Computing Dual Simulation for Graph Database Queries Technical Report

Stephan Mennicke Jan-Christoph Kalo
Denis Nagel Hermann Kroll Wolf-Tilo Balke
Institut für Informationssysteme, TU Braunschweig, Braunschweig, Germany
{mennicke,kalo,kroll,balke}@ifis.cs.tu-bs.de denis.nagel@tu-bs.de

1 Introduction

Extensive knowledge graphs form a commonplace backbone in today's information infrastructures. Therefore, scalable query processing in graph databases has sparked a vivid interest in the database community. Already at an early stage, specialized graph query languages have been designed, such as SPARQL, the W3C recommendation for querying RDF data by SQL-like formulations [29]. Such languages provide easy to use, yet expressive query capabilities on graph structures, but need to severely break down structural complexity to allow for fast query evaluation. Indeed, the evaluation of complex graph patterns is computationally expensive and thus, a variety of implementational avenues have been proposed [9, 25, 5].

For illustration, let us look at a sample query. At the heart of SPARQL, basic graph patterns (BGPs) form the syntactically least complex queries. BGPs are graphs and their result sets contain all graph-homomorphic matches from the graph database instance. Consider query ($\mathscr{X}_1$) retrieving all persons (cf. variable `?director`), who directed some movie (`?movie`) and worked with another person (`?coworker`),

```
SELECT * WHERE {
  ?director directed ?movie .              (X1)
  ?director worked_with ?coworker . }
```

($\mathscr{X}_1$) consists of two triple patterns, t_1 asking for a `directed` link between variables `?director` and `?movie` and t_2 for a `worked_with` relationship between `?director` and `?coworker`. Evaluating ($\mathscr{X}_1$) w. r. t. the example database instance, depicted in Fig. 1(a), retrieves the two subgraphs in bold print, having nodes `B. De Palma` or `G. Hamilton` assigned to variable `?director`.

Besides full-fledged graph query languages, simple graph pattern matching for diverse querying tasks enjoys a similar interest in the database community [8, 13, 12, 10, 17, 20, 11, 24, 31]. Many of these applications emply *simulation graph pattern matching*, showing advantages over NP-complete subgraph isomorphism, not only in computation time. An in-depth analysis of the approaches incorporating forms of simulation [8, 20, 24, 31] reveals two shortcomings.

(1) Although not the main focus of these papers, when it comes to performance evaluation of the simulation algorithms, they only compare to state-of-the-art subgraph isomorphism algorithms. These algorithms are not specifically designed for graph database querying tasks, in the same way as state-of-the-art graph database management systems like Virtuoso [9]. Since all of the desired applications are database applications, it would only be fair to let their algorithms run against database systems. Isomorphism queries can be restated as SPARQL queries with conjunction and filter conditions [21]. The performance evaluations of graph pattern matching papers show good evaluation times but we

Technical Report
Online available since June 2018
Created on June 2, 2018

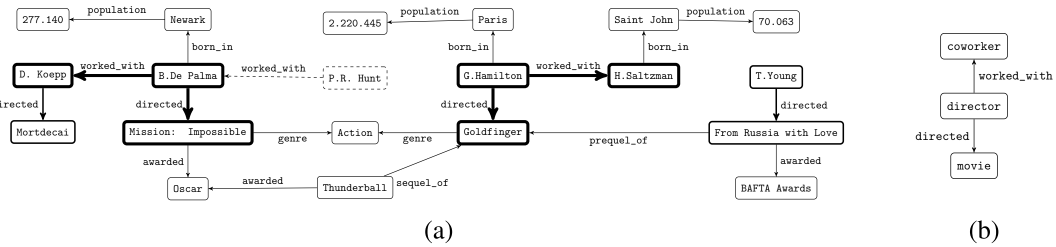


Figure 1: Representation of (a) an Example Graph Database and (b) a Graph Pattern for ($\mathscr{X}_1$)

have reason to believe that Virtuoso and other graph database systems would perform better (cf. Table 1). Therefore, we have to find out whether or not we can algorithmically catch up with graph database systems, since simulation queries may not as easily formulated as SPARQL queries [21].

(2) What all of the classical graph pattern matching problems have in common is the input given as a graph. There is no possibility of building more complex patterns as in graph query languages. Also therefore, we have to study whether there are principle boundaries of an incorporation of graph query operators into the pattern matching process.

Towards (1), we investigate dual simulation, a version of simulation specifically developed for the graph data setting [20]. The algorithm developed by Ma et al. follows a single passive strategy that checks whether the definition of dual simulation is satisfied, yielding a huge amount of iterations needed to compute dual simulations. The duality in dual simulation allows for an active computation (cf. Lemma 1) that finds many contradictions to the definition in one iteration. We develop a novel and more flexible algorithmic solution to the dual simulation problem by the fixpoint of a system of inequalities (SOI), allowing for *fast dual simulation processing* in the graph query setting. We give formal proof of the correctness of our algorithm as well as experimental justification for the performance improvements due to our solution.

Regarding (2), our contribution also is a conservative extension of dual simulation to work with typical graph query operations, exemplarily taken from SPARQL. Thereby, we obtain an overapproximation of the actual SPARQL query results for further inspection, filtering, or actual query processing, depending on the specific application. These extensions are sound in that none of the matches due to the SPARQL semantics is neglected by dual simulation, i. e., it is safe to use the result for further query processing. Our algorithmic framework remains efficient, since the additions we have to make are directly implementable in the SOI solution and do not influence the overall quadratic-time complexity.

In Sect. 5, we perform extensive experiments on two real-world large-scale databases. First, we provide evidence of the runtime improvements due to our solution over the algorithm by Ma et al.. Second, we step into one possible application, namely per-query database pruning. By dual simulation processing, more than 95% irrelevant triples are disqualified for all evaluated queries, being the reason for improved query evaluation times with two state-of-the-art graph databases Virtuoso [9] and RDFox [25]. Further, we observe that our dual simulation may directly be incorporating as a pruning preprocessing step in RDFox. In Sect. 6, we elaborate on related work.

2 Graphs, Data and Matching

By *graphs* we refer to edge-labeled directed graphs with a finite set of nodes V, a finite (label) alphabet Σ, and a directed labeled edge relation $E \subseteq V \times \Sigma \times V$. A *graph* is a triple $G = (V, \Sigma, E)$ of the aforemen-

Figure 2: Two Graph Patterns

tioned components. As exemplified in Fig. 1, nodes will be depicted as rounded-corner rectangles (with its identifier/name as centered label) while edges are represented by directed arrows (with associated labels next to the arrow) between nodes. We often identify by V_G and E_G the respective components of graph G. We assume all graphs to be labeled over a fixed alphabet Σ. For every label $a \in \Sigma$, we associate with G two adjacency maps, a forwards map $\mathfrak{F}^a_G$ and a backwards map of G, $\mathfrak{B}^a_G$. Both mappings associate a subset of nodes with each node $v \in V_G$, in case of forwards maps, the set of successor nodes, and in case of backwards maps, the set of predecessor nodes of v. Formally, define $\mathfrak{F}^a_G(v) := \{w \mid (v,a,w) \in E_G\}$ and $\mathfrak{B}^a_G(v) := \{u \mid (u,a,v) \in E_G\}$.

In the *Resource Description Framework* (RDF), the basic ingredients are *triples* (s, p, o), describing a relationship (p) between two database resources s and o. By analogy, s, p and o are thought of as *subject*, *predicate* and *object*. Database resources (s or o) stem from two universes, $\mathscr{O}$, the set of all objects, each usually referenced by an IRI (Internationalized Resource Identifier) and $\mathscr{L}$, the set of literals. A literal is an element from an arbitrary data domain, such as the integers, usually to describe attribute values of objects. Predicates are also implemented by IRIs, stemming from the universe $\mathscr{P}$. Simplifying the presentation, we assume all three universes to be disjoint. Furthermore, we abstract from the implementation as IRIs and use intuitive names to identify database objects and predicates (cf. example database in Fig. 1(a)). RDF allows for generalized triples of type $\mathscr{O} \times \mathscr{P} \times (\mathscr{O} \cup \mathscr{L})$, sufficient to formulate interrelations and attributes of objects. Attributes connect an object with a literal, e. g., in Fig. 1(a), the information that `Saint John` has `70,063` inhabitants is reflected by the triple $(\texttt{Saint John}, \texttt{population}, \texttt{70,063})$. Further note that literals may only occur in the third component of a triple.

A *graph database* is a finite instance of all possible triples. We formalize it as a graph with all objects and literals occurring in triples as the set of nodes, and all predicates as the alphabet. Handling literals properly leads to a divergence from our initial model.

Definition 1 (Graph Database) A *graph database* is a graph $DB = (O_{DB}, \Sigma, E_{DB})$ with a finite set of database objects and literals $O_{DB} \subset_{fin} \mathscr{O} \cup \mathscr{L}$, a finite set of properties $\Sigma \subset_{fin} \mathscr{P}$, and a labeled edge relation $E_{DB} \subseteq (O_{DB} \cap \mathscr{O}) \times \Sigma \times O_{DB}$. ■

All notions for graphs carry over to graph databases.

As the last point in this section, we discuss the principles of *dual simulation graph pattern matching* [20]. A dual simulation between two graphs G_1, G_2 is a binary relation $S \subseteq V_{G_1} \times V_{G_2}$ such that for each pair of nodes $(v_1, v_2) \in S$, all incoming and outgoing edges of v_1 are also featured by v_2 and the adjacent nodes of v_1 and v_2 belong to S. For a dual simulation S, $(v_1, v_2) \in S$ means that v_2 dual simulates v_1. As an example, consider the graphs depicted in Fig. 2(a) and (b) as G_1 and G_2. A dual simulation relates the nodes with the same label, e. g., `place` in G_2 dual simulates node `place` in G_1, and both nodes, director1 and director2 in G_1, relate to director in G_2, as in

$$\left\{\begin{array}{l}(\texttt{place},\texttt{place}),(\texttt{director1},\texttt{director}),\\(\texttt{director2},\texttt{director}),(\texttt{movie},\texttt{movie}),\\(\texttt{coworker},\texttt{coworker})\end{array}\right\} \quad (1)$$

For instance, node director2 features two outgoing edges, one labeled born_in to node place, the other labeled directed to movie. Node director in G_2 dual simulates director2, since it has an outgoing edge, labeled born_in to node place, and place in G_2 dual simulates place in G_1. The same argument holds for node movie. By following through the argumentation for every pair of nodes in (1), it can be shown that indeed G_2 dual simulates G_1 by the indicated dual simulation. Observe that a single node, here director, may dual simulate more than one node.

Definition 2 (Dual Simulation [20]) Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1,2)$ be two graphs. A relation $S \subseteq V_1 \times V_2$ is a *dual simulation between* G_1 *and* G_2 iff for each $(v_1, v_2) \in S$,

(i) $(v_1, a, w_1) \in E_1$ implies $\exists w_2 \in V_2 : (v_2, a, w_2) \in E_2$ and $(w_1, w_2) \in S$,

(ii) $(u_1, a, v_1) \in E_1$ implies $\exists u_2 \in V_2 : (u_2, a, v_2) \in E_2$ and $(u_1, u_2) \in S$.

We say that G_2 *dual simulates* G_1 iff there is a dual simulation between G_1 and G_2. ■

Note that the trivial dual simulation $S = \emptyset$ is always allowed by our definition. In a graph query setting, we call G_1 *pattern graph* and G_2 is the *graph database* [20].

Reconsider the introductory example query $(\mathscr{X}_1)$. First notice, the graph in Fig. 2(b) dual simulates the graph representation of $(\mathscr{X}_1)$ in Fig. 1(b). The dual simulation is realized by simply ignoring node place. Hence, not every node of the graph database has to play a role in the dual simulation relation. Furthermore, the graph in Fig. 2(a) neither dual simulates nor is dual simulated by the graph in Fig. 1(b). Regarding the graph database depicted in Fig. 1(a) and the graph representation of $(\mathscr{X}_1)$ in Fig. 1(b), the following dual simulation is particularly interesting for our purposes.

$$\left\{\begin{array}{l}(\texttt{director},\texttt{B. De Palma}),(\texttt{director},\texttt{G. Hamilton}),\\(\texttt{coworker},\texttt{D. Koepp}),(\texttt{coworker},\texttt{H. Saltzman}),\\(\texttt{movie},\texttt{Mission: Impossible}),(\texttt{movie},\texttt{Goldfinger})\end{array}\right\} \quad (2)$$

It comprises exactly the nodes of the two subgraphs from the result set of $(\mathscr{X}_1)$. Instead of considering the full graph database (i. e., Fig. 1(a)), we would ignore all graph database nodes but those mentioned by dual simulation (2). Computing this dual simulation is possible in PTIME [20], as opposed to SPARQL query evaluation being PSPACE-complete [27]. How to perform this computation fast is subject to the next section. We apply dual simulation pattern matching principles to SPARQL query processing in Sect. 4.

3 A Perspective on Dual Simulation

At the end of the last section, we have seen a dual simulation between a graph representation of a SPARQL query (BGP $(\mathscr{X}_1)$) and a graph database (Fig. 1(a)), covering all nodes relevant for computing the result set of $(\mathscr{X}_1)$. By Theorem 1 we show that the existence of such a dual simulation is not coincidental, since every match for SPARQL queries like $(\mathscr{X}_1)$ is contained in a maximal dual simulation. A dual simulation S is maximal iff there is no dual simulation S' such that $S \subset S'$. Fortunately, there is exactly one such maximal dual simulation between any two graphs, the *largest dual simulation*.

Proposition 1 (Proposition 2.1 [20]) *For any two graphs G_1 and G_2, there is a unique largest dual simulation $S_{\max}$ between G_1 and G_2, i. e., for any dual simulation S between G_1 and G_2, $S \subseteq S_{\max}$.*

The proof uses the fact that, whenever we have two dual simulations S_1 and S_2 between the graphs, their union $S_1 \cup S_2$ is a dual simulation [20]. Incorporating dual simulation in graph pattern matching or SPARQL query processing amounts to computing the largest dual simulation between an appropriate representation of the query and the graph database. All graph database nodes captured by the largest dual simulation are relevant for answering the query.

Computing the largest (dual) simulation is the algorithmic basis for solving the graph (dual) simulation problem, that is, given two graphs G_1 and G_2, does G_2 (dual) simulate G_1. To the best of our knowledge, all published algorithms for this task [20, 16] work on the same principles. Starting with the largest possible relation between the two node sets, the algorithms incrementally disqualify pairs of nodes contradicting Def. 2. The procedures are guaranteed to terminate when no more pair of nodes can be disqualified. Although the standard algorithm for computing dual simulations [20] enjoys quadratic run-time in the size of the two graphs, we observe that this algorithm does not take advantage of the database query setting and also only allows for the naive evaluation strategy described above. This inflexibility yields high query running times that would easily be beaten by state-of-the-art query evaluation as performed by Virtuoso (cf. Sect. 5).

Subsequently, we develop a novel solution to computing the largest dual simulation, exploiting the graph database query setting as well as run-time analytics into account. The key to our solution is the reformulation of the algorithm as a system of inequalities which allows for two dynamically interchangeable evaluation strategies. Although the resulting algorithm remains a quadratic-time algorithm, we gain a degree of freedom allowing for a systematic reduction of iterations to eventually reach the largest dual simulation (cf. Sect. 3.3). The price we have to pay for this solution is the extra-space for organizing the system of inequalities, growing constantly in the size of the query. Since queries are usually assumed to be much smaller than the database, this organization overhead is negligible. As we show in Sect. 5, the new procedure shows extremely low computation times, a solid basis for query processing, e. g., for SPARQL. Our solution is engineered in three steps. First, we define a set of inequalities equivalent to the coinductive definition of dual simulation in Def. 2. In Sect. 3.2, we show how to derive a fast implementation based on bit-vectors and bit-matrices. Last, we provide a discussion on optimizations, realized in our software prototype[1].

3.1 Groundwork

This subsection lays out the foundation of our new procedure. Any binary relation $R \subseteq A \times B$, over sets A and B, has a characteristic function $\chi_R : A \to 2^B$ with $\chi_R(a) := \{b \in B \mid (a,b) \in R\}$. For a dual simulation S between two graphs, G_1 and G_2, χ_S associates with each node $v \in V_{G_1}$ a set of dual simulating nodes $\chi_S(v) \subseteq V_{G_2}$. Consider an edge (v,a,w) of G_1 and node $v' \in \chi_S(v)$. Taking now χ_S into account, if S is a dual simulation, then

$$\exists w' : (v',a,w') \in E_2 \text{ and } w' \in \chi_S(w) \tag{3}$$

holds. The problem with (3) is that there may be many w' qualifying for $(v',a,w') \in E_2$ but $w' \notin \chi_S(w)$. We pursue to have a single operation allowing us to quickly verify the existence of w'. Therefore, recall that for any graph, here the graph database G_2, we have a forwards adjacency map $\mathfrak{F}^a_{G_2}$ for each label $a \in \Sigma$ (cf. Sect. 2). Exploiting these maps, we prove existence of a w' in (3) simply by intersecting the

[1] Our *OpenSource* tool is available at GitHub `https://github.com/ifis-tu-bs/sparqlSim`

row of v' in $\mathfrak{F}^a_{G_2}$ and the nodes simulating w, i. e.,

$$\mathfrak{F}^a_{G_2}(v') \cap \chi_S(w) \neq \emptyset. \tag{4}$$

(4) still only checks for one pair of nodes. Combining this equation for all $v' \in \chi_S(v)$ yields

$$\textstyle\bigwedge_{v' \in \chi_S(v)} \quad \mathfrak{F}^a_{G_2}(v') \cap \chi_S(w) \neq \emptyset. \tag{5}$$

The same encoding applies to Def. 2(ii), this time using the backwards map,

$$\textstyle\bigwedge_{w' \in \chi_S(w)} \quad \mathfrak{B}^a_{G_2}(w') \cap \chi_S(v) \neq \emptyset. \tag{6}$$

Combining both equations, (5) and (6), yields two inequalities equivalent to the definition of dual simulation and the key for our efficient implementation.

Lemma 1 *Let $G_1 = (V_1, \Sigma, E_1)$ and $G_2 = (V_2, \Sigma, E_2)$ be graphs with $(v, a, w) \in E_1$. For a binary relation $S \subseteq V_1 \times V_2$ satisfying* (5) *and* (6)*, it holds that*

$$\begin{array}{rrcl} (i) & \chi_S(w) & \subseteq & \bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v') \qquad \textit{and} \\ (ii) & \chi_S(v) & \subseteq & \bigcup_{w' \in \chi_S(w)} \mathfrak{B}^a_{G_2}(w') \end{array} \tag{7}$$

are satisfied.

PROOF: W. l. o. g., we show inequality (i) only. Inequality (ii) is completely analogous. Towards a contradiction, assume $\chi_S(w) \not\subseteq \bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v')$. Hence, there is a $w' \in \chi_S(w)$ such that for each $v' \in \chi_S(v)$, $w' \notin \mathfrak{F}^a_{G_2}(v')$, i. e., $(v', a, w') \notin E_2$. This also means that $\chi_S(v)$ and $\mathfrak{B}^a_{G_2}(w')$ are disjoint for each $v' \in \chi_S(v)$, contradicting our assumption that (6) holds. Therefore, such a w' cannot exist, allowing to conclude that $\chi_S(w) \subseteq \bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v')$. □

Phrased differently, dual simulations S satisfy (7) for every edge (v, a, w) of G_1. Please note that by Lemma 1 we have revealed an important observation, that, to the best of our knowledge, has not been published so far. The reason why (7) holds is that part (ii) prevents part (i) from getting ill-formed and vice versa. The fast algorithm we obtain here is a consequence of the duality in dual simulation. Conversely, every solution to (7) is a dual simulation.

Proposition 2 *Let G_1 and G_2 be graphs. $S \subseteq V_1 \times V_2$ is a dual simulation between G_1 and G_2 iff for every edge $(v, a, w) \in E_1$,* (7) *holds for S.*

PROOF: The implication, i. e., a dual simulation S satisfies (7), is analogous to the proof of Lemma 1. Therefore, assume that one of the inequalities is not satisfied and conclude the assumption that S is a dual simulation is violated.

Conversely, assume we have $S \subseteq V_1 \times V_2$ such that (7) holds for every $(v, a, w) \in E_1$. We need to show that S is a dual simulation. Let $(v, v') \in S$, i. e., $v' \in \chi_S(v)$, and $(v, a, w) \in E_1$. We need to show that there is a w' such that $(v', a, w') \in E_2$ and $(w, w') \in S$. From (7)(ii) we get that for some $w' \in \chi_S(w)$ we have that $v' \in \mathfrak{B}^a_{G_2}(w')$. This w' completes the proof, since (1) from $v' \in \mathfrak{B}^a_{G_2}(w')$ follows $(v', a, w') \in E_2$ and (2) from $w' \in \chi_S(w)$, we get that $(w, w') \in S$. Case $(u, a, v) \in E_1$ is completely analogous. □

Hence, (7) characterizes dual simulations and we may use it to compute the largest dual simulation. The algorithm works as follows. We begin with $S_0 := V_1 \times V_2$. For each edge of G_1, check whether or not (7) is satisfied by S_0. Assume (7)(i) fails for an edge (v, a, w). Then S_1 is computed by $\chi_{S_1}(u) := \chi_{S_0}(u)$ for $u \neq w$ and $\chi_{S_1}(w) := \chi_{S_0}(w) \cap \bigcup_{v' \in \chi_{S_0}(v)} \mathfrak{F}^a_{G_2}(v')$. We get rid of all non-simulating nodes of w relative to

S_0 in a single iteration. This procedure is repeated for $S_1, S_2, \ldots$ until we reach an S_k satisfying (7) for every edge of G_1.

Even though we maintain the PTIME nature of the original algorithms, we still lack a way to quickly compute $\bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v')$ and access $\chi_S(v)$. The forthcoming implementation, therefore, works with bit-representations of $\chi_S(v)$ and $\mathfrak{F}^a_{G_2}, \mathfrak{B}^a_{G_2}$, paving the way for optimization in time- and space-consumption (e. g., [5]). In that setting, we derive a *system of inequalities* (SOI) from Prop. 2, for which dual simulations S serve as valid assignments.

3.2 Engineering

The goal of this subsection is to describe the facilities achieving a fast implementation of dual simulation pruning. Recall that we need to compute the largest dual simulation and we do this by a *system of inequalities* according to (7). The challenge is to find a way to quickly compute the unions

$$\bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v') \quad \text{and} \quad \bigcup_{w' \in \chi_S(w)} \mathfrak{B}^a_{G_2}(w'). \tag{8}$$

Combinations of vectors and matrices, especially encoding information only bit-wise, promise fast computations. Hence, we interpret the adjacency maps of G_2 as adjacency bit matrices. Reconsider the graph in Fig. 2(a). For label `born_in`, this graph provides two adjacency matrices,

$$\mathfrak{F}^{\texttt{born\_in}}_{\textit{Fig. 2(a)}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathfrak{B}^{\texttt{born\_in}}_{\textit{Fig. 2(a)}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

We assume here the set of nodes of graphs to be ordered by some pre-defined index, in this example, $\{v_1, v_2, v_3, v_4, v_5\}$ with

$$v_1 = \texttt{place}, v_2 = \texttt{director1}, v_3 = \texttt{director2}, v_4 = \texttt{coworker}, v_5 = \texttt{movie}.$$

Also χ_S can be seen as a matrix with $k = |V_{G_1}|$ rows, one for each node of pattern graph G_1, and $n = |V_{G_2}|$ columns. Specifically for a dual simulation S, a 1 in position (i, j) means that the i^{th} node of the pattern graph is simulated by the j^{th} node of the graph database. Note that for ease of presentation, the pattern graph G_1 does not have an indexed node set. Consequently, for node v of the pattern graph and $j \leq n$, we access the j^{th} component of v's row by $\chi_S(v, j)$. By $\chi_S(v)$ we get v's row vector sliced from matrix χ_S. The desired unions (8) are now achieved by bit-matrix multiplications (symbol $\times_b$)[2],

$$\chi_S(v) \times_b \mathfrak{F}^a_{G_2} \quad \text{and} \quad \chi_S(w) \times_b \mathfrak{B}^a_{G_2}. \tag{9}$$

Here, the result is the effect a-labeled edges have upon nodes v' simulating v. For instance, assume that $\chi_S(\texttt{director}) = \chi_S(\texttt{place}) = (1,1,1,1,1)$. Then for edge $(\texttt{director}, \texttt{born\_in}, \texttt{place})$,

$$\begin{array}{llcl} \chi_S(\texttt{director}) & \times_b \; \mathfrak{F}^{\texttt{born\_in}}_{\textit{Fig. 2(a)}} & = & (1,0,0,0,0) = r_1 \\ \chi_S(\texttt{place}) & \times_b \; \mathfrak{B}^{\texttt{born\_in}}_{\textit{Fig. 2(a)}} & = & (0,1,1,0,0) = r_2. \end{array}$$

Hence, the effect `born_in`-labeled forwards edges have upon any $v' \in \chi_S(\texttt{director})$ reveals that we only reach node `place`. Conversely, by `born_in`-labeled backwards edges we reach `director1` as well as `director2`. The results are used to update a given relation S, according to (7). In the example above,

[2] For vector $\mathfrak{v}$ and matrix $\mathfrak{A}$, $\mathfrak{v} \times_b \mathfrak{A} = \mathfrak{w}$ where $\mathfrak{w}(j) = 1$ iff there is an i such that $\mathfrak{v}(i) = 1$ and $\mathfrak{A}(i, j) = 1$.

the second result tells us that $\chi_S(\texttt{director}) \neq (1,1,1,1,1)$, since the only possible nodes we can reach are director1 and director2. Thus, $\chi_S(\texttt{director}) = (1,1,1,1,1) \not\leq (0,1,1,0,0) = \chi_S(\texttt{place}) \times_b \mathfrak{B}^{\texttt{born\_in}}_{Fig.\ 2(a)}$, but according to Prop. 2, a dual simulation S satisfies (7), now possible to formulate by bit-matrix operations for edges $(v,a,w) \in E_{G_1}$,

$$\begin{array}{rcll} \chi_S(w) & \leq & \chi_S(v) \times_b \mathfrak{F}^a_{G_2} & \text{and} \\ \chi_S(v) & \leq & \chi_S(w) \times_b \mathfrak{B}^a_{G_2}. & \end{array} \tag{10}$$

After observing the wrong value of $\chi_S(\texttt{director})$, we update relation S to S' by $\chi_{S'}(\texttt{director}) := \chi_S(\texttt{director}) \wedge r_2$ (component-wise conjunction of the two vectors). This enables us to give an algorithm for the dual simulation problem between two graphs G_1 and G_2 as a solution of the system of inequalities $\mathscr{E} = (\mathsf{Var}, \mathsf{Eq})$, where every node v of the graph pattern is a variable, i. e., $\mathsf{Var} := V_{G_1}$, and Eq contains for each pattern edge $(v,a,w) \in E_{G_1}$, the following equations:

$$w \leq v \times_b \mathfrak{F}^a_{G_2} \quad \text{and} \quad v \leq w \times_b \mathfrak{B}^a_{G_2}. \tag{11}$$

Fig. 3 shows the SOI for computing dual simulations for the example graphs in Fig. 2(a) and (b). Assignments to the variables, $v,w \in \mathsf{Var}$, are relations $S \subseteq V_{G_1} \times V_{G_2}$. The algorithm computing the largest dual simulation between G_1 and G_2 proceeds as follows.

1. Set $S_0 := V_{G_1} \times V_{G_2}$ and all inequalities in Eq *unstable*.
2. Let S_i be the current relation. Pick any unstable inequality $\varepsilon \in \mathsf{Eq}$
   (a) If S_i is valid for ε, set ε stable and continue with (2).
   (b) If S_i is not valid for $\varepsilon = v \leq w \times_b \mathfrak{A}$ (for $\mathfrak{A} \in \{\mathfrak{F}^a_{G_2}, \mathfrak{B}^a_{G_2} \mid a \in \Sigma\}$), then $\chi_{S_i}(w) \times_b \mathfrak{A} = r$ and $\chi_S(v) \not\leq r$. Update S_i to S_{i+1} such that
   $$\chi_{S_{i+1}}(x) := \begin{cases} S_i(x) \wedge r & \text{if } x = v \text{ and} \\ S_i(x) & \text{otherwise.} \end{cases}$$
   Furthermore, every inequality $y \leq v \times \mathfrak{A} \in \mathsf{Eq}$ are reset to unstable. Mark ε stable and continue with (2).

Please note that the initialization step of S_0 can also be expressed in terms of inequalities, in that for every pattern node v, we add (12) to the set of inequalities Eq.

$$v \leq \underline{\mathbf{1}} \tag{12}$$

Note that $\underline{\mathbf{1}}$ is the vector containing a 1 in every component. The dual simulation given by (1) is the largest solution to the SOI in Fig. 3, thus is the largest dual simulation.

3.3 Optimization and Discussion

Our reformulation of dual simulation and its implementation open up dynamic evaluation strategies for the constructed SOI. First, the order in which the equations are evaluated has an impact on the overall runtime. We have chosen an order that prefers those inequalities, more likely to produce smaller output vectors, i. e., where the adjacency matrix has more empty columns, an indicator for sparsity of the matrix. Second, the computation of r (in step 2b of the algorithm) may be performed row-wise or column-wise. Also here, we follow the strategy of less iterations, i. e., in $v \leq w \times_b \mathfrak{A}$ we choose a row-wise evaluation if and only if $\chi_S(w)$ has fewer bits set than $\chi_S(v)$.

$$
\begin{array}{rcl}
\texttt{place} & \leq & \texttt{director1} \times_b \mathfrak{F}^{\texttt{born\_in}}_{Fig.\ 2(b)} \\
\texttt{place} & \leq & \texttt{director2} \times_b \mathfrak{F}^{\texttt{born\_in}}_{Fig.\ 2(b)} \\
\texttt{director1} & \leq & \texttt{place} \times_b \mathfrak{B}^{\texttt{born\_in}}_{Fig.\ 2(b)} \\
\texttt{director2} & \leq & \texttt{place} \times_b \mathfrak{B}^{\texttt{born\_in}}_{Fig.\ 2(b)} \\
\texttt{coworker} & \leq & \texttt{director1} \times_b \mathfrak{F}^{\texttt{worked\_with}}_{Fig.\ 2(b)} \\
\texttt{director1} & \leq & \texttt{coworker} \times_b \mathfrak{B}^{\texttt{worked\_with}}_{Fig.\ 2(b)} \\
\texttt{movie} & \leq & \texttt{director2} \times_b \mathfrak{F}^{\texttt{directed}}_{Fig.\ 2(b)} \\
\texttt{director2} & \leq & \texttt{movie} \times_b \mathfrak{B}^{\texttt{directed}}_{Fig.\ 2(b)}
\end{array}
$$

Figure 3: System of Inequalities Characterizing Largest Dual Simulation between Fig. 2(a) and (b)

An immediate optimization is given by altering the initial relation S_0, syntactically exploiting that for a variable/node v in G_1, candidate nodes are only those supporting incident edges of v. Therefore, let us denote by $\mathfrak{f}^a_{G_2}$ the bit-vector that summarizes the rows of $\mathfrak{F}^a_{G_2}$ in that $\mathfrak{f}^a_{G_2}(i) = 1$ if there is a j with $\mathfrak{F}^a_{G_2}(i,j) = 1$, and $\mathfrak{f}^a_{G_2}(i) = 0$ otherwise. In the same lines, $\mathfrak{b}^a_{G_2}$ is defined as the summary of $\mathfrak{B}^a_{G_2}$. Then for each variable/node v in G_1, we replace inequality (12) by

$$v \quad \leq \quad \bigwedge_{(v,a,w)\in E_1} \mathfrak{f}^a_{G_2} \wedge \bigwedge_{(u,a,v)\in E_1} \mathfrak{b}^a_{G_1}. \tag{13}$$

Our prototype implementation is a proof-of-concept. We keep G_2 by its adjacency matrices in memory, which is not necessary since for each instance of G_2, only the matrices of labels occurring in G_1 need to be loaded, at least partially. Combined with the memory-economical implementation by Atre et al. [5], even promised to have significantly improved [4], our implementation may directly be used within the preprocessing step of the *BitMat* tool set. As we show in Sect. 5, our dual simulation processing applied to SPARQL queries yields decent pruning factors significantly improving upon those reported by Atre [4].

Let us now apply our algorithmic dual simulation framework to SPARQL queries.

4 Dual Simulation for SPARQL

Having clarified the foundational and algorithmic aspects of dual simulation graph pattern matching, we may now approach an actual query language, in this paper SPARQL. We choose SPARQL for its high-quality standardization by the W3C [29] and its extensive formal treatment, e. g., [27, 2, 3]. Although SPARQL 1.1 has been around for some time, the fundamental properties of the query language remain the same as for SPARQL 1.0. We are aware of the recent report on the semantic foundation of the Neo4J query language CYPHER [14], and confident about the wider applicability of the forthcoming techniques to this language. For SPARQL's least complex constructs, we canonically obtain a dual simulation processing, respecting all matches any SPARQL query processor would find, in the following subsection. Sections 4.2 and 4.3 discuss SPARQL's extensions by join constructs. In all of the upcoming subsections, we obtain a soundness result stating that the original SPARQL matches are preserved for further filtering and processing.

4.1 Basic Graph Patterns

As for RDF, triples are first-class citizens of SPARQL. For the presentation of the upcoming material, we assume subject and object of a triple $t = (s, p, o)$ to be variables from an infinite domain of variables

$\mathscr{V}$, ranging over by $\mathtt{v}, \mathtt{v}_1, \mathtt{v}_2, \ldots$. We call SPARQL triples also *triple patterns*. A variable $\mathtt{v}_1$ is usually introduced by a leading question mark as $?\mathtt{v}_1$ and every triple closes with period (cf. $(\mathscr{X}_1)$).

Querying a graph database $DB = (O_{DB}, \Sigma, E_{DB})$ yields a set of (partial) mappings from the set of variables to actual database objects. For instance, the single triple pattern $t = (\mathtt{v}_1, \mathtt{population}, \mathtt{v}_2)$ gives rise to a match identifying $\mathtt{v}_1$ with node `Saint Join` and $\mathtt{v}_2$ with the literal `70.063` (cf. Fig. 1(a)). By $vars(t)$ we denote the set of variables occurring in triple t, i. e., $vars(t) = \{\mathtt{v}_1, \mathtt{v}_2\}$ for $t = (\mathtt{v}_1, \mathtt{population}, \mathtt{v}_2)$. A *candidate w. r. t. DB* is a partial function $\mu : \mathscr{V} \to O_{DB}$. The set of variables for which candidate μ is defined is denoted by $dom(\mu)$. A candidate μ is a *match for triple t in DB* iff $dom(\mu) = vars(t)$ and, assuming $t = (\mathtt{v}_1, a, \mathtt{v}_2)$, $(\mu(\mathtt{v}_1), a, \mu(\mathtt{v}_2)) \in E_{DB}$, abbreviated as $\mu(t) \in DB$.

We call sets of triple patterns $\mathbb{G}$ *basic graph patterns* (BGPs). Function *vars* and thereupon the notions of candidates and matches extend to BGPs by $vars(\mathbb{G}) = \bigcup_{t \in \mathbb{G}} vars(t)$ and μ is a candidate/match for $\mathbb{G}$ iff μ is a candidate/match for all triples $t \in \mathbb{G}$. The result set $[\![\mathbb{G}]\!]_{DB}$ for $\mathbb{G}$ w. r. t. DB contains all matches for $\mathbb{G}$. Every BGP $\mathbb{G}$ can be seen as a graph $G(\mathbb{G}) = (V_{\mathbb{G}}, \Sigma, \mathbb{G})$ by taking the set of variables occurring in $\mathbb{G}$ as set of nodes, i. e., $V_{\mathbb{G}} := \{\mathtt{v}, \mathtt{w} \mid (\mathtt{v}, a, \mathtt{w}) \in \mathbb{G}\}$. The graph in Fig. 1(b) represents such a conversion of $(\mathscr{X}_1)$.

For dual simulation processing of a BGP $\mathbb{G}$ w. r. t. DB, we compute the largest dual simulation between $G(\mathbb{G})$ and DB. This procedure is sound in that every match μ to $\mathbb{G}$ is a dual simulation and therefore must be contained in the largest dual simulation.

Lemma 2 *Let DB be a graph database, $\mathbb{G}$ a BGP and $\mu \in [\![\mathbb{G}]\!]_{DB}$. Then μ is a dual simulation between $G(\mathbb{G})$ and DB.*

PROOF: We show that μ is a dual simulation between $G(\mathbb{G})$ and DB. Let $(\mathtt{v}, o) \in \mu$, i. e., $\mu(\mathtt{v}) = o$, and let $t \in \mathbb{G}$ such that $\mathtt{v} \in vars(t)$. There are two cases to distinguish, for some $a \in \Sigma$ and $\mathtt{w}, \mathtt{u} \in vars(\mathbb{G})$, (a) $t = (\mathtt{v}, a, \mathtt{w})$ and (b) $t = (\mathtt{u}, a, \mathtt{v})$. Since case (b) is completely analogous, we consider only (a). As μ is a a match for $\mathbb{G}$, it is a match for t, i. e., there is exactly one $o' = \mu(\mathtt{w})$ and $(o, a, o') \in E_{DB}$. Hence, o' meets the requirements of Def. 2(i). □

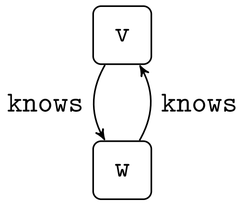
The nodes disqualified by the largest dual simulation are irrelevant for any further query processing obeying the original SPARQL semantics.

Theorem 1 *Let DB be a graph database, $\mathbb{G}$ a BGP and S be the largest dual simulation between $G(\mathbb{G})$ and DB. For each database node $o \in O_{DB}$ such that there are $\mathtt{v} \in vars(\mathbb{G})$ and $\mu \in [\![\mathbb{G}]\!]_{DB}$ with $\mu(\mathtt{v}) = o$, it holds that $(\mathtt{v}, o) \in S$.*

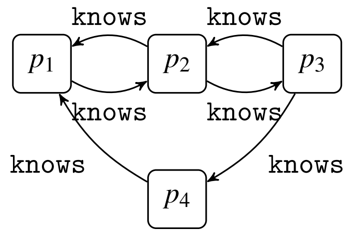
PROOF: Towards a contradiction, assume there is a database node o relevant to match variable $\mathtt{v}$ by $\mu \in [\![\mathbb{G}]\!]_{DB}$ with $(\mathtt{v}, o) \notin S$. But then $S \cup \mu$ is a dual simulation larger than S, contradicting the assumption that S is the largest one. □

Unfortunately, the converse direction, i. e., irrelevant nodes for BGP result sets are ruled out by the largest dual simulation, does not hold in full generality. Consider the example graphs P and K depicted in Fig. 4(a) and (b). The largest dual simulation between P and K includes node p_4 which is, however, not belonging to any match for the respective BGP. The reason why p_4 is not disqualified for variable/node $\mathtt{v}$ is that both nodes, p_1 and p_3 share the obligations for variable/node $\mathtt{w}$. Informally, p_1 knows p_4 via p_2 and p_3, although p_1 and p_3 do not have a direct link to one another. In a way, non-symmetric relationships are sometimes forced to appear symmetric by dual simulation. As long as acyclic queries are concerned, our process is complete. Since this class of queries is rather small, we are not formally justifying this statement.

We compute the largest dual simulation by the largest solution to the SOI constructed from $G(\mathbb{G})$, as explained in Sect. 3. Theorem 1 provides a definition of sound solutions of any of our systems of inequalities.



(a)



(b)

Figure 4: Example (a) Graph Pattern P and (b) Graph Database K, an example adapted from Ma et al. [20]

Definition 3 Let DB be a graph database, $\mathscr{Q}$ a SPARQL query and $\mathscr{E}$ any SOI representation of $\mathscr{Q}$ with solutions $S \subseteq vars(\mathscr{Q}) \times O_{DB}$. $\mathscr{E}$ is *sound* iff for the largest solution S of $\mathscr{E}$, it holds that if $\mu(v) = o$ for some $v \in vars(\mathscr{Q})$ and $\mu \in [\![\mathscr{Q}]\!]_{DB}$, then $(v, o) \in S$. ■

Soundness means that we do not remove nodes from the database important for any further processing of matches.

4.2 Advanced Graph Patterns

BGPs, and SPARQL queries in general, may be combined by operators, further restricting and combining the sets of matches. This subsection is devoted to applying dual simulation principles to queries with UNION- and AND-operators. The latter is best characterized as inner-joins of the results of two queries.

The UNION-operator is the least invasive operator. It combines any two queries $\mathscr{Q}_1$ and $\mathscr{Q}_2$ to query $\mathscr{Q}_1$ UNION $\mathscr{Q}_2$. The result set is the union of the result sets of the constituent queries, i. e., $[\![\mathscr{Q}_1 \text{ UNION } \mathscr{Q}_2]\!]_{DB} := [\![\mathscr{Q}_1]\!]_{DB} \cup [\![\mathscr{Q}_2]\!]_{DB}$. It is well-known that any SPARQL query may be rewritten as the union of finitely many *union-free* queries (cf. Proposition 3.8 [27]). A SPARQL query $\mathscr{Q}$ is *union-free* if the UNION-operator does not occur in $\mathscr{Q}$.

Proposition 3 (Proposition 3.8 [27]) *Let $\mathscr{Q}$ be a* SPARQL *query. Then there exist union-free* SPARQL *queries $\mathscr{Q}_1, \mathscr{Q}_2, \ldots, \mathscr{Q}_k$ ($k \in \mathbb{N}$) such that $\mathscr{Q}$ is equivalent to $\mathscr{Q}' = \mathscr{Q}_1 \text{ UNION } \mathscr{Q}_2 \text{ UNION} \ldots \text{UNION } \mathscr{Q}_k$, i. e., $[\![\mathscr{Q}]\!]_{DB} = [\![\mathscr{Q}']\!]_{DB}$.*

The construction of $\mathscr{Q}'$ follows similar principles as constructing the DNF (disjunctive normal form) in propositional logic. In consequence, the result set of $\mathscr{Q}$ is the union of the result sets of all the $\mathscr{Q}_i$ ($1 \leq i \leq k$). This means that instead of $\mathscr{Q}$, we may process each union-free part of $\mathscr{Q}$ individually and later combine their results. Hence, in the rest of the paper, we assume every query to be union-free.

While SPARQL's disjunction unifies the result sets of the constituents, conjunction builds the union of compatible results, i. e., those results agreeing upon shared variables. Two matches μ_1 and μ_2 are *compatible*, denoted $\mu_1 \rightleftharpoons \mu_2$, if for all $v \in dom(\mu_1) \cap dom(\mu_2)$ (v shared by μ_1 and μ_2), $\mu_1(v) = \mu_2(v)$. Let $\mathscr{Q}_1$ and $\mathscr{Q}_2$ be two queries. Their conjunction is the query $\mathscr{Q}_1$ AND $\mathscr{Q}_2$. As an example, the SPARQL representation of the graph pattern in Fig. 4(a) may be described as the conjunction of two BGPs, $\mathbb{G}_1 = \{(\texttt{v}, \texttt{knows}, \texttt{w})\}$ and $\mathbb{G}_2 = \{(\texttt{w}, \texttt{knows}, \texttt{v})\}$. The semantics of conjunctions is the set of unions of all compatible matches, i. e.,

$$[\![\mathscr{Q}_1 \text{ AND } \mathscr{Q}_2]\!]_{DB} := \{\mu_1 \cup \mu_2 \mid \mu_i \in [\![\mathscr{Q}_i]\!]_{DB} \wedge \mu_1 \rightleftharpoons \mu_2\}.$$

For example, the query from Fig. 4 matches $\mu_i \in [\![\mathbb{G}_i]\!]_{DB}$ ($i = 1, 2$) with $\mu_1(\texttt{v}) = \mu_2(\texttt{v}) = p_1$ and $\mu_1(\texttt{w}) =$

$\mu_2(\mathtt{w}) = p_2$ are compatible and thus $(\mu_1 \cup \mu_2) \in [\![\mathbb{G}_1 \text{ AND } \mathbb{G}_2]\!]_{DB}$. On the other hand, taking μ_1 as before but μ_2 with $\mu_2(\mathtt{w}) = p_2$ and $\mu_2(\mathtt{v}) = p_3$ yields incompatible matches, thus $(\mu_1 \cup \mu_2) \notin [\![\mathbb{G}_1 \text{ AND } \mathbb{G}_2]\!]_{DB}$. In a relational setting, conjunction simply is the explicit expression of inner joins.

Regarding our dual simulation process, for conjunctions $\mathscr{Q}_1$ AND $\mathscr{Q}_2$, we create the systems of inequalities for $\mathscr{Q}_1$ and $\mathscr{Q}_2$ separately, denoted by $\mathscr{E}(\mathscr{Q}_1)$ and $\mathscr{E}(\mathscr{Q}_2)$. Recall that the variables of both queries directly refer to variables occurring in $\mathscr{Q}_1$ and $\mathscr{Q}_2$, respectively. The semantics of conjunctions requires matches to queries $\mathscr{Q}_1$ and $\mathscr{Q}_2$ to be compatible. In consequence, assignments to the variables must be identical. This may be achieved by simply unifying the systems of inequalities of both queries. The following lemma defines the necessary system of inequalities.

Lemma 3 *Let DB be a graph database and $\mathscr{Q}_1, \mathscr{Q}_2$ two* SPARQL *queries with sound systems of inequalities $\mathscr{E}(\mathscr{Q}_1) = (\mathtt{Var}_1, \mathtt{Eq}_1)$ and $\mathscr{E}(\mathscr{Q}_2) = (\mathtt{Var}_2, \mathtt{Eq}_2)$. Then $\mathscr{E} = (\mathtt{Var}_1 \cup \mathtt{Var}_2, \mathtt{Eq}_1 \cup \mathtt{Eq}_2)$ is sound for $\mathscr{Q}_1$ AND $\mathscr{Q}_2$.*

PROOF: Let $\mu \in [\![\mathscr{Q}_1 \text{ AND } \mathscr{Q}_2]\!]_{DB}$. It holds that $\mu = \mu_1 \cup \mu_2$ for compatible $\mu_i \in [\![\mathscr{Q}_i]\!]_{DB}$ $(i = 1,2)$. Let $v \in vars(\mathscr{Q}_1 \text{ AND } \mathscr{Q}_2)$ with $\mu(v) = o$. We need to show that the largest solution S of $\mathscr{E}$ contains (v,o). In case $v \in vars(\mathscr{Q}_i) \setminus vars(\mathscr{Q}_j)$ $(i, j = 1,2$ and $i \neq j)$, $(v,o) \in S$ follows from soundness of $\mathscr{E}(\mathscr{Q}_i)$, since variable v cannot be influenced by any triple pattern of $\mathscr{E}(\mathscr{Q}_j)$. Otherwise, $v \in vars(\mathscr{Q}_1) \cap vars(\mathscr{Q}_2)$ and it holds that $\mu_1(v) = \mu_2(v) = o$. Hence, the largest solutions S_i of $\mathscr{E}(\mathscr{Q}_i)$ $(i = 1,2)$ contain (v,o), i. e., $(v,o) \in S_1 \cap S_2$. It remains to be shown that $S_1 \cap S_2 \subseteq S$. Let $(v,o) \in S_1 \cap S_2$. By construction, any $\varepsilon \in \mathtt{Eq}$ either comes from $\mathtt{Eq}_1$ or $\mathtt{Eq}_2$, and since $(v,o) \in S_1 \cap S_2$, (v,o) cannot contradict ε. Thus, (v,o) belongs to the largest solution S. □

Please note that the lemma is independent of the shape of $\mathscr{Q}_1$ and $\mathscr{Q}_2$. We only require that the respective SOIs are sound w. r. t. the queries (cf. Def. 3).

4.3 Optional Patterns

The last syntactic construct of SPARQL for which we provide a soundness argument for a dual simulation procedure is that of optional patterns. While it is the most involved SPARQL operator, our pruning procedure needs a rather small adjustment. Reconsider our introductory query $(\mathscr{X}_1)$, where we asked for directors and their coworkers. If we are not sure whether or not for every director there is a listed person she worked with, then we may put this information in an optional pattern, yielding query $(\mathscr{X}_2)$.

```
SELECT * WHERE {
  ?director directed ?movie .
  OPTIONAL {
    ?director worked_with ?coworker .  } }
```

$(\mathscr{X}_2)$

Optional patterns are left-outer joins in the relational model, i. e., matches to $(\mathscr{X}_2)$ assign nodes from the database to variable `?director` and `?movie` but only also to variable `?coworker` if there is one. Regarding the example graph database in Fig. 1(a), we obtain all bold subgraphs, as before, and additionally the semi-thick subgraphs (with `D. Koepp` and `T. Young` as `?director`). In general, for queries $\mathscr{Q}_1$ and $\mathscr{Q}_2$, the result set of $\mathscr{Q}_1$ AND $\mathscr{Q}_2$ is contained in the result set of the optional pattern. Additionally, all matches to $\mathscr{Q}_1$ that have no compatible matches to $\mathscr{Q}_2$ are matches. Formally,

$$[\![\mathscr{Q}_1 \text{ OPTIONAL } \mathscr{Q}_2]\!]_{DB} := [\![\mathscr{Q}_1 \text{ AND } \mathscr{Q}_2]\!]_{DB} \cup \{\mu \in [\![\mathscr{Q}_1]\!]_{DB} \mid \nexists \mu' \in [\![\mathscr{Q}_2]\!]_{DB} : \mu \rightleftharpoons \mu'\}.$$

In $(\mathscr{X}_2)$, variable `?director` occurs in two different roles. First, the optional pattern mandates variable `?director` to feature triples with label `directed`. Second, triples labeled `worked_with` are only

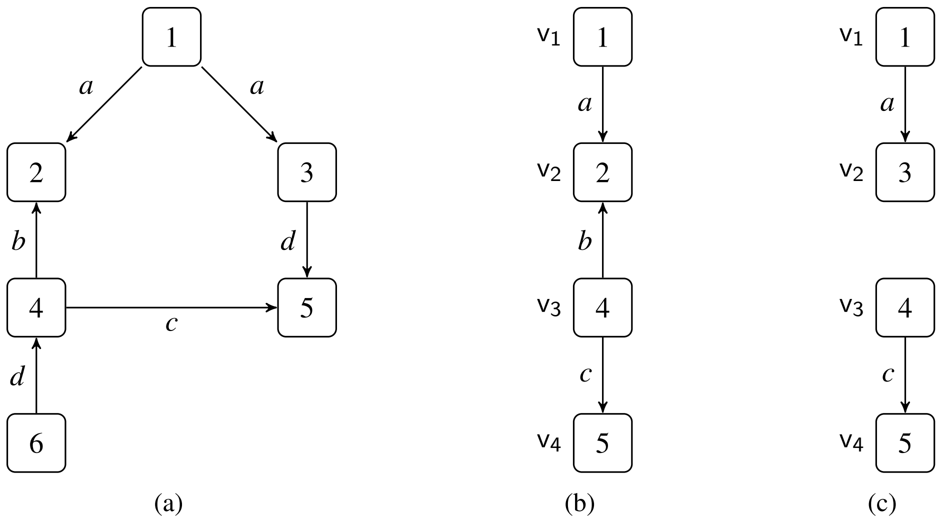


Figure 5: Small Graph Database and Graph Representations of the Matches of ($\mathscr{X}_3$)

optional. These two roles must be reflected by our SOI representation of ($\mathscr{X}_2$) in that we have two copies of that variable, $\texttt{?director}_m$ (mandatory) and $\texttt{?director}_o$ (optional) with the property that a solution S in variable $\texttt{?director}_o$ must not exceed S in variable $\texttt{?director}_m$. In other words, there is no database node matching $\texttt{director}_o$ that does not match $\texttt{?director}_m$. This is expressed by the following inequality,

$$\texttt{?director}_o \leq \texttt{?director}_m. \tag{14}$$

Towards the general case, let us discuss another example query,

$$(\{(\mathtt{v}_1, a, \mathtt{v}_2)\} \text{ OPTIONAL } \{(\mathtt{v}_3, b, \mathtt{v}_2)\}) \text{ AND } \{(\mathtt{v}_3, c, \mathtt{v}_4)\}. \qquad (\mathscr{X}_3)$$

The query consists of three triple patterns, the first two constitute an optional pattern and the results are joined with the third triple pattern. Fig. 5(b) and (c) show graph representations of the matches of ($\mathscr{X}_3$) w. r. t. the graph database in Fig. 5(a), in which the variable assignments are indicated as labels next to the nodes. As before, we have variables occurring in two different roles, here $\mathtt{v}_2$ and $\mathtt{v}_3$. Analogous to ($\mathscr{X}_2$), we simply derive $\mathtt{v}_{2m}$ and $\mathtt{v}_{2o}$ with $\mathtt{v}_{2o} \leq \mathtt{v}_{2m}$ from the optional pattern. However, the first occurrence of $\mathtt{v}_3$ is optional w. r. t. to second occurrence (in the third triple pattern), since $\mathtt{v}_3$ must match a c-labeled edge and may feature the b-labeled edge from the optional pattern. We observe that occurrences of the same variables, e. g., $\mathtt{v}_3$, may have interdependencies that we need to take into account, even beyond optional patterns. Therefore, we first introduce the full syntax we cover in this paper to derive a system of inequalities for each query following that syntax. Second, we derive sound SOIs for optional patterns.

Our query language $\mathscr{S}$ comprises union-free SPARQL queries with AND and OPTIONAL operators. Queries in $\mathscr{S}$ are derived by the following grammar,

$$\mathscr{Q} \quad ::= \quad \mathbb{G} \;\big|\; \mathscr{Q} \text{ AND } \mathscr{Q} \;\big|\; \mathscr{Q} \text{ OPTIONAL } \mathscr{Q}$$

where $\mathbb{G}$ ranges over by BGPs. Queries range over by $\mathscr{Q}, \mathscr{Q}_1, \mathscr{Q}_2, \ldots \in \mathscr{S}$ As observed above, we need to take mandatory and optional variable occurrences into account. Function *mand* maps queries $\mathscr{Q}$ from $\mathscr{S}$ to the set of variables that occur as mandatory in the $\mathscr{Q}$, defined by

1. $\mathit{mand}(\mathbb{G}) := \mathit{vars}(\mathbb{G})$,

2. $mand(\mathscr{Q}_1 \text{ AND } \mathscr{Q}_2) := mand(\mathscr{Q}_1) \cup mand(\mathscr{Q}_2)$, and
3. $mand(\mathscr{Q}_1 \text{ OPTIONAL } \mathscr{Q}_2) := mand(\mathscr{Q}_1)$.

For handling optional pattern $\mathscr{Q}_1$ OPTIONAL $\mathscr{Q}_2$ correctly, we need to decide, in which cases an occurrence of variable $\mathtt{v}$ in $\mathscr{Q}_2$ has an optional dependency to another occurrence of the same variable. The case $\mathtt{v} \in vars(\mathscr{Q}_1)$ is reflected by the example query ($\mathscr{X}_2$). Upon identification of such mandatory/optional pairs, we rename the optional occurrences of variables in our SOI and add an inequality as before, e. g., Eq. (14). More precisely, for the special case of query $\mathscr{Q} = \mathscr{Q}_1$ OPTIONAL $\mathscr{Q}_2$, we create the SOI representation for $\mathscr{Q}$ by first identifying mandatory/optional dependencies between $\mathscr{Q}_1$ and $\mathscr{Q}_2$, that are occurrences of variables $\mathtt{v} \in vars(\mathscr{Q}_2) \cap mand(\mathscr{Q}_1)$. For $\mathtt{v} \in vars(\mathscr{Q}_2) \cap mand(\mathscr{Q}_1)$, we reserve a unique name $\mathtt{v}_{\mathscr{Q}_2}$, which we use to replace $\mathtt{v}$ in every inequality of $\mathscr{Q}_2$, achieved by a renaming $\rho := \{(\mathtt{v}, \mathtt{v}_{\mathscr{Q}_2}) \mid \mathtt{v} \in vars(\mathscr{Q}_2) \cap mand(\mathscr{Q}_1)\}$. Furthermore, the inequality

$$\mathtt{v}_{\mathscr{Q}_2} \leq \mathtt{v} \tag{15}$$

for $\mathtt{v} \in vars(\mathscr{Q}_2) \cap mand(\mathscr{Q}_1)$ is added to the overall SOI. The largest solution to the resulting SOI contains assignments to the new variables $\mathtt{v}_{\mathscr{Q}_2}$, i. e., to variables not occurring in the original formulation of the query. Since these variables are only needed to handle optionality correctly, and since the largest solution of these variables are subsumed by the respective mandatory variables (cf. Eq. 15), we may ignore them in the final result of the pruning step.

Lemma 4 *Let DB be a graph database and $\mathscr{Q}_1, \mathscr{Q}_2$ two* SPARQL *queries with sound systems of inequalities $\mathscr{E}(\mathscr{Q}_1) = (\mathtt{Var}_1, \mathtt{Eq}_1)$ and $\mathscr{E}(\mathscr{Q}_2) = (\mathtt{Var}_2, \mathtt{Eq}_2)$. Furthermore, define renaming as $\rho := \{(\mathtt{v}, \mathtt{v}_{\mathscr{Q}_2}) \mid \mathtt{v} \in vars(\mathscr{Q}_2) \cap mand(\mathscr{Q}_1)\}$. Then*

$$\mathscr{E} \quad = \quad (\mathtt{Var}_1 \cup \rho(\mathtt{Var}_2), \mathtt{Eq}_1 \cup \rho(\mathtt{Eq}_2) \cup \mathtt{Eq}_o)$$

with $\mathtt{Eq}_0 := \{\mathtt{v}_{\mathscr{Q}_2} \leq \mathtt{v} \mid \mathtt{v} \in vars(\mathscr{Q}_2) \cap mand(\mathscr{Q}_1)\}$ is sound for $\mathscr{Q}_1$ OPTIONAL $\mathscr{Q}_2$.

PROOF: Let $\mu \in [\![\mathscr{Q}_1 \text{ OPTIONAL } \mathscr{Q}_2]\!]_{DB}$ with $\mu(v) = o$ for $\mathtt{v} \in vars(\mathscr{Q}_1 \text{ OPTIONAL } \mathscr{Q}_2)$. We need to show that $(\mathtt{v}, o) \in S$ where S is the largest solution of $\mathscr{E}$. There are two cases to distinguish, (a) $\mu = \mu_1 \cup \mu_2$ where $\mu_i \in [\![\mathscr{Q}_i]\!]_{DB}$ $(i = 1,2)$ with $\mu_1 \rightleftharpoons \mu_2$ and (b) $\mu = \mu_1$ where $\mu_1 \in [\![\mathscr{Q}_1]\!]_{DB}$ and there is no $\mu_2 \in [\![\mathscr{Q}_2]\!]_{DB}$ compatible to μ_1. Case (a) becomes analogous to the proof of Lemma 3, considering that for any occurrence of $\mathtt{v}_{\mathscr{Q}_2}$ in $\rho(\mathtt{Eq}_2)$, inequality (15) makes the requirements upon $\mathtt{v}_{\mathscr{Q}_2}$ only weaker. Hence, $\mu_2(\mathtt{v})$ is preserved. In case (b), we distinguish two further cases for variable $\mathtt{v}$, (i) $\mathtt{v} \in vars(\mathscr{Q}_1) \setminus vars(\mathscr{Q}_2)$ and (ii) $\mathtt{v} \in vars(\mathscr{Q}_1) \cap vars(\mathscr{Q}_2)$. The claim for case (i) directly follows from the sound SOI $\mathscr{E}(\mathscr{Q}_1)$. In case (ii), it might be that in the largest solution S_2 of $\mathscr{E}(\mathscr{Q}_2)$, $(\mathtt{v}, o) \notin S_2$. However, $\mathtt{v}$ is subject to renaming, since it is a variable of both sub-queries. Therefore $(\mathtt{v}_{\mathscr{Q}_2}, o) \notin \rho S_2$ but as we added inequality (15) to $\mathtt{Eq}_o$, we get that $(\mathtt{v}, o) \in S$ by soundness of $\mathtt{E}(\mathscr{Q}_1)$. □

4.3.1 The General Case

The general case, outlined by example ($\mathscr{X}_3$), needs to take the contexts of optional patterns into account. Since $\mathtt{v}_3$ has a mandatory occurrence in ($\mathscr{X}_3$) but an optional in the sub-query $\{(\mathtt{v}_1, a, \mathtt{v}_2)\}$ OPTIONAL $\{(\mathtt{v}_3, b, \mathtt{v}_2)\}$, SPARQL's evaluation semantics defines the second occurrence of $\mathtt{v}_3$ to be mandatory w. r. t. the first. For any optional pattern $\mathscr{Q}_1$ OPTIONAL $\mathscr{Q}_2$ occurring as a sub-query of a query $\mathscr{Q} \in \mathscr{S}$, if a variable $\mathtt{v} \in vars(\mathscr{Q}_2)$ occurs as mandatory in $\mathscr{Q}$, then we perform the same renaming as in Lemma 4 for $\mathscr{Q}_2$. For a variable $\mathtt{v} \in vars(\mathscr{Q}_2)$, there may be several candidates. From all the choices we pick the *syntactically closest*. As an example, consider the optional patterns

$$\begin{aligned} P &= (P_1 \text{ OPTIONAL } P_2) \text{ OPTIONAL } P_3 \text{ and} \\ R &= R_1 \text{ OPTIONAL } (R_2 \text{ OPTIONAL } R_3). \end{aligned}$$

Assume that $\mathtt{y} \in vars(P_i)$ $(i = 1,2,3)$ and $\mathtt{z} \in vars(R_i)$ $(i = 1,2,3)$. The occurrences of $\mathtt{y}$ in P_2 and P_3 are syntactically closest to the mandatory occurrence of $\mathtt{y}$ in P_1, giving rise to inequalities

$$\begin{aligned} \mathtt{y}_{P_2} &\leq \mathtt{y} \\ \mathtt{y}_{P_3} &\leq \mathtt{y}. \end{aligned}$$

It may also be that $\mathtt{x} \in vars(P_i)$ $(i = 2,3)$ and $\mathtt{x} \notin vars(P_1)$. In these situations, we rename $\mathtt{x}$ to $\mathtt{x}_{P_2}$ and $\mathtt{x}_{P_3}$, respectively, but would not add any interdependencies between these variables. In extreme cases, the original variable $\mathtt{x}$ may not occur in the resulting SOI at all. In these cases, the soundness proof requires that every solution to $\mathtt{x}_{P_2}$ or $\mathtt{x}_{P_3}$ also is a solution to variable $\mathtt{x}$.

The occurrence of $\mathtt{z}$ in R_3 is closest to the occurrence in R_2, and the occurrence in R_2 is closest to R_1, raising the following inequalities,

$$\begin{aligned} \mathtt{z}_{R_3} &\leq \mathtt{z}_{R_2} \\ \mathtt{z}_{R_2} &\leq \mathtt{z}. \end{aligned}$$

Handling the general case formally, needs to conduct a notion of $\mathscr{S}$-contexts, being queries with holes. Since the proof of the resulting soundness lemma is completely analogous, thus gives no more insights than the proof of Lemma 4, our considerations about optional patterns are complete.

We have now given sound SOI representations for SPARQL queries. The following subsection gives the cumulating theorem in this respect as well as some final remarks on the limits of our pruning process. Sect. 5 provides evidence of the effectiveness and efficiency of the derived procedure.

4.4 Discussion

Before discussing an important query type, we conclude this section by the soundness theorem.

Theorem 2 (Soundness) *Let DB be a graph database and $\mathscr{Q} \in \mathscr{S}$. Then $\mathscr{E}(\mathscr{Q})$ is a sound SOI.*

The proof is by induction over the structure of $\mathscr{Q}$ and uses all results obtained so far.

Our theoretical considerations are limited to SPARQL queries in which every node of a triple pattern is a variable. SPARQL also allows to mention constant nodes from the database, often drastically reducing the number of possible results. The key idea to integrate constant nodes into our pruning technique is to alter the initialization inequality (12).

As already mentioned, our dual simulation process is not restricted to well-designed patterns, being SPARQL queries $\mathscr{Q}$ with the property that for every sub-query $\mathscr{Q}_1$ OPTIONAL $\mathscr{Q}_2$ and every $\mathtt{v} \in vars(\mathscr{Q}_2)$ that also occur outside the optional pattern, $\mathtt{v} \in vars(\mathscr{Q}_1)$ [27]. The example query $(\mathscr{X}_3)$ is non-well-designed, since $\mathtt{v}_3$ occurs as an optional variable but another time outside the optional sub-pattern. Non-well-designed patterns give rise to cross-product results, as indicated by the match in Fig. 5(c). Assume that we have several c-labeled edges, then each of these edges together with the a-labeled edge forms an answer to the query. In these situations, our procedure remains effective, since it handles both occurrences of variable $\mathtt{v}_3$ separately. The process does not take compatibility into account, leaving our evaluation algorithm efficient.

There are two reasons making well-designed patterns interesting. First, the fragment containing only well-designed patterns has a coNP-complete evaluation problem [27, 2], as opposed to PSPACE-completeness of SPARQL's evaluation problem. Second, every well-designed pattern is *weakly monotone* [2], an important property when discussing *NULL* semantics. It is unknown whether the property of well-designed queries completely characterizes weakly monotone queries [2] in that there might be queries that are non-well-designed but weakly monotone. To this end, we cannot tell whether or not we handle all weakly monotone queries effectively. However, the next section provides evidence that our pruning is quite effective.

5 Evaluation

In this section, we evaluate the prototype implementation of our dual simulation-based query processing, called SPARQLSIM. First, we compare our algorithm to the state-of-the-art dual simulation algorithm as introduced introduced by Ma et al. [20] and used in implementations of [24, 31, 20] for evaluation purposes. Furthermore, we analyze how our SPARQL extension of dual simulation may be used to effectively and efficiently prune graph databases to improve SPARQL query processing on an in-memory RDF database and a triple store based on relational database technology. Here, we first analyze the pruning effectiveness and then compare query evaluation times with two graph database systems on two very large graph datasets comprising 750 million and 1.3 billion triples. Our focus is on highly time-consuming optional queries, also being used by Atre [4]. More details concerning the evaluation results, a list of queries and our implementation can be found on our project's Github page[3].

5.1 Experimental Setup

For the first experiment, we have implemented the dual-simulation algorithm of Ma et al. in C++ together with the prototype implementation of our algorithm. To evaluate our prototypes' performance as a pruning mechanism, we employed one of the fastest RDF databases Virtuoso [9] and the high-performance in-memory database RDFox [25]. Several combinations of these attributes are indexed using B-trees, ensuring fast data access and high join performance. RDFox on the other hand, is a centralized and scalable in-memory database which is highly space-efficient. All experiments were performed on a server running Ubuntu 16.04 with four XEON E7-8837, 2.67 GHz, having 8 Cores each, 384GB RAM and Kingston DCP1000 NVMe PCI-E SSD. To achieve stable query results, we deactivated caching for Virtuoso. RDFox is not using caching techniques at all, since it is an in-memory database. For the evaluation, we have run all queries 10 times on each database and averaged the times. Furthermore, we measured the query time in Virtuoso using the query profile command. Overall, both database systems only very small fluctuations in querying times could be observed.

Since we provide a dual simulation algorithm that can be used as an external pruning mechanism, we imported the result sets from our tool into the two databases manually and then processed the queries on the pruned result sets in comparison to queries on the full databases. Here, we did not take into account the export time from our tool and the import time into the database, because our tool could easily be integrated into a standard database system, using our computations internally.

Our evaluation data comprises two popular RDF datasets: (1) The latest DBpedia dump 2016-10 in the English language version [6] and (2) the synthetic Lehigh University Benchmark [15] (LUBM) dataset generated for 1000 universities. The DBpedia dataset comprises 751,603,507 triples, including 216,132,665 distinct nodes and 65,430 predicates. As DBpedia benchmark queries we use the queries as reported in [4] (cf. $\mathscr{D}_0$-$\mathscr{D}_5$) and benchmark queries (cf. $\mathscr{B}_0$-$\mathscr{B}_{21}$) from the DBpedia benchmark dataset in [23]. The synthesized LUBM benchmark dataset was built from 10,000 universities using a generation tool, comprising 1,381,692,508 triples with 18 distinct predicates and 328,620,750 nodes. LUBM is a dataset generator for benchmark datasets to measure RDF database's scalability. In comparison to DBpedia, the selectivity of LUBM attributes is extremely low, since it only comprises 18 different predicates on much more triples. Since official query sets hardly cover optional patterns, we again rely on queries that have been used by Atre [4] (cf. $\mathscr{L}0$-$\mathscr{L}5$).

[3]`https://github.com/ifis-tu-bs/sparqlSim`

Table 1: Running time of our dual simulation algorithm for BGPs from queries $\mathscr{B}_0$-$\mathscr{B}_{21}$ compared to Ma et al. [20].

Query	$t_{\text{SPARQLSIM}}$	$t_{\text{MA ET AL.}}$
$\mathscr{B}_0$	0.10385	6.72121
$\mathscr{B}_1$	0.03876	3.33471
$\mathscr{B}_2$	0.79097	3.84781
$\mathscr{B}_3$	0.69797	5.62662
$\mathscr{B}_4$	0.00003	0.00004
$\mathscr{B}_5$	0.04091	0.31700
$\mathscr{B}_6$	0.00190	0.00342
$\mathscr{B}_7$	0.41105	0.54291
$\mathscr{B}_8$	0.26991	0.51206
$\mathscr{B}_9$	0.13562	5.51084
$\mathscr{B}_{10}$	0.02551	0.08707
$\mathscr{B}_{11}$	0.02397	0.27126
$\mathscr{B}_{12}$	0.01392	0.02099
$\mathscr{B}_{13}$	0.01477	0.02287
$\mathscr{B}_{14}$	0.35515	11.30355
$\mathscr{B}_{15}$	10.34747	22.90143
$\mathscr{B}_{16}$	5.46599	16.63957
$\mathscr{B}_{17}$	13.43710	24.99660
$\mathscr{B}_{18}$	0.00002	0.00003
$\mathscr{B}_{19}$	1.12649	2.30390
$\mathscr{B}_{20}$	0.32056	0.54057
$\mathscr{B}_{21}$	0.69515	5.15070

5.2 Evaluation Analysis

Comparing Dual Simulation Algorithms In this first experiment, we compare our dual-simulation algorithm to the results of the standard dual simulation algorithm as proposed in [20]. Due to the fact that Ma et al.'s algorithm only considers BGPs as input, we have modified benchmark queries $\mathscr{B}_0$-$\mathscr{B}_{21}$ by removing all SPARQL keywords, i. e., OPTIONALs. We present the results in Table 1. Here, we observe that the optimizations allowed by SPARQLSIM (cf. Sect. 3.3) pay off, since our processing times are always lower than for Ma et al., often even by an order of magnitude. It is this order of magnitude, the naive algorithm lacks when running in graph database query scenarios.

Dual-Simulation as a Pruning Mechanism for SPARQL In Table 2, we first analyze SPARQLSIM's pruning effectiveness of dual-simulation for all LUBM and DBpedia queries. We observe that the number of triples is drastically decreased from the original databases for all queries. In our experiments, for queries with empty result sets, SPARQLSIM is capable of producing this answer, making any further query evaluation unnecessary. Over all tested queries, we prune at least 95% of all database triples.

SPARQLSIM's evaluation time is heavily depending on the query and the dataset. With LUBM, having only 18 distinct predicates, we have a an extreme case that often needs more than 30 iterations over the SOI, leading to high runtimes of the algorithm. Some large LUBM queries have a very complex

Table 2: Evaluation queries, the result set size, the runtime time of our dual-simulation algorithm in seconds and the number of triples after applying our algorithm. (Note that the number of single triples after pruning can be smaller than the number of results.)

Query	Result Set Size	$t_{\text{SPARQLSIM}}$	Triples after Pruning
$\mathscr{L}0$	10448905	106.451	10181730
$\mathscr{L}1$	226641	8.464	25429750
$\mathscr{L}2$	32828280	147.335	48674046
$\mathscr{L}3$	11	0.138	126
$\mathscr{L}4$	10	0.125	101
$\mathscr{L}5$	7	1.220	35
$\mathscr{D}_0$	523066	4.396	3141102
$\mathscr{D}_1$	0	0.002	0
$\mathscr{D}_2$	12	0.088	60
$\mathscr{D}_3$	5794	0.143	28704
$\mathscr{D}_4$	25102459	6.230	22691521
$\mathscr{D}_5$	365693	0.574	79944
$\mathscr{B}_0$	12	0.088	60
$\mathscr{B}_1$	859751	0.022	726812
$\mathscr{B}_2$	913786	0.532	1588127
$\mathscr{B}_3$	438542	0.606	386020
$\mathscr{B}_4$	0	0.000	0
$\mathscr{B}_5$	0	0.033	0
$\mathscr{B}_6$	365693	0.568	79944
$\mathscr{B}_7$	815522	0.503	886939
$\mathscr{B}_8$	34991	0.443	37965
$\mathscr{B}_9$	8416	0.113	30258
$\mathscr{B}_{10}$	8247	0.022	13116
$\mathscr{B}_{11}$	8061	0.027	12642
$\mathscr{B}_{12}$	9849	0.018	8955
$\mathscr{B}_{13}$	9554	0.018	8660
$\mathscr{B}_{14}$	123467	0.273	365154
$\mathscr{B}_{15}$	25102459	7.454	22691521
$\mathscr{B}_{16}$	22673220	4.322	27747192
$\mathscr{B}_{17}$	0	0.000	0
$\mathscr{B}_{18}$	2	0.009	4
$\mathscr{B}_{19}$	7898331	0.917	8294385
$\mathscr{B}_{20}$	66903	0.472	41808
$\mathscr{B}_{21}$	879460	0.602	292541

structure which in some cases is a problem for our evaluation order heuristic leading to a large number of iterations and therefore high pruning times. Considering the size of LUBM and the query complexity, our pruning times are still moderately fast. For DBpedia queries, we usually perform the computation in only a split second. Queries with low selectivity relationships/joins take a couple of seconds, since fewer triples are pruned during the process.

Comparison to State-Of-The-Art RDF Databases The next experiments compare the query evaluation time of the in-memory database RDFox to SPARQLSIM in combination with RDFox as a query processor. Table 3 lists all results. First of all, we observe that the query time in 15 out of 34 queries is improved. Particularly interesting is our improvement on query $\mathscr{L}1$ with a query processing time of 25,900 seconds on RDFox. Here, we could run our dual-simulation algorithm in only 8 seconds, decreasing the query time of RDFox by more than 20 times. For $\mathscr{L}0$, however, $t_{\text{SPARQLSIM}}$ alone is around 5 times slower than RDFox (t_{DB}). We strongly believe that here our evaluation order heuristic failed, massively increasing the number of iterations of our algorithm. Also in queries $\mathscr{D}_5$, $\mathscr{B}_0$, $\mathscr{B}_7$-$\mathscr{B}_9$, $\mathscr{B}_{17}$, $\mathscr{B}_{21}$, we show good improvements of the in-memory databases query times. For most of the remaining queries, we show comparable results to RDFox only varying by some milliseconds.

The results on pruning Virtuoso queries are listed in Table 4. Here, we can improve upon the runtimes of only 3 queries. For most other queries, evaluation times are on par with t_{DB}. We notice that for some queries our pruning could not increase Virtuoso's evaluation time as much as for RDFox. A detailed analysis of Virtuosos query plans showed that this was due to changes in the join order that sometimes drastically increases the number of intermediate results, e. g., $\mathscr{D}_4$ with doubled evaluation time $t_{\text{DB pruned}}$ on the 3% portion of DBpedia (cf. Table 2). We believe that Virtuoso could benefit by an integration of SPARQLSIM as a pruning technique, directly benefiting from Virtuoso's extremely good runtime estimations. On the downside, our algorithm often is slightly slower than the professionally implemented and highly optimized RDF triple store. Particularly, some of the more complex queries took longer to produce the pruning than for Virtuoso to produce the answers. These queries took several iterations in SPARQLSIM, because the evaluation order of our system of inequalities was far from optimal. We believe that we can benefit from more sophisticated join order optimization techniques as used for example in Virtuoso which could boost our query time tremendously. The very fast pruning times $\mathscr{L}1$ (only 8.464s) for a highly complex query, using only two iterations, shows the potential of our technique. Therefore, in future work, we plan to further investigate the influence of the evaluation order on query times.

6 Related Work

Recently, graph pattern matching has become a trending topic for graph databases, different from the canonical though costly candidate of graph isomorphism (or homomorphism), with the goal of reducing structural requirements of the answer graphs. Especially simulations have been implemented for different graph database tasks [8, 12, 10, 20, 24]. Their experiments indicate advantages of simulation-based matching relations when analyzing social network patterns, as they offer the possibility to collapse several nodes into one node and vice versa. Ma et al. [19] introduce the notion of *dual simulation*. Having a simulation preorder taking forwards and backwards edges into account already appeared in the book of Abiteboul et al. [1], where it is proposed as a typing mechanism for semistructured data. On the downside, the performance improvements by dual simulation comes with a loss of topology for graph queries as criticized by Ma et al. [19]. Hence, they proposed an extension of dual simulation, putting distance restrictions on simulating nodes, in order to preserve as much topology as possible.

Several emerging applications ask for a simulation-based querying paradigm. Mottin et al. also build on strong simulation while proposing a new query paradigm called *Exemplar Queries* [24]. In exemplar querying, the query is a graph that works as an example answer the user would like to be returned by the database. The use of strong simulation is motivated by its ability to group matching nodes in possible answers, relaxing the requirements of strict isomorphic matchings but still preserving the semantics of the original query. It is not hard to see that exemplar queries as well as other applications of graph pattern matching may exhibit the portion of SPARQL we integrated into our framework, making their proposals more attractive to users.

Using simulation as a pruning mechanism for graph database queries has been proposed in [31]. The authors propose to compute a subgraph simulation on a graph database to filter unnecessary tuples to answer isomorphism. In their large-scale evaluation, they show the improvements in query time against several other isomorphic-based query processors. In contrast, we rely on dual simulation being more effective in pruning unnecessary triples, and we implement a fast dual simulation algorithm operating on bit matrices which is particularly fast for large graph databases. Furthermore, we use a more expressive query model that could also be integrated into their pruning technique to support more complex queries. Other existing approaches for optimizing graph database querying rely on adapting traditional database optimization techniques, usually leading to major improvements with regard to the query performance [7, 9]. However, graph database queries usually consist of a large number of joins with oftentimes huge intermediate results, requiring specialized optimization techniques. Therefore, join order estimation for graph databases, especially RDF triple stores, is still an active field [30, 26, 18]. Our proposal is to appreciate the data model of graph databases and perform light-weight graph algorithms to support the traditional database optimization. In fact, upon effectiveness of SPARQLSIM huge intermediate results may be avoided.

A large portion of research has been invested into simulation-based indexing techniques, which have already been used for join-ahead pruning in XML databases [22]. The index is created by computing bisimulating equivalence classes of nodes on the original database. Each equivalence class groups structurally bisimilar nodes, i. e., nodes that are found to be bisimulation equivalent [28, 32]. Bisimulation is more restrictive than dual simulation which we use throughout this paper. However, our algorithm could benefit from similar ideas. For us, it would be sufficient to produce dual simulation equivalence classes, promising to obtain a much smaller database fingerprint than possible with bisimulations, since dual simulation equivalence is coarser than bisimulation.

7 Conclusion

In this paper, we proposed an efficient graph pattern matching processing of SPARQL queries. Our algorithm is based on dual simulation and for all extensions, due to SPARQL, we provided soundness proofs, i. e., the result contains as much information to further filter the data without losing any matches due to SPARQL. In order to derive an algorithm competing with state-of-the-art graph databases, we contribute an alternative characterization of dual simulation in terms of a system of inequalities (cf. Sect. 3) in the size of the query. Dual simulation is directly applicable to SPARQL's BGPs, whereas composite queries, including AND and OPTIONAL operators, are handled by sound conservative extensions of dual simulation.

Our evaluation has shown that we outperform standard dual simulation algorithms on a variety of real-world SPARQL BGPs. Furthermore, our dual simulation algorithm can be used to aggressively prune triples, speeding up graph database query processing for state-of-the-art graph databases. In comparison to these graph databases, we could improve the query evaluation time for several queries drastically, and show comparable results for the others. We believe that most database systems would benefit from our technique by directly integrating it into their query processor. Further applications already using dual simulation may benefit from our SPARQL extension to offer more expressive query pattern capabilities.

References

[1] Serge Abiteboul, Peter Buneman & Dan Suciu (2000): *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[2] Marcelo Arenas, Claudio Gutierrez, Daniel P. Miranker, Jorge Pérez & Juan F. Sequeda (2013): *Querying Semantic Data on the Web?* SIGMOD Rec. 41(4), pp. 6–17, doi:10.1145/2430456.2430458. Available at `http://doi.acm.org/10.1145/2430456.2430458`.

[3] Marcelo Arenas & Martin Ugarte (2017): *Designing a Query Language for RDF: Marrying Open and Closed Worlds*. ACM Trans. Database Syst. 42(4), pp. 21:1–21:46, doi:10.1145/3129247. Available at `http://doi.acm.org/10.1145/3129247`.

[4] Medha Atre (2015): *Left Bit Right: For SPARQL Join Queries with OPTIONAL Patterns (Left-outer-joins)*. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, ACM, New York, NY, USA, pp. 1793–1808, doi:10.1145/2723372.2746483. Available at `http://doi.acm.org/10.1145/2723372.2746483`.

[5] Medha Atre, Vineet Chaoji, Mohammed J. Zaki & James A. Hendler (2010): *Matrix "Bit" Loaded: A Scalable Lightweight Join Query Processor for RDF Data*. In: Proceedings of the 19th International Conference on World Wide Web, WWW '10, ACM, New York, NY, USA, pp. 41–50, doi:10.1145/1772690.1772696. Available at `http://doi.acm.org/10.1145/1772690.1772696`.

[6] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak & Zachary Ives (2007): *DBpedia: A Nucleus for a Web of Open Data*. In: The Semantic Web, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 722–735.

[7] Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea & Bishwaranjan Bhattacharjee (2013): *Building an Efficient RDF Store over a Relational Database*. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, ACM, New York, NY, USA, pp. 121–132, doi:10.1145/2463676.2463718. Available at `http://doi.acm.org/10.1145/2463676.2463718`.

[8] Joel Brynielsson, Johanna Hogberg, Lisa Kaati, Christian Martenson & Pontus Svenson (2010): *Detecting Social Positions Using Simulation*. In: Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining, ASONAM '10, IEEE Computer Society, Washington, DC, USA, pp. 48–55, doi:10.1109/ASONAM.2010.52. Available at `http://dx.doi.org/10.1109/ASONAM.2010.52`.

[9] Orri Erling & Ivan Mikhailov (2009): *RDF Support in the Virtuoso DBMS*, pp. 7–24. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-02184-8_2. Available at `https://doi.org/10.1007/978-3-642-02184-8_2`.

[10] Wenfei Fan (2012): *Graph Pattern Matching Revised for Social Network Analysis*. In: Proceedings of the 15th International Conference on Database Theory, ICDT '12, ACM, New York, NY, USA, pp. 8–21, doi:10.1145/2274576.2274578. Available at `http://doi.acm.org/10.1145/2274576.2274578`.

[11] Wenfei Fan, Zhe Fan, Chao Tian & Xin Luna Dong (2015): *Keys for Graphs*. Proc. VLDB Endow. 8(12), pp. 1590–1601, doi:10.14778/2824032.2824056. Available at `http://dx.doi.org/10.14778/2824032.2824056`.

[12] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu & Yunpeng Wu (2010): *Graph Pattern Matching: From Intractable to Polynomial Time*. Proc. VLDB Endow. 3(1-2), pp. 264–275, doi:10.14778/1920841.1920878. Available at `http://dx.doi.org/10.14778/1920841.1920878`.

[13] Wenfei Fan, Jianzhong Li, Shuai Ma, Hongzhi Wang & Yinghui Wu (2010): *Graph Homomorphism Revisited for Graph Matching*. Proc. VLDB Endow. 3(1-2), pp. 1161–1172, doi:10.14778/1920841.1920986. Available at http://dx.doi.org/10.14778/1920841.1920986.

[14] Nadime Francis, Alastair Green, Paolo Guargliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Martin Schuster, Petra Selmer & Andrés Taylor (2018): *Formal Semantics of the Language Cypher*. Available at http://arxiv.org/abs/1802.09984.

[15] Yuanbo Guo, Zhengxiang Pan & Jeff Heflin (2005): *LUBM: A benchmark for OWL knowledge base systems*. Web Semantics: Science, Services and Agents on the World Wide Web 3(2), pp. 158 – 182, doi:https://doi.org/10.1016/j.websem.2005.06.005. Available at http://www.sciencedirect.com/science/article/pii/S1570826805000132. Selcted Papers from the International Semantic Web Conference, 2004.

[16] M. R. Henzinger, T. A. Henzinger & P. W. Kopke (1995): *Computing Simulations on Finite and Infinite Graphs*. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, FOCS '95, IEEE Computer Society, Washington, DC, USA, pp. 453–. Available at http://dl.acm.org/citation.cfm?id=795662.796255.

[17] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics & Jeong-Hoon Lee (2013): *An in-depth comparison of subgraph isomorphism algorithms in graph databases*. In: Proceedings of the 39th international conference on Very Large Data Bases, PVLDB'13, VLDB Endowment, pp. 133–144. Available at http://dl.acm.org/citation.cfm?id=2448936.2448946.

[18] Andrés Letelier, Jorge Pérez, Reinhard Pichler & Sebastian Skritek (2013): *Static Analysis and Optimization of Semantic Web Queries*. ACM Trans. Database Syst. 38(4), pp. 25:1–25:45, doi:10.1145/2500130. Available at http://doi.acm.org/10.1145/2500130.

[19] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai & Tianyu Wo (2011): *Capturing Topology in Graph Pattern Matching*. Proc. VLDB Endow. 5(4), pp. 310–321, doi:10.14778/2095686.2095690. Available at http://dx.doi.org/10.14778/2095686.2095690.

[20] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai & Tianyu Wo (2014): *Strong Simulation: Capturing Topology in Graph Pattern Matching*. ACM Trans. Database Syst. 39(1), pp. 4:1–4:46, doi:10.1145/2528937. Available at http://doi.acm.org/10.1145/2528937.

[21] Stephan Mennicke, Denis Nagel, Jan-Christoph Kalo, Niklas Aumann & Wolf-Tilo Balke (2017): *Reconstructing Graph Pattern Matches Using SPARQL*. In: Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Rostock, Germany, September 11-13, 2017., p. 152. Available at http://ceur-ws.org/Vol-1917/paper24.pdf.

[22] Tova Milo & Dan Suciu (1999): *Index Structures for Path Expressions*. In: Proceedings of the 7th International Conference on Database Theory, ICDT '99, Springer-Verlag, London, UK, UK, pp. 277–295. Available at http://dl.acm.org/citation.cfm?id=645503.656266.

[23] Mohamed Morsey, Jens Lehmann, Sören Auer & Axel-Cyrille Ngonga Ngomo (2011): *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*. In: The Semantic Web – ISWC 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 454–469, doi:10.1007/978-3-642-25073-6_29.

[24] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis & Themis Palpanas (2016): *Exemplar Queries: A New Way of Searching*. The VLDB Journal 25(6), pp. 741–765, doi:10.1007/s00778-016-0429-2. Available at https://doi.org/10.1007/s00778-016-0429-2.

[25] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu & Jay Banerjee (2015): *RDFox: A Highly-Scalable RDF Store*, pp. 3–20. Springer International Publishing, doi:10.1007/978-3-319-25010-6_1. Available at https://doi.org/10.1007/978-3-319-25010-6_1.

[26] Thomas Neumann & Gerhard Weikum (2009): *Scalable Join Processing on Very Large RDF Graphs*. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, ACM, New York, NY, USA, pp. 627–640, doi:10.1145/1559845.1559911. Available at http://doi.acm.org/10.1145/1559845.1559911.

[27] Jorge Pérez, Marcelo Arenas & Claudio Gutierrez (2009): *Semantics and Complexity of SPARQL*. *ACM Trans. Database Syst.* 34(3), pp. 16:1–16:45, doi:10.1145/1567274.1567278. Available at http://doi.acm.org/10.1145/1567274.1567278.

[28] François Picalausa, Yongming Luo, George H. L. Fletcher, Jan Hidders & Stijn Vansummeren (2012): *A Structural Approach to Indexing Triples*. In: *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*, ESWC'12, Springer-Verlag, Berlin, Heidelberg, pp. 406–421, doi:10.1007/978-3-642-30284-8_34. Available at http://dx.doi.org/10.1007/978-3-642-30284-8_34.

[29] Eric (W3C) Prud'hommeaux & Bristol) Seaborne, Andy (Hewlett-Packard Laboratories (2008): *SPARQL Query Language for RDF*. Technical Report, W3C. Available at https://www.w3.org/TR/rdf-sparql-query/.

[30] Michael Schmidt, Michael Meier & Georg Lausen (2010): *Foundations of SPARQL Query Optimization*. In: *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, ACM, New York, NY, USA, pp. 4–33, doi:10.1145/1804669.1804675. Available at http://doi.acm.org/10.1145/1804669.1804675.

[31] Miao Xie, Sourav S. Bhowmick, Gao Cong & Qing Wang (2017): *PANDA: toward partial topology-based search on large networks in a single machine*. *The VLDB Journal* 26(2), pp. 203–228, doi:10.1007/s00778-016-0447-0.

[32] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Özsu & Dongyan Zhao (2011): *gStore: Answering SPARQL Queries via Subgraph Matching*. *Proc. VLDB Endow.* 4(8), pp. 482–493, doi:10.14778/2002974.2002976. Available at http://dx.doi.org/10.14778/2002974.2002976.

Table 3: Query processing times, query time on the pruned dataset and the query time including pruning time for RDFox. All times are measured in seconds.

Query	t_{DB}	$t_{\text{DB pruned}}$	$t_{\text{DB pruned}} + t_{\text{SPARQLSIM}}$
$\mathscr{L}0$	19.100	1.401	107.852
$\mathscr{L}1$	25900.000	888.000	896.464
$\mathscr{L}2$	161.000	15.690	163.025
$\mathscr{L}3$	0.000	0.000	0.138
$\mathscr{L}4$	0.000	0.000	0.125
$\mathscr{L}5$	0.000	0.000	1.223
$\mathscr{D}_0$	1.400	1.115	5.511
$\mathscr{D}_1$	0.000	0.000	0.002
$\mathscr{D}_2$	1.100	0.003	0.091
$\mathscr{D}_3$	0.620	0.002	0.145
$\mathscr{D}_4$	5.960	3.493	9.722
$\mathscr{D}_5$	3.230	0.016	0.590
$\mathscr{B}_0$	1.468	0.000	0.088
$\mathscr{B}_1$	0.117	0.001	0.023
$\mathscr{B}_2$	0.004	0.001	0.027
$\mathscr{B}_3$	0.001	0.000	0.018
$\mathscr{B}_4$	0.001	0.001	0.019
$\mathscr{B}_5$	0.643	0.022	0.294
$\mathscr{B}_6$	6.416	4.326	11.779
$\mathscr{B}_7$	3.282	1.998	6.319
$\mathscr{B}_8$	0.941	0.000	0.000
$\mathscr{B}_9$	0.000	0.000	0.009
$\mathscr{B}_{10}$	0.758	0.310	1.227
$\mathscr{B}_{11}$	0.099	0.030	0.052
$\mathscr{B}_{12}$	0.119	0.001	0.473
$\mathscr{B}_{13}$	18.750	0.048	0.650
$\mathscr{B}_{14}$	0.348	0.110	0.642
$\mathscr{B}_{15}$	0.104	0.012	0.618
$\mathscr{B}_{16}$	0.033	0.000	0.000
$\mathscr{B}_{17}$	0.000	0.000	0.033
$\mathscr{B}_{18}$	2.981	0.013	0.581
$\mathscr{B}_{19}$	12.830	0.042	0.544
$\mathscr{B}_{20}$	14.410	0.002	0.444
$\mathscr{B}_{21}$	0.793	0.001	0.114

Table 4: Query processing times, query time on the pruned dataset and the query time including pruning time for RDFox. All times are measured in seconds.

Query	t_{DB}	$t_{\text{DB pruned}}$	$t_{\text{DB pruned}} + t_{\text{SPARQLSIM}}$
$\mathscr{L}0$	5.126	2.261	108.712
$\mathscr{L}1$	50.853	0.971	9.435
$\mathscr{L}2$	56.676	26.767	174.102
$\mathscr{L}3$	0.001	0.000	0.138
$\mathscr{L}4$	0.000	0.000	0.125
$\mathscr{L}5$	0.000	0.000	1.223
$\mathscr{D}_0$	0.395	0.359	4.755
$\mathscr{D}_1$	0.001	0.000	0.002
$\mathscr{D}_2$	0.002	0.000	0.089
$\mathscr{D}_3$	0.010	0.003	0.147
$\mathscr{D}_4$	2.148	4.008	10.238
$\mathscr{D}_5$	0.039	0.021	0.595
$\mathscr{B}_0$	0.002	0.000	0.088
$\mathscr{B}_1$	0.003	0.001	0.023
$\mathscr{B}_2$	0.003	0.003	0.030
$\mathscr{B}_3$	0.001	0.002	0.020
$\mathscr{B}_4$	0.001	0.002	0.020
$\mathscr{B}_5$	0.054	0.031	0.303
$\mathscr{B}_6$	3.179	4.886	12.340
$\mathscr{B}_7$	1.082	0.441	4.762
$\mathscr{B}_8$	0.000	0.000	0.000
$\mathscr{B}_9$	0.000	0.000	0.009
$\mathscr{B}_{10}$	0.121	0.099	1.016
$\mathscr{B}_{11}$	0.043	0.009	0.031
$\mathscr{B}_{12}$	0.012	0.003	0.476
$\mathscr{B}_{13}$	0.102	0.056	0.658
$\mathscr{B}_{14}$	0.069	0.064	0.596
$\mathscr{B}_{15}$	0.045	0.013	0.619
$\mathscr{B}_{16}$	0.000	0.000	0.000
$\mathscr{B}_{17}$	0.000	0.000	0.034
$\mathscr{B}_{18}$	0.042	0.026	0.594
$\mathscr{B}_{19}$	0.022	0.013	0.516
$\mathscr{B}_{20}$	0.003	0.001	0.444
$\mathscr{B}_{21}$	0.021	0.005	0.118