state-of-art reasoning techniques.

Financial Use Cases. We work in the context of Central Bank of Italy's company KG [19]. Essentially, it consists of 4.059M companies as nodes, linked by 3.960M *ownership* edges that are labelled with the fraction of shares that a company owns of another company. Specifically, we select two well-known scenarios in the financial domain that, due to the presence of recursion and monotonic aggregations, are extremely time- and space-demanding to solve.

Close Link Problem. This scenario consists in determining whether there exists a (direct or indirect) *link* between two companies, based on a high overlap of shares. Formally, two companies c_1 and c_2 are close links if: (i) c_1 (resp. c_2) owns directly or indirectly, through one or more other companies, 20% or more of the share of c_2 (resp. c_1); or (ii) a third-party owns directly or indirectly, through one or more other companies, 20% or more of the share of c_1 and c_2. Determining whether two companies are closely-linked is extremely important for banking supervision since a company cannot act as a guarantor for loans to another company if they share such a relationship [20]. This problem can be modeled via the following set of recursive Warded Datalog$^{\pm}$ TGDs.

$$Own(c_1, c_2, \mathrm{s}) \rightarrow MCl(c_1, c_2, \mathrm{s}) \tag{1}$$

$$MCl(c_1, c_2, s_1), Own(c_2, c_3, s_2) \rightarrow MCl(c_1, c_3, s_1 \cdot s_2) \tag{2}$$

$$MCl(c_1, c_2, s), \mathrm{ts} = msum(s), \mathrm{ts} \geq 0.2 \rightarrow Cl_1(c_1, c_2) \tag{3}$$

$$Cl_1(c_1, c_2), Cl_1(c_1, c_3), \neg Cl_1(c_2, c_3), c_2 \neq c_3 \rightarrow Cl_2(c_2, c_3) \tag{4}$$

$$Cl_1(c_1, c_2) \rightarrow Cl(c_1, c_2) \tag{5}$$

$$Cl_2(c_1, c_2) \rightarrow Cl(c_1, c_2) \tag{6}$$

Two companies c_1 and c_2, connected by an ownership edge with a share s, are possible close links (rule 1). If c_1 and c_2 are possible close links with a share s_1 and there exists an ownership edge from c_2 to a company c_3 with a share s_2, then also c_1 and c_3 are possible close links with a total share of $s_1 \cdot s_2$ (rule 2). If the sum of all the partial shares s of c_2 owned (directly or indirectly) by c_1 is greater than or equal to 0.2, then c_1 and c_2 are close links (rule 3). The third-party case is modeled according to point (ii) of the definition (rule 4).

Consider, as ontological reasoning task, answering the BCQ Q = $Cl(x,y)$ to check whether two companies x and y are close links.

Company Control Problem. This scenario consists in determining who takes decisions in a company network, that is, who controls the majority of votes for each company [23]. A company c_1 controls a company c_2, if: (i) c_1 directly owns more than 50% of c_2; or, (ii) c_1 controls a set of companies that jointly, and possibly together with c_1 itself, own more than 50% of c_2 [20]. This problem can be modeled via the following set of recursive Warded Datalog$^{\pm}$ TGDs.

$$Own(c_1, c_2, \mathrm{s}), \mathrm{ts} = msum(s) \rightarrow MControl(c_1, c_2, \mathrm{ts}) \tag{1}$$

$$Control(c_1, c_2), Own(c_2, c_3, \mathrm{s}), c_1 \neq c_3, \mathrm{ts} = msum(s) \rightarrow MControl(c_1, c_3, \mathrm{ts}) \tag{2}$$

$$MControl(c_1, c_2, \mathrm{ts}), ts \geq 0.5 \rightarrow Control(c_1, c_2) \tag{3}$$

A company c_1 might control a company c_2 if c_1 owns an amount of shares s of c_2 (rule 1). If c_1 controls c_2 and c_2 owns an amount of shares s of a company c_3, then c_1 might control c_3 (rule 2). A company c_1 controls a company c_2 if the amount of shares that c_1 owns of c_2, directly of indirectly (i.e., through other controlled companies), is greater than or equal to 0.5 (rule 3).

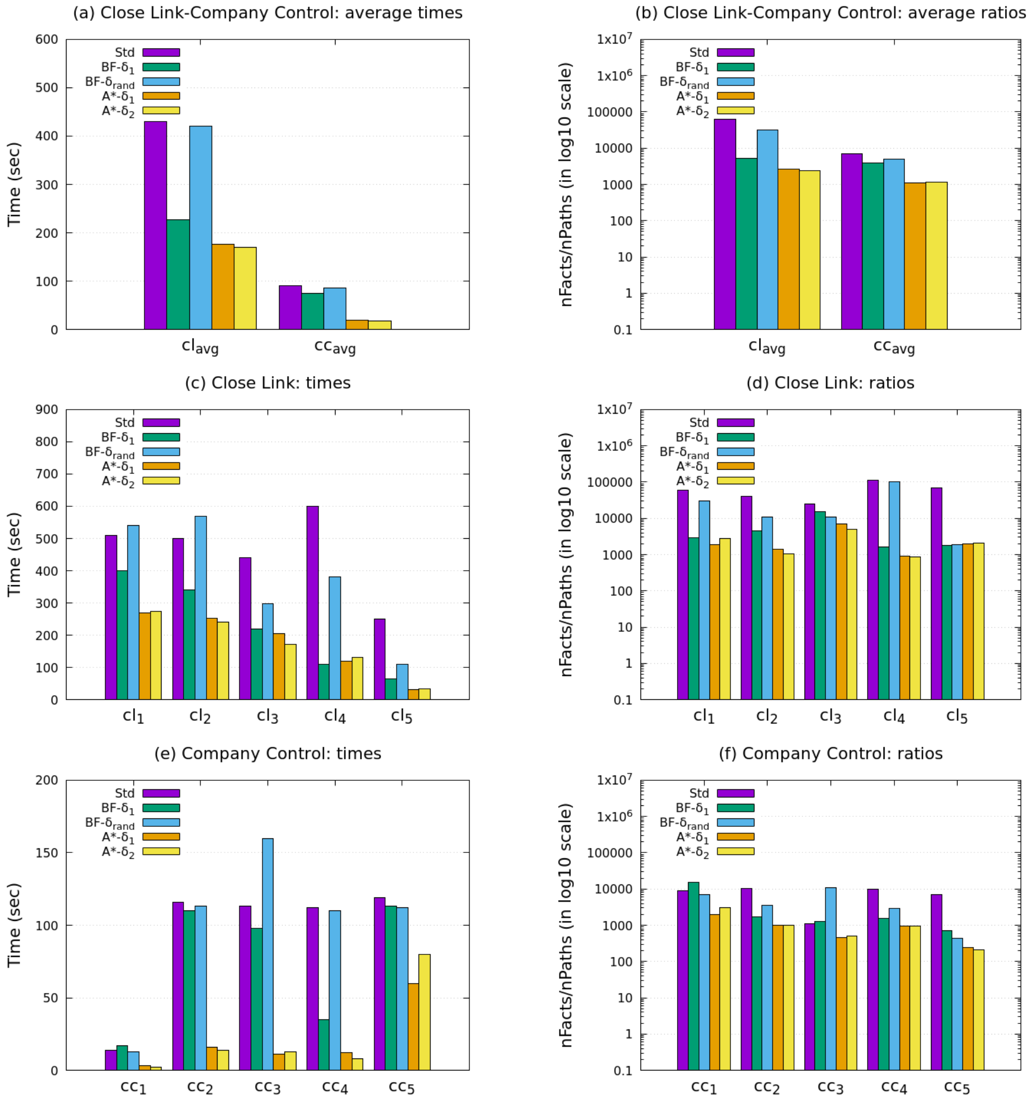
Consider, as ontological reasoning task, answering the BCQ Q = $Control(x,y)$ to check whether a company x controls a company y.

Heuristic-based Reasoning Implementation. We implemented an ad-hoc system that performs our heuristic-based reasoning approach to solve the Close Link and the Company Control problems. From empirical evidence, we observed that prioritizing the exploration of nodes in the KG with a higher *in-degree* allows us to discover a greater number of ownership paths and, consequently, possible close links and control relationships. Thus, we defined the following 3 GHFs mapping the extensional facts: (i) δ_{rand} with random weights; (ii) δ_1 with weights directly proportional to the in-degree of the target nodes; and (iii) δ_2 with weights that consider both the in-degree and the share of the ownership. Based on such heuristics, the system implements: (i) a standard streaming approach (Std), which applies rules according to a *round-robin* strategy and does not employ a priority for the applicable homomorphisms; and (ii) a heuristic-based approach, which applies the heuristic chase step guided by Best-First or A^* with the above GHFs as input (BF-δ_1, BF-δ_{rand}, A^*-δ_1, A^*-δ_2, respectively). The system has as input the set of TGDs for the scenario, the company KG, the BCQ, the GHF and the informed search method to employ, and it provides as output whether pairs of companies are close links or are involved in a control relationship. Moreover, it adopts a streaming architecture with a pull-based pipeline of the facts for rule activation, according to the priority of the corresponding applicable homomorphisms. Such priority is based on a queue, ordered by the ECHF that employs the selected heuristic: each time a new fact is generated, the queue is updated with the resulting set of applicable homomorphisms.

Experiments and Results. We run the experiments on a cloud instance of our system in an AWS EC2 Ubuntu machine with 64 vCPU and 128 GB of RAM. We compared the approaches discussed above to solve Close Link and Company Control over 100 pairs of companies in the company KG, selected randomly among the ones that are known to be close links or in a control relationship. We adopted a time limit of 10 minutes for each BCQ. We collected both the execution times in seconds and *nFacts/nPaths*, i.e., the ratio between the number of generated facts and the number of discovered paths for each pair of companies in the BCQ. Intuitively, it provides an indication of the average number of facts to generate for discovering a new path between the candidates. Figure 2(a) and Figure 2(b) illustrate the average values of the measures over the 100 pairs in $\log_{10}$ scale. Many evaluations with Std did not terminate and we considered 600 seconds as their execution time. Then, we focus on the resulting times and ratios for 5 random pairs of the 100 close links (Figure 2(c) and Figure 2(d), resp.) and as many companies in a control relationship (Figure 2(e) and Figure 2(f), resp.). The experiments confirm the validity of the heuristic-based reasoning we propose, both in terms of the time required to evaluate the BCQs and the ability to prevent the generation of superfluous facts for the answers. Indeed, exploiting fitting heuristics such as δ_1 and δ_2 ensures much better results than the random-based one, which in some instances behaves worse than Std. Combining them with the A^* search proved to be particularly effective, being on average over 3 times faster than Std.

5. Related Work

Current literature includes a variety of optimization techniques that exploit external knowledge to efficiently solve query-answering tasks. For instance, in the database realm, hint operators are employed to statically guide the optimizer in the creation of the query execution plan, by enforcing specific directives (e.g., join order, choice of physical operators for joins, etc.) [24, 25]. Similarly, ad-hoc constraints, modeled via hinting languages [26], can be injected to support the evaluation of the query plan with the lowest estimated cost. In Datalog settings, especially featuring recursion and monotonic aggregations, query answering optimizations employ a


(a) Close Link-Company Control: average times

(b) Close Link-Company Control: average ratios

(c) Close Link: times

(d) Close Link: ratios

(e) Company Control: times

(f) Company Control: ratios

Figure 2: Reasoning statistics for the experimental evaluation.

priori knowledge on: (i) a syntactic level, rewriting the program into a new version that can be evaluated more efficiently [12]; and, (ii) a semantic level, excluding non-relevant derivations in the query execution via eager computations [27]. Moreover, the injection of knowledge also in the form of embeddings has recently been proposed to enable faster reasoning and query-answering over large-scale datasets [28, 29]. Yet, the heuristic-based reasoning we propose is, to the best of our knowledge, the first methodology that manages a BCQ answering task as an informed search problem, guiding the chase via fitting heuristics. Indeed, it enriches our *DHint* approach, which guided rule activation via heuristics only from extensional facts, and extends it by considering the prioritization of all the applicable homomorphisms in the chase [30].

6. Conclusion

Reasoning over large knowledge graphs, especially in the presence of recursive settings, can be very time- and space-demanding even for state-of-the-art Datalog$^{\pm}$-based reasoning systems. However, when answering Boolean Conjunctive Queries, a subset of the generated facts is often superfluous for the result. To tackle this problem, we presented heuristic-based reasoning, a novel reasoning approach that optimizes query evaluation by exploiting external domain knowledge in the form of a heuristic function. We provided experimental evaluation and applied such methodology to efficiently solve computationally demanding scenarios over large KGs in the financial domain. Future work includes extending heuristic-based reasoning with more advanced informed search strategies, as well as adaptive approaches to select the most suitable ground heuristic function depending on the use case and the KG.

Acknowledgments

The work on this paper was partially supported by the Vienna Science and Technology Fund (WWTF) grant VRG18-013.

References

[1] M. Krötzsch, V. Thost, Ontologies for knowledge graphs: Breaking the rules, in: International Semantic Web Conference, Springer, 2016, pp. 376–392.

[2] L. Bellomarini, G. Gottlob, A. Pieris, E. Sallinger, Swift logic for big data and knowledge graphs, in: IJCAI, Springer, 2017, pp. 2–10.

[3] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational constraints, JAIR 48 (2013) 115–174.

[4] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, JoWS 14 (2012) 57–83.

[5] J.-F. Baget, M. Leclère, M.-L. Mugnier, Walking the decidability line for rules with existential variables., KR 10 (2010) 466–476.

[6] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, A. Pieris, Datalog+/-: A family of logical knowledge representation and query languages for new applications, in: 2010 25th annual IEEE symposium on logic in computer science, IEEE, 2010, pp. 228–242.

[7] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, G. Washburn, Design and implementation of the LogicBlox system, in: SIGMOD, 2015, pp. 1371–1382.

[8] P. Barceló, R. Pichler, Datalog in Academia and Industry: Second International Workshop, Datalog 2.0, Vienna, Austria, September 11-13, Proceedings, volume 7494, Springer, 2012.

[9] A. Calì, G. Gottlob, A. Pieris, Towards more expressive ontology languages: The query answering problem, Artificial Intelligence 193 (2012) 87–128.

[10] V. Vianu, Datalog unchained, in: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, 2021, pp. 57–69.

[11] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Transactions on Database Systems (TODS) 4 (1979) 455–469.

[12] Y. R. Wang, M. A. Khamis, H. Q. Ngo, R. Pichler, D. Suciu, Optimizing recursive queries with program synthesis, arXiv preprint arXiv:2202.10390 (2022).

[13] P. Atzeni, L. Bellomarini, D. Benedetto, E. Sallinger, Traversing knowledge graphs with good old (and new) joins., in: KR4L@ ECAI, 2020, pp. 3–9.

[14] S. Abiteboul, R. Hull, V. Vianu, Foundations of databases, volume 8, Addison-Wesley Reading, 1995.

[15] M. Alviano, W. Faber, G. Greco, N. Leone, Magic sets for disjunctive datalog programs, Artificial Intelligence 187 (2012) 156–192.

[16] A. Dsouza, N. Tempelmeier, R. Yu, S. Gottschalk, E. Demidova, Worldkg: A world-scale geographic knowledge graph, in: CIKM, 2021, pp. 4475–4484.

[17] L. Bellomarini, D. Benedetto, G. Gottlob, E. Sallinger, Vadalog: A modern architecture for automated reasoning with large knowledge graphs, Information Systems (2020) 101528.

[18] L. Bellomarini, E. Sallinger, G. Gottlob, The Vadalog System: Datalog-based reasoning for knowledge graphs, VLDB 11 (2018).

[19] P. Atzeni, L. Bellomarini, M. Iezzi, E. Sallinger, A. Vlad, Augmenting logic-based knowledge graphs: The case of company graphs., in: KR4L@ ECAI, 2020, pp. 22–27.

[20] P. Atzeni, L. Bellomarini, M. Iezzi, E. Sallinger, A. Vlad, Weaving enterprise knowledge graphs: The case of company ownership graphs., in: EDBT, 2020, pp. 555–566.

[21] T. Baldazzi, L. Bellomarini, E. Sallinger, P. Atzeni, Eliminating harmful joins in warded datalog+/-, in: International Joint Conference on Rules and Reasoning, Springer, 2021, pp. 267–275.

[22] S. Russell, P. Norvig, Ai a modern approach, Learning 2 (2005) 4.

[23] A. Gulino, S. Ceri, G. Gottlob, E. Sallinger, L. Bellomarini, Distributed company control in company shareholding graphs, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), IEEE, 2021, pp. 2637–2648.

[24] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, T. Neumann, How good are query optimizers, really?, VLDB 9 (2015) 204–215.

[25] S. Chaudhuri, Query optimizers: time to rethink the contract?, in: Proceedings of the 2009 ACM SIGMOD, 2009, pp. 961–968.

[26] N. Bruno, S. Chaudhuri, R. Ramamurthy, Power hints for query optimization, in: 2009 IEEE 25th ICDE, IEEE, 2009, pp. 469–480.

[27] C. Zaniolo, A. Das, J. Gu, Y. Li, M. Li, J. Wang, Developing big-data application as queries: an aggregate-based approach (2021).

[28] A. Vlad, S. Vahdati, M. Nayyeri, L. Bellomarini, E. Sallinger, Towards hybrid logic-based and embedding-based reasoning on financial knowledge graphs., in: EDBT/ICDT Workshops, 2022.

[29] L. Bellomarini, E. Sallinger, S. Vahdati, Reasoning in knowledge graphs: An embeddings spotlight, in: Knowledge Graphs and Big Data Processing, volume 12072 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 87–101.

[30] T. Baldazzi, D. Benedetto, M. Brandetti, A. Vlad, L. Bellomarini, E. Sallinger, Heuristic-based reasoning on financial knowledge graphs., in: EDBT/ICDT Workshops, 2022.