Heuristic-based Reasoning on Financial 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

When reasoning over large knowledge graphs, Datalog+/- languages offer a good trade-off between expressive power and computational complexity. However, in case of considerably large inputs and in the presence of recursive rules, even state-of-the-art reasoners struggle to accomplish reasoning tasks, e.g. answering Boolean Conjuctive Queries (BCQs). To address this problem, we introduce the notion of heuristic-based reasoning, that is, evaluating Datalog+/- rules according to an order of the facts determined via a user-defined heuristic. We show that adopting a fitting heuristic that guides the reasoning process provides an optimized way to answer a BCQ. To achieve this behaviour in practice, we enrich Datalog+/- programs with a context-aware operator, which we name Dynamic Hint Operator. We apply our new methodology to efficiently solve the Close Link problem on the knowledge graph of Italian companies, a relevant financial problem that estimates the risk to grant a specific loan to a company that is backed by collateral issued by another company.

Keywords

Datalog, ontological reasoning, financial knowledge graph, heuristic, artificial intelligence, close link

1. Introduction

The growing need of enriching financial and corporate knowledge and delivering AI-enhanced applications is pushing companies to adopt data-intensive intelligent systems with *reasoning* capabilities. Under the rising paradigm of *Knowledge Graphs* (KGs), modern *reasoners* support the augmentation of extensional data with logical representations of knowledge as ontologies and programs (i.e., sets of facts and rules) [1].

Ontological Reasoning on KGs. As a main requirement, languages for knowledge representation and reasoning must exhibit high expressive power, being able to model complex real domains with full recursion and existential quantification. At the same time, they must achieve low computational complexity, enabling scalability in practice [2]. The Datalog$^{\pm}$ family [3, 4, 5, 6, 7, 8, 9] is one of the commonly adopted families of logic languages (technically, *fragments*) for reasoning on KGs [10]. It covers these requirements, offering a good trade-off between expressive power and complexity. It guarantees decidability and, in some fragments [11], tractability of query answering. Datalog$^{\pm}$ rules are function-free Horn clauses, potentially including existential quantification, i.e., tuple-generating dependencies (TGDs). The semantics of TGDs is usually specified in an operational way via the CHASE procedure [12], an algorithmic tool that takes as input a database D and a set Σ of TGDs, and modifies D by adding new tuples until Σ is satisfied.

Critical Aspects in Query Answering. In relevant real-world application scenarios, navigational capabilities, empowered by recursion in combination with arbitrary joins, are vital for expressing complex reasoning tasks over KGs. At its essence, graph navigation represents the most time and space demanding computation, as it may consider the whole KG as input for an indefinite number of chase steps. Additionally, when answering a *Boolean Conjuctive Query* (BCQ) – i.e., a query where the answer is a boolean value – under a set of (recursive) TGDs, in most of the cases, not all the generated facts actually contribute to the result. As an example, consider the following set of Datalog$^{\pm}$ TGDs.

Example 1. *An example of knowledge graph navigation.*

$$Failure(b), b = BNP \rightarrow Shock(b) \quad (1)$$

$$Shock(b_1), Credit(b_1, b_2) \rightarrow Shock(b_2) \quad (2)$$

This example represents a credit shock propagation scenario. The bank BNP *experiences a shock due to failure (rule* 1*). If* b_1 *is a bank that undergoes a shock and* b_2 *is a bank in a credit exposure constraint with* b_1*, then the shock is propagated to* b_2 *(rule* 2*).*

Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK
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)

Consider, as ontological reasoning task, the BCQ

$$Q = \mathit{Shock}(\text{Barclays}) \wedge \mathit{Shock}(\text{UBS})$$

to check whether *Barclays* and *UBS* may experience a credit shock, and the database instance D = {*Failure*(*BNP*), *Credit*(*Deutsche*, *Barclays*), *Credit*(*MPS*, *Unicredit*), *Credit*(*BNP*, *MPS*), *Credit*(*Barclays*, *UBS*), *Credit*(*BNP*, *Deutsche*)}. Observe that there are many possible ways to activate rule 2, causing the generation of a distinct number of facts that do not actually contribute to the query result. In fact, by activating rule 2 from the facts *Shock*(*BNP*) and *Credit*(*BNP*, *Deutsche*), we generate *Shock*(*Deutsche*). Then, we can use the latter fact and *Credit*(*Deutsche*, *Barclays*) to trigger again rule 2 and produce *Shock*(*Barclays*). Finally, we generate *Shock*(*UBS*) by applying rule 2 over the facts *Credit*(*Barclays*, *UBS*) and *Shock*(*Barclays*). On the other hand, if we activated rule 2 via different facts, additional tuples would be generated that, while not providing any contribution to the result, would affect the performance of the computation (*Shock*(*MPS*), *Shock*(*Unicredit*)).

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 by adopting a tuple-binding prioritization strategy for the evaluation of Datalog$^{\pm}$ rules. Specifically, we employ a user-defined heuristic as a guideline for TGDs activation in the CHASE.

A Financial Use Case. Financial Knowledge Graphs are central objects in corporate economics and are of high importance for central banks, financial authorities and national statistical offices, to solve relevant problems in distinct areas: *banking supervision*, *credit-worthiness evaluation*, *anti-money laundering*, *insurance fraud detection*, *economic and statistical research* and many more. In particular, in the Central Bank of Italy's enterprise KG of Italian companies (ICKG) [13, 14], ownership is a core concept: nodes are companies and persons, and ownership edges are labelled with the fraction of shares that a company or person x owns of a company y. The ICKG consists of 4.059M companies and people as nodes and 3.960M shareholdings as edges.

Producing facts that do not contribute to the BCQ result becomes particularly relevant on such large inputs, especially in the presence of recursive TGDs, since their evaluation might cause a blowup of the unavailing facts. These complications heavily affect the resolution of a high interest scenario in the financial context, that is, the *Close Link* problem. It consists of determining whether there exists a (direct or indirect) *link* between a company x and a company y based on a high overlap of shares, and it can be modeled with a set of recursive TGDs. Solving this problem allows us to estimate the risk to grant a specific loan to a company that is backed by collateral issued by another company [13]. As per European Central Bank's directive [15, 16], a company cannot act as guarantor for a different one if they are close links.

It can be empirically shown that, to determine whether x and y are close links, adopting a heuristic that prioritizes the exploration of nodes with a higher in-degree allows to reach the query result in an efficient fashion and limits the generation of unavailing facts. Motivated by this, we efficiently solve the Close Link problem on the ICKG by introducing a new reasoning methodology that enriches the semantics of TGDs with *context-awareness*. Specifically, we evaluate Datalog$^{\pm}$ rules according to an order of the input facts that is determined by employing the aforementioned heuristic. In detail, the main **contributions** of this paper are:

- A **novel scalable approach** to tackle the BCQ answering problem in recursive settings. We introduce the notion of *heuristic-based reasoning*, that is, evaluating Datalog$^{\pm}$ rules according to an established order of the facts, guided by a user-defined heuristic. Specifically, we enrich Datalog$^{\pm}$ programs with a context-aware operator, which we name *Dynamic Hint Operator* (DHint). This allows the reasoning engine to prioritize specific tuple bindings in the evaluation of a Datalog$^{\pm}$ rule, in order to efficiently answer a specific BCQ.
- A **relevant application** of the Dynamic Hint Operator to solve the Close Link problem on the enterprise KG of Italian companies. In particular, we provide an ad-hoc solution that exploits an empirically-defined heuristic as a guideline for the TGDs activation.
- An **evaluation of the performance** of the DHint-based approaches in recursive settings. In particular, we compare the ad-hoc DHint implementation with state-of-the-art BCQ evaluation techniques for the Close Link problem and we show that a fitting heuristic applied via DHint provides an optimized way to evaluate a BCQ.

Related Work. To outline the related work for this paper, we distinguish the following two perspectives. From a financial point of view, distinct reasoning-based approaches have been adopted to tackle and solve relevant business problems. Among them, we mention the effective application of "Golden Powers" (i.e., the possibility of the central Government to veto specific acquisition transactions) to prevent hostile *company takeovers* [17]. Other use cases exploit rule-based reasoning to counteract *money laundering* processes [18] and to determine whether a company or a person exerts control, through the majority of voting rights, on another company, i.e., the *Company Control* problem [13]. From a technical standpoint, we refer to approaches that exploit external knowledge to efficiently solve query-answering tasks.

Specifically, in the context of DBMSs, there exist multiple alternatives to inject specific criteria that guide the query plan creation. Standard hint operators provide a mechanism that instructs the optimizer on how to determine the execution plan for a certain query, based on specific directives (e.g., join order, choice of physical operators for joins, etc.) [19, 20]. Additionally, ad-hoc techniques to enhance the optimizer in the plan computation have been proposed: for instance, "Phint", a hinting language that introduces specific constraints to guide the optimizer in the evaluation of the most suitable query plan, with the lowest estimated cost [21]. Such approaches act statically in the query evaluation, modifying how the query is structured and how the query plan is created. Alternatively, there exist techniques that exclude the non-relevant derivations in the query execution via an eager computation: such approaches cannot be generally adopted in any query, but they only take place in the presence of monotonic aggregations [22]. Yet, in the context of ontological reasoning, the DHint operator is, to the best of our knowledge, the first technique that enriches the semantics of the TGDs with a heuristic to guide reasoning and optimize BCQ answering.

Overview. This paper is organized as follows. We start by laying out the background notions in Section 2. In Section 3, we define the notion of heuristic-based reasoning and we introduce the DHint operator. In Section 4, we discuss the application of DHint for the Close Link problem and we provide the experimental evaluation. We draw our conclusions in Section 5.

2. Reasoning with TGDs

Let C, N, and V be disjoint countably infinite sets of *constants*, *(labeled) nulls* and (regular) *variables*, respectively. A *(relational) schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. A *term* is a either a constant or 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* (or simply *database*) over $\mathbf{S}$ associates to each relation symbol in $\mathbf{S}$ a relation of the respective arity over the domain of constants and nulls. The members of relations are called *tuples* or, alternatively, *facts*.

Datalog$^{\pm}$ languages generalize standard Datalog rules by introducing existential quantifiers in rule heads. A Datalog$^{\pm}$ program consists of a set of *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 with constants and variables. For brevity, we omit quantifiers and denote the conjunction $\wedge$ by comma.

The Chase Procedure. The CHASE [12] is a fundamental algorithmic tool that modifies a database D to enforce the satisfaction of a set of dependencies Σ over D. It expands D with facts derived via the application of rules $\in \Sigma$ over D, with the *chase steps* into a new database *chase*(D,Σ), possibly containing labelled nulls as placeholders for the existentially quantified objects. Starting from D, the CHASE incrementally applies chase steps and builds new database instances, denoted as I. Given a TGD $\varphi(\bar{x},\bar{y})\rightarrow\exists\bar{z}\ \psi(\bar{x},\bar{z})$, a *TGD chase step* is *applicable* to I if there exists a homomorphism h that maps the atoms of $\varphi(\bar{x},\bar{y})$ to atoms of I (i.e., $h(\varphi(\bar{x},\bar{y})) \subseteq I$) and there does not exist any extension $h' \supseteq h$, such that $h'(\psi(\bar{x},\bar{z}))) \subseteq I$. When the chase step is applicable, the atom $h''(\psi(\bar{x},\bar{z}))) \subseteq I$ is added to I, where h'' is obtained by extending h so that $h''(\bar{z}) \in N$ is a fresh labelled null, for each $z_i \in \bar{z}$.

Boolean Conjunctive Queries (BCQs). An answer to a BCQ is defined via homomorphisms, which are constant-preserving mappings h from variables in $\bar{x}$ and $\bar{y}$ to elements of $C \cup N$, s.t. $\forall\bar{x}\forall\bar{y}\ \varphi(\bar{x},\bar{y}) \subseteq I$ and $h(\bar{x}) = t$ is a tuple of the query answer. The answer to a BCQ $Q \leftarrow \exists\bar{y}\ \psi(\bar{y})$ over an instance I is positive iff there exists a homomorphism h: $\bar{y} \rightarrow C \cup N$ s.t. $h(\varphi(\bar{y})) \subseteq I$.

3. Dynamic Hint Operator

In this section, we discuss the theoretical aspects of our new reasoning methodology. As the goal of this paper is to highlight how our approach improves the performance of a real-world scenario, here we only provide the main notions required to achieve such purpose.

The standard chase step definition, addressed in Section 2, lacks a specific strategy to choose among all the applicable homomorphisms that map the variables of a certain atom. Indeed, such choice impacts on the number of chase steps required to generate a certain fact. Moreover, while the order of the chase steps does not affect correctness of query answering (as the same final instance *chase*(D,Σ) is always produced), in case of BCQs selecting a suitable homomorphism favours the efficiency of the reasoning task, as it allows to produce less facts that do not contribute to answering the query.

Heuristic-based Reasoning. To apply a strategy that selects such favorable homomorphisms, efficiently solving the task at hand, we enrich the program by injecting an external knowledge of the reasoning setting, provided by a domain expert. To better understand this concept, we refer to Example 1. Assume, as external knowledge in a non-realistic scenario, that every Italian bank only grants credit to other Italian banks. Therefore, it is intuitive to observe that, when answering the BCQ Q, not all the applicable homomorphisms from rule 2 are required. Indeed, we only need to consider the ones whose images (tuples) correspond to *Credit* facts that do not involve

Italian banks, as to answer Q in this domain the only contributing facts refer to shock events of non-Italian banks. We represent this external knowledge via a heuristic function, defined as follows.

Definition 1 (Ground Heuristic Function**).** *Consider a database D over a relational schema S and a set of tuples $F = \{x | x \notin D\}$. A Ground Heuristic Function (GHF) $\delta : D \rightarrow \mathbb{R}_{>0} \cup F \rightarrow 0$ is a user-defined function that maps each tuple of D to a weight $w \in \mathbb{R}_{>0}$ and each tuple of F to 0. We define Extended Ground Heuristic Function (EGHF) $\Delta : P(D) \rightarrow \mathbb{R}_{>0} \cup P(F) \rightarrow 0$ as a function that maps a set of tuples $T \subseteq D \cup F$ to a weight $\bar{w}$ such that $\bar{w} = \sum_{\forall t \in T} \delta(t)$, i.e., $\bar{w}$ is the sum of the weights assigned by δ to each $t \in T$.*

Note that, when the input of the GHF is not a ground fact (i.e., a tuple not in D), the output is always 0, as our heuristic function does not consider facts that are generated in the CHASE. This statement also holds for the EGHF, as the contributions of input non-ground facts are irrelevant. We exploit the EGHF as a guideline in the CHASE to determine which homomorphism is the most suitable at each chase step. We define them as follows.

Definition 2 (Heuristic Chase Step Application**).** *Consider a TGD $\sigma : \varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi(\bar{x}, \bar{z})$, an intermediate chase instance I and an EGHF Δ. Let H be the set of possible homomorphisms applicable from σ to I. A Heuristic chase step consists of the application of $h = max[\Delta(h_i(\varphi(\bar{x}, \bar{y})))]_{\forall h_i \in H}$ over I, i.e., h is the homomorphism in H whose images (tuples) maximise Δ.*

Intuitively, the heuristic chase step exploits the knowledge of the domain expert by prioritizing the homomorphism whose images (i.e., tuples) have the highest EGHF value. Based on such revised form of the chase step application, we devise a new reasoning methodology, which we name *heuristic-based reasoning*. It enables the generation of less unavailing facts, with respect to the standard CHASE, when answering BCQs in the presence of TGDs, and consequently favours the efficiency of the task resolution while preserving correctness.

Dynamic Hint Operator. To achieve this behaviour in practice, we enrich the Datalog$^{\pm}$ syntax with a rule-level operator. We name it *Dynamic Hint Operator* (DHint), where (i) "dynamic" stands for it working at execution time, and (ii) "hint" recalls the homonymous operator adopted by standard DBMSs. Given a TGD σ, we denote $@\mathtt{hint}_{\sigma}(\Delta)$ as the DHint assigned to σ by applying the EGHF Δ. It represents the fact that the evaluation of σ is performed via heuristic chase steps, using Δ to determine the most suitable homomorphism among the applicable ones. Let us now extend Example 1 with our DHint.

Example 2. *Example 1 enriched with DHint.*

$$Failure(b), b = BNP \rightarrow Shock(b) \quad (1)$$

$$@hint_2(\Delta) Shock(b_1), Credit(b_1, b_2) \rightarrow Shock(b_2) \quad (2)$$

where the EGHF Δ : {Credit(Deutsche, Barclays) $\rightarrow$ 5.0, Credit(MPS, Unicredit) $\rightarrow$ 1.0, Credit (BNP, MPS) $\rightarrow$ 1.0, Credit(Barclays, UBS) $\rightarrow$ 5.0, Credit(BNP, Deutsche) $\rightarrow$ 5.0}. For brevity, we omit the facts not included in the database D, as the corresponding value for Δ is 0.

As apparent from how Δ is defined, the facts in D that refer to Italian banks are mapped to lower values. In fact, we recall that, in our example, Italian banks only grant credit to other Italian banks. As a result, only the *Shock* facts that actually contribute to answering Q are generated, i.e., the ones that are related to non-Italian banks. Therefore, the task is closed in an efficient fashion and the correctness is upheld.

4. Financial Use Case

In this section, we provide an experimental evaluation of our heuristic-based reasoning approach. We apply our novel reasoning methodology to solve the Close Link problem over the ICKG.

The Close Link Problem. Per European Central Bank's directive, a company c_1 cannot act as guarantor for a company c_2 if c_1 and c_2 are close links [15, 16]. Specifically, 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*. Given these settings, the problem can be modeled via the set of recursive TGDs we provide in Example 3.

Example 3. *Close Link modeled with a set of TGDs.*

$$Own(c_1, c_2, s) \rightarrow MCl(c_1, c_2, s) \quad (1)$$

$$MCl(c_1, c_2, s_1), Own(c_2, c_3, s_2) \rightarrow MCl(c_1, c_3, s_1 \cdot s_2) \quad (2)$$

$$MCl(c_1, c_2, s), ts = sum(s), ts \geq 0.2 \rightarrow Cl_1(c_1, c_2) \quad (3)$$

$$Cl_1(c_1, c_2), Cl_1(c_1, c_3), c_2 \neq c_3 \rightarrow Cl_2(c_2, c_3) \quad (4)$$

$$Cl_1(c_1, c_2) \rightarrow Cl(c_1, c_2) \quad (5)$$

$$Cl_2(c_1, c_2) \rightarrow Cl(c_1, c_2) \quad (6)$$

This example represents the Close Link scenario with reference to the ICKG. A company c_1 and a company c_2, connected by an ownership edge with a share s, can be close links (rule 1). If two companies c_1 and c_2 can be 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 can be close links with a total share of $s_1 \cdot s_2$ (rule 2). Every partial share s of c_2 owned (directly or indirectly)

by c_1 is summed to compute the total share that c_1 owns of c_2; as stated by (i) of the close link definition, if the total share is greater or equal than 0.2, then c_1 and c_2 are close links (rule 3). The third-party case is modeled according to (ii) of the definition (rule 4).

Limits of BCQ Evaluation Techniques. Consider the BCQ Q = *Cl*(x,y) and the TGDs of Example 3 to check whether the companies x and y are close links. When we consider a high number of input *Ownership* edges, as in the case of ICKG, even state-of-the-art query evaluation techniques struggle to efficiently answer Q. Modern reasoners encode CHASE-based procedures by adopting two main approaches: (i) a *materialization* technique that consists of producing and storing all the facts for each predicate by adopting the so-called *semi-naive* evaluation [23]. In this case, the rules are evaluated according to bottom-up strategies, starting from the initial database and repeatedly applying the rules until a fixpoint is reached; (ii) a *streaming* technique that adopts reasoning query graphs [24], where nodes correspond to relational algebra operators (select, project and join) and edges are dependency connections between the rules. Such graph forms an active pipeline and the data flows through its nodes, each receiving input data from the previous nodes and performing the required transformations. Both these approaches may cause the creation of facts that do not contribute to answering the BCQ, as the fact generation process is independent from the query itself. For instance, the materialization approach creates all the facts corresponding to a predicate, which may be unavailing to answer the BCQ. On the other hand, in the streaming approach the sequence of the generated facts depends on the order in which the input facts flow in the pipeline.

A Heuristic for Close Links. We empirically observed that prioritizing the exploration of nodes with a higher in-degree allows us to find a greater number of ownership paths between the close link candidates, compared to standard graph visiting techniques. Motivated by how the limitations of BCQ evaluation techniques affect the detection of close links between companies on the ICKG [13], we exploit such intuition and build an ad-hoc EGHF Δ to assign a weight to the ownership edges in input that is directly proportional to the in-degree of the target nodes. With reference to Example 3, we enrich rule 2 with our DHint operator with such Δ as follows:

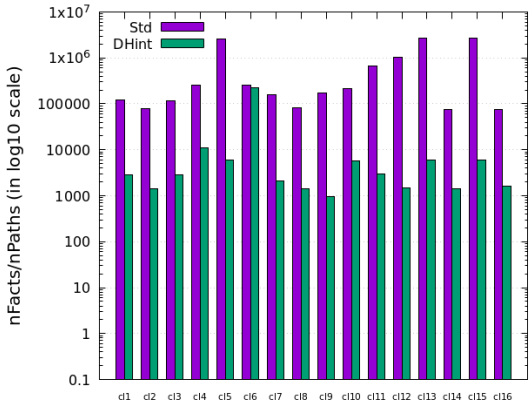
$$\texttt{@hint}_2(\Delta) \quad \mathit{MCl}(c_1, c_2, s_1), \mathit{Own}(c_2, c_3, s_2) \rightarrow \mathit{MCl}(c_1, c_3, s_1 \cdot s_2)$$

Intuitively, rule 2 represents a graph navigation guided by the EGHF Δ, where the edges whose target nodes have a higher in-degree are prioritized.

DHint Implementation. We implemented the Close Link Solver (CLS), an ad-hoc system that allows us to answer Q under the TGDs of Example 3, i.e., it is able to determine whether two companies are close links. CLS adopts a streaming architecture such that each iteration produces a new tuple representing a path between two companies c_1 and c_2 with weight w. Such w corresponds to the partial share that c_1 owns of c_2 and it is calculated as the product of all the shares of the ownership edges in the traversed path. Whenever the new tuple represents a path between the companies x and y involved in the BCQ Q, the system checks whether the sum of the partial shares of y owned (directly or indirectly) by x exceeds 0.2, in which case x and y are close links and the task terminates successfully.
We integrated CLS with two distinct approaches:

- a *Standard* one (Std), which essentially consists in a classic *transitive closure* evaluation, which adopts a *nested-loop join* and a FIFO strategy to guide the generation of the facts;
- a *DHint-based* one (DHint), which employs the EGHF Δ defined above to order the applicable homomorphisms. As a result, it prioritizes the generation of the tuple whose *provenance* (namely, the facts it derives from) maximises Δ.

Experiments and Results. We compared the two approaches over 100 pairs of companies in the ICKG. We selected them randomly among a set of companies that are known to be close links. We run our system on a cloud instance of CLS in a running Ubuntu Standard E8s V4 Linux machine, with 8 virtual v4 cores, 64 GB of RAM and 64 GB SSD. For 84 pairs of companies, both approaches terminated in less than two minutes. For the remaining 16 pairs, we extended the execution time up to 10 minutes and we ran our system again with both approaches. We collected the number of facts generated and the number of paths found between the close link candidates. As a metric of comparison, we adopted the ratio between the total number of facts generated and the number of paths between the close link candidates discovered by the two approaches: we name this metric *nFacts/nPaths ratio*. Intuitively, this ratio represents the average number of facts to generate in order to find a new ownership path between the close link candidates. Figure 2 illustrates that, for all the 16 candidate pairs, the DHint approach discovers a much higher number of paths than the Standard one in the same time. Moreover, our heuristic-based approach allows us to prevent the generation of many unavailing facts. Indeed, a considerably smaller number of facts are generated in order to achieve an answer to the query, i.e., to determine whether the candidates are close links. Hence, our novel approach achieves an inferior nFacts/nPaths ratio, as shown in Figure 1.



(a) Ratio comparison on log10 scale.

cl pairs	Std	DHint
cl1	119130	2877
cl2	80320	1413
cl3	116008	2800
cl4	259711	11110
cl5	2536042	6050
cl6	253600	220704
cl7	157786	2112
cl8	81811	1414
cl9	174043	954
cl10	212584	5614
cl11	669927	2944
cl12	1039643	1458
cl13	2730736	6072
cl14	75874	1420
cl15	2730916	6051
cl16	75054	1650

(b) Actual ratio values.

Figure 1: Comparison between the nFacts/nPaths ratios of Std and DHint approaches on 16 close links candidates.

	cl1	*cl2*	*cl3*	*cl4*	*cl5*	*cl6*	*cl7*	*cl8*
Std Facts	35500777	35501852	35498623	535502578	35504600	35503661	35502031	35506112
Std Paths	298	442	306	1367	14	140	225	434
DHint Facts	18029520	18052525	18007667	2066622	18007733	1544932	18071984	18061181
DHint Paths	6432	12775	6432	186	2976	7	8556	12765

	cl9	*cl10*	*cl11*	*cl12*	*cl13*	*cl14*	*cl15*	*cl16*
Std Facts	35504837	35501582	35506138	35503290	35499577	35509277	35501917	35500872
Std Paths	204	167	53	204	13	468	13	473
DHint Facts	18978699	18043902	18091683	18088721	18041516	17972972	18052265	1023257
DHint Paths	19885	3214	6144	12405	2971	12653	2983	620

Figure 2: Num. of facts produced and number of paths discovered with Std and DHint approaches on 16 close links candidates.

5. Conclusion

Reasoning over large knowledge graphs, especially in the presence of recursive settings, can be extremely demanding even for state-of-the-art reasoners. However, when answering Boolean Conjuctive Queries (BCQs), not all the generated facts actually contribute to the result. For this reason, we introduced the heuristic-based reasoning, a novel reasoning approach that injects external domain knowledge, in the form of a heuristic function, into the program to prevent the generation of such unavailing facts and solve the task at hand in an efficient fashion. To achieve this behaviour in practice, we presented the Dynamic Hint Operator, a context-aware operator that guides the reasoning process to optimize BCQ answering according to such heuristic. We employed our heuristic-based method to efficiently solve the Close Link problem on the ICKG and we laid the foundations for a domain-aware approach to Datalog-based reasoning.

References

[1] M. Krötzsch, V. Thost, Ontologies for knowledge graphs: Breaking the rules, in: International Semantic Web Conference (1), volume 9981 of *LNCS*, 2016, pp. 376–392.

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

[3] 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.

[4] P. Barceló, R. Pichler (Eds.), Datalog in Academia and Industry - Second International Workshop, Datalog 2.0, Vienna, Austria, September 11-13, 2012. Proceedings, volume 7494 of *LNCS*, Springer, 2012.

[5] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational

constraints, J. Artif. Intell. Res. 48 (2013) 115–174.
[6] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, J. Web Sem. 14 (2012) 57–83.
[7] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, A. Pieris, Datalog+/-: A family of logical knowledge representation and query languages for new applications, in: LICS, 2010, pp. 228–242.
[8] A. Calì, G. Gottlob, A. Pieris, Towards more expressive ontology languages: The query answering problem, Artificial Intelligence 193 (2012) 87–128.
[9] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, in: Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 77–86. URL: https://doi.org/10.1145/1559795.1559809. doi:10.1145/1559795.1559809.
[10] 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 LICS, IEEE, 2010, pp. 228–242.
[11] G. Gottlob, A. Pieris, E. Sallinger, Vadalog: recent advances and applications, in: JELIA, Springer, 2019, pp. 21–37.
[12] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Transactions on Database Systems 4 (1979) 455–468.
[13] 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.
[14] P. Atzeni, L. Bellomarini, M. Iezzi, E. Sallinger, A. Vlad, Augmenting logic-based knowledge graphs: The case of company graphs (2021).
[15] GUIDELINE (EU) 2011/14 OF THE ECB, https://www.ecb.europa.eu/ecb/legal/pdf/l_33120111214en000100951.pdf, 2014. [Online; accessed 14-Gen-2022].
[16] GUIDELINE (EU) 2018/570 OF THE ECB, https://www.ecb.europa.eu/ecb/legal/pdf/celex_32018o0003_en_txt.pdf, 2018. [Online; accessed 14-Gen-2022].
[17] L. Bellomarini, M. Benedetti, A. Gentili, R. Laurendi, D. Magnanimi, A. Muci, E. Sallinger, Covid-19 and company knowledge graphs: assessing golden powers and economic impact of selective lockdown via ai reasoning, arXiv preprint arXiv:2004.10119 (2020).
[18] L. Bellomarini, E. Laurenza, E. Sallinger, Rule-based anti-money laundering in financial intelligence units: experience and vision, in: RuleML+ RR (Supplement), 2020.
[19] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, T. Neumann, How good are query optimizers, really?, Proceedings of the VLDB Endowment 9 (2015) 204–215.
[20] S. Chaudhuri, Query optimizers: time to rethink the contract?, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, 2009, pp. 961–968.
[21] N. Bruno, S. Chaudhuri, R. Ramamurthy, Power hints for query optimization, in: 2009 IEEE 25th International Conference on Data Engineering, IEEE, 2009, pp. 469–480.
[22] C. Zaniolo, A. Das, J. Gu, Y. Li, M. Li, J. Wang, Developing big-data application as queries: an aggregate-based approach (2021).
[23] S. Abiteboul, R. Hull, V. Vianu, Foundations of databases, volume 8, Addison-Wesley Reading, 1995.
[24] G. Graefe, W. J. McKenna, The volcano optimizer generator: Extensibility and efficient search, in: Proceedings of IEEE 9th International Conference on Data Engineering, IEEE, 1993, pp. 209–218.