Short paper

Towards Planning of Regular Queries with Memory

Thomas Mulder, Nikolay Yakovets, George Fletcher
Technische Universiteit Eindhoven
The Netherlands
[t.mulder,hush,g.h.l.fletcher]@tue.nl

ABSTRACT

We investigate efficient evaluation of regular path queries with memory (RQM), which are an extension of regular path queries (RPQ) with additional constraints on data encountered along a path in a graph. We show how *Waveguide*, a state of the art system capable of planning RPQs, can be extended to facilitate RQM planning. Furthermore, we show that RQM planning is not as trivial as finding a conventional optimal join order and adding-on data constraints. Rather, we showcase that efficient evaluation of RQMs poses a number of non-trivial novel challenges.

1 INTRODUCTION

Regular path queries with memory (RQM) are subgraph-matching queries that allow the definition of constraints on both *topology* and *data* in graphs [3]. RQMs extend the expressive power of regular path queries (RPQ) [1] in many useful ways while maintaining an acceptable PSPACE-complete combined complexity.

An example of a query that is expressible as an RQM but not as an RPQ is to find all pairs of people (x, y) such that x directly or indirectly knows y and all people along the chain of acquaintance have the same age.

The initial study of RQMs [3], provides an extension of regular expressions called regular expressions with memory (REM) that are used to write RQMs, along with a procedure for constructing a *k-register data path automaton* that can be used for the evaluation of an RQM. While k-register data path automata are an excellent tool for the investigation of the expressive power and complexities of RQMs, they do not represent effective query plans, and do not provide opportunities for query optimisation.

We aim to develop the first practical evaluation engine for RQMs [4]. Here we take first steps towards this goal by (1) studying the shortcomings of the proposed automata from a query planning- and evaluation perspective, (2) addressing these shortcomings by proposing a more expressive type of automata that can be used to represent query plans and (3) showing that optimising such plans for *topological*- and *data* constraints are *orthogonal* problems. That is, a query plan that is optimal for evaluating only the topological constraints of a query on a particular graph, and another query plan that is optimal for evaluating the topological- *and* data constraints of the same query on the same graph, need not consider the topology of the query in the same order.

We extend Waveguide [5], a cost-based optimizer for property paths which builds query plans called *waveplans* that guide query evaluation. Waveplans are based on automata, which allows us to combine concepts from waveplans and k-register data path automata to obtain k-register waveplans.

2 REGULAR QUERIES WITH MEMORY

Data graphs are defined over a finite alphabet Σ and a countably infinite set of data values $\mathcal{D}$ as a triple $G = \langle V, E, \rho \rangle$, where:

- V is a finite set of nodes;
- $E \subseteq V \times \Sigma \times V$ is a set of labelled edges; and
- $\rho : V \rightarrow \mathcal{D}$ is a function that assigns a data value to each node in V.

To write regular expressions that can specify paths in data graphs, regular expressions with memory (REMs) over a finite alphabet Σ and set of variables $x_1, \ldots, x_k$ are introduced and defined by the grammar

$$e := \epsilon \mid a \mid e + e \mid e \cdot e \mid e^+ \mid e[c] \mid \downarrow \overline{x}.e \mid (e), \tag{1}$$

where $a \in \Sigma$, c is a condition and $\overline{x}$ a tuple of variables from $x_1, \ldots, x_n$. A condition, in turn, is defined by the grammar

$$c := x_i^= \mid x_i^{\neq} \mid z^= \mid z^{\neq} \mid c \wedge c \mid c \vee c \mid \neg c \mid (c),\ 1 \leq i \leq k, \tag{2}$$

where z is a data value in $\mathcal{D}$, also referred to as a constant.

As an example, consider the following REM:

$$(\texttt{owns} \cdot \downarrow x_1.\texttt{isLocatedIn})^+ \cdot \texttt{hasCapital}[x_1^=] \tag{3}$$

This REM specifies paths where we encounter at least one sequence of two edges labelled owns and isLocatedIn, followed by an edge labelled hasCapital. Additionally, the data value associated with the source of an edge labelled isLocatedIn is stored in the first register, which is subsequently compared to the data value associated with the target of an edge labelled hasCapital.

A regular query with memory is defined as an expression of the form $Q := x \xrightarrow{e} y$ where e is an REM, and x and y are variables that are mapped to nodes in a data graph. Hence, the evaluation of Q amounts to finding pairs $(u, v) \in V \times V$ such that there exists a path from u to v that adheres to e. See Section 3.3 of [3] for the formal semantics of REMs and RQMs.

3 QUERY PLANNING

Query planning in the most general sense, is the process of finding an ordering of operations that, when executed, produce the solution to the given query. In the context of regular queries with memory, this means finding an ordering of *edge labels*, *assignments*, *conditions* and *projections*.

3.1 Automata as Query Plans

Query plans for relational database systems are often represented as trees. Due to the recursive nature of RQMs introduced by the Kleene plus ($^+$) operator, it is more convenient to represent query plans for RQMs as *automata* instead.

Regular Data Path Automata (RDPA) [3] have been proposed as a representation of RQMs. These RDPAs, however, capture the operations necessary for the evaluation of an RQM $Q = x \xrightarrow{e} y$ only in the order corresponding to the left-deep parsing of e. Hence, RDPAs are not a suitable formalism for representing query plans for RQMs.

Instead, we will consider *Waveguide* [5], a state of the art query optimiser for RPQs. It represents query plans as *waveplans*, a automaton-based formalism that allows for a rich variety of query plans, due to its *inverse transitions* and *transitions over views*. Conceptually, a waveplan consists of one or more *wavefronts*, which are automata that are used to compute (part of) the solution to a query. We extend wavefronts with assignments, conditions and projection over data values and registers.

Inverse- and view transitions. Let Σ be a finite alphabet, and L a finite set of state labels disjoint from Σ. We define $\Sigma^* = \bigcup_{a \in \Sigma \cup L} \{/a, a/\}$ as a set of labels. A transition with label $/a \in \Sigma^*$ is said to *append* edges labelled a to an intermediate result, whereas a transition labelled $a/ \in \Sigma^*$ is said to *prepend* edges labelled a to an intermediate result. The latter is referred to as an *inverse* transition. Inverse transitions allow our plans to represent many different orders of edge labels, such as a right-deep order.

A transition with label $/2 \in \Sigma^*$ appends paths computed by the state with label 2 to an intermediate result. Such a transition is called a *transition over a view*. Transitions over views further extend the orders of edge labels our plans can express by including *bushy* plans.

Projection. *Data paths* are defined as a sequence of interleaving nodes and edge labels that always start and end with a node [3]. Consider the REM e from (3) and a data path:

$$\pi = v_1 \, \texttt{owns} \, v_2 \, \texttt{isLocatedIn} \, v_3 \, \texttt{hasCapital} \, v_4$$

Checking whether or not π is accepted by e in a right-deep order means first finding edges labelled `hasCapital`, then amongst those finding edges that are preceded by edges labelled `isLocatedIn`, etc. This is a valid order of evaluating the edge labels and may be more efficient than a left-deep order, depending on the input graph. This order provides a problem with respect to the assignment- and condition in this REM. Namely, we would like to check that $\rho(v_4) = x_1$ where x_1 is the value in the first register. We can only do so once the first register has been assigned the value $\rho(v_2)$. Hence, to make this ordering of the edge labels work, $\rho(v_4)$ will have to be stored until the assignment has been made. To indicate which data- and register values must be stored at which point during query evaluation, we associate with each state in our automata finite sets $P_{\mathfrak{D}} \subset \mathbb{N}$ and $P_r \subset \mathbb{N}$ that contain the positions of nodes in a path and the indices of registers that must be kept, respectively. For instance, to indicate that $\rho(v_4)$ and x_1 (i.e. the value of the first register) must be kept, we would set $P_{\mathfrak{D}} = \{4\}$ and $P_r = \{1\}$.

3.2 k-Register Waveplans

We will refer to the extension of waveplans and wavefronts with assignments, conditions and projections as *k-register waveplans* and *k-register wavefronts*, respectively.

Let Σ^* be a finite labelling alphabet, k a natural number and C a finite set of conditions. Formally, a *k-register wavefront* is a tuple $w_l = \langle l, S, Q, q_0, \Pi_{\mathfrak{D}}, \Pi_r, \delta, F, \tau_0 \rangle$, where

- l is a wavefront label,
- S is a seed,
- Q is a set of states,
- q_0 is the starting state $q_0 \in Q$,
- $\Pi_{\mathfrak{D}} : Q \to 2^{\mathbb{N}}$ is a function assigning a projection $P_{\mathfrak{D}}$ of path positions corresponding to data values to each state,
- $\Pi_r : Q \to 2^{\mathbb{N}}$ is a function assigning a projection P_r of register indices to each state,

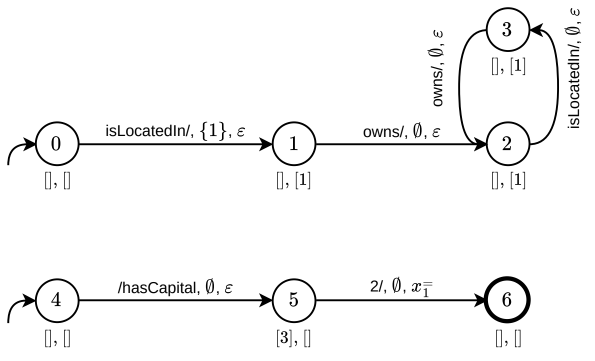

Figure 1: A k-register waveplan for (3)

- δ is a transition relation $\delta : Q \times \Sigma^* \times 2^{[k]} \times C \times Q$,
- $F \subseteq Q$ is a set of accepting states, and
- $\tau_0 \in \mathfrak{D}_\perp^k$ is the initial configuration of the registers.

A tuple $d = (q_1, a, K, c, q_2) \in \delta$ consists of source- and target states q_1 and q_2, label a, a set $K \subseteq \{1, ..., k\}$ indicating which registers are to be assigned a value during this transition and a condition c which is to be checked during this transition.

A *k-register waveplan* p is an ordered set of wavefronts. Consider any pair of wavefronts $w, w' \in p$ such that
$w = \langle l, S, Q, q_0, \Pi_{\mathfrak{D}}, \Pi_r, \delta, F, \tau_0 \rangle$ and
$w' = \langle l', S', Q', q_0', \Pi'_{\mathfrak{D}}, \Pi'_r, \delta', F', \tau_0 \rangle$. Then p defines an order $<_p$ on wavefronts as follows:

$$\forall w, w' \in p \mid w <_p w' \; : \; l' \notin S \land l' \notin L$$

Given this order, lower wavefronts cannot use labels of higher wavefronts in their seeds or transitions.

The role of *seeds* in Waveguide is quite complex. Here it suffices to say that seeds are necessary to ensure semantically correct plans when dealing with multiple wavefronts in a waveplan, or single wavefronts implementing a closure over an expression.

Figure 1 shows a k-register waveplan for the REM from (3). It consists of two wavefronts. The first wavefront computes the result to the expression $e' = (\texttt{owns}/\!\downarrow x_1.\texttt{isLocatedIn})^+$ in a right-deep order. It assigns a value to the first register (i.e. x_1 is set to the data value at the source of an edge labelled `isLocatedIn`). The second wavefront computes the result to (3) by first computing $e'' = \texttt{hasCapital}$, storing the data value at the target of edges labelled `hasCapital` in state 5. The result of state 2 is prepended to the result of state 5, and x_1 is checked for equality against the data value at the end of the resulting paths (i.e. at the target of an edge labelled `hasCapital`).

3.3 Topological Order

Let e be a regular expression with memory. We will refer to t_e as the topology of e which is a regular expression obtained by recursively replacing

- every sub-expression $e_1[c]$ of e by e_1, and
- every sub-expression $\downarrow \overline{x}.e_1$ of e by e_1

such that t_e no longer contains assignments and conditions.

We can now define the order in which a k-register waveplan p considers the topology of a query as follows. Consider a wavefront $w \in p$. We define the *topological order* λ_w of w as a sequence $\langle t_0, ..., t_n \rangle$ where t_i is the sub-expression of t_e for which the state with label i in p returns an intermediate result. We

ignore all states that are part of a cycle in w, except for accepting states, when constructing any sequence λ_w. We define the topological order λ_p of a k-register waveplan p as a sequence of sequences $\langle\langle t_0, ..., t_l\rangle, ..., \langle t_m, ..., t_n\rangle\rangle$, where each inner sequence corresponds to the topological order of the wavefronts that make up p, in the order defined by $<_p$. Consider the waveplan in Figure 1. Its topological order is
$\langle\langle \epsilon, \texttt{isLocatedIn}, (\texttt{owns/isLocatedIn})^+\rangle$,
$\langle \epsilon, \texttt{hasCapital}, (\texttt{owns/isLocatedIn})^+/\texttt{hasCapital}\rangle\rangle$.

3.4 Query Evaluation

As per [5], query evaluation is done using a procedure based on breadth-first search in which a waveplan guides the search (i.e. determines which edges are to be explored based on their label and possibly the satisfaction of conditions) until a fix-point is reached.

Edge walks. The metric by which we will judge the performance of a plan on the query evaluation task is that of the total number of *edge walks*. Every distinct tuple that is added to the queue during the search is considered one edge walk. Notice that a tuple that is added to the queue is an n-tuple with $n \geq 3$. A tuple contains at least a pair of nodes representing the end-points of a path and the state of the waveplan that produced this pair. Additionally, for a query $Q := x \xrightarrow{e} y$, a tuple can also contain up to k register values and up to l data values, where l is bounded by the number of sub-expressions of e of the shape $e_1[c]$. Hence $n \leq k + l + 3$.

Optimality of query plans. - Consider a pair of RQMs $Q := x \xrightarrow{e} y$ and $Q' := x \xrightarrow{t_e} y$. These queries are topologically equivalent (since t_e is the topology of e), but Q may contain assignments and conditions. Let P_Q and $P_{Q'}$ be sets of k-register waveplans for Q and Q', respectively. For every plan $p \in P_Q$ there exists a plan $q \in P_{Q'}$ such that $\lambda_p = \lambda_q$ because Q and Q' are topologically equivalent. Hence, we can construct a relation $R \subset P_Q \times \mathbb{N} \times P_{Q'} \times \mathbb{N}$ where $(p, n, q, m) \in R$ if and only if $\lambda_p = \lambda_q$ and be assured that there exists a tuple $r \in R$ for every $p \in P_Q$ such that p is part of r. The values n and m denote the total number of edge walks produced by p and q, respectively.

We can define sets $OPT_Q \subseteq P_Q$ and $OPT_{Q'} \subseteq P_{Q'}$ as

$$OPT_Q = \{\, p \mid \exists(p, n, q, m) \in R \land \forall(p', n', q', m') \in R : n \leq n' \,\} \tag{4}$$

$$OPT_{Q'} = \{\, q \mid \exists(p, n, q, m) \in R \land \forall(p', n', q', m') \in R : m \leq m' \,\} \tag{5}$$

That is, OPT_Q and $OPT_{Q'}$ are the sets of plans for Q and Q' that produce a minimal number of edge walks. We say that any $q \in OPT_{Q'}$ is *optimal with respect to topology*, and any $p \in OPT_Q$ is *optimal with respect to topology and data*.

We define a *simple* query Q as a query where

$$\forall(p, n, q, m) \in R \mid q \in OPT_{Q'} \Rightarrow p \in OPT_Q$$

which states that a simple query is one where optimality with respect to topology guarantees optimality with respect to topology and data.

We investigate the performance difference between a plan $p_2 \in OPT_Q$ and a plan $p_1 \notin OPT_Q$ for which there exists $(p_1, n_1, q_1, m_1) \in R$ such that $q_1 \in OPT_{Q'}$. Such an investigation will yield insights into the performance improvements that are neglected when query plan- and graph topology are assumed to be the determining factors in query performance.

To this end, we define a performance ratio between the edge walks produced by such plans p_1 and p_2. Formally, φ is defined over Q (from which R, OPT_Q and $OPT_{Q'}$ are derived) as

$$\varphi(Q) \begin{cases} \frac{n_1}{n_2}, & \text{if } \exists(p_1, n_1, q_1, m_1), (p_2, n_2, q_2, m_2) \in R \mid \\ & \quad q_1 \in OPT_{Q'} \land p_1 \notin OPT_Q \land p_2 \in OPT_Q \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

Notice that $\varphi(Q) = 1$ if and only if Q is simple.

4 EXPERIMENTAL SETUP

In order to show that optimising k-register waveplans with respect to the topology of a query is orthogonal to optimising these plans with respect to the query's data constraints, we will construct a workload $\mathcal{W}$ consisting of pairs (Q, G) where Q is a regular path query with memory, and G is a data graph. We will count the number of pairs (Q, G) such that Q is *simple* on G, and investigate the average- and worst-case improvements in performance that are neglected when the topology of a query plan and graph are assumed to be the determining factors in query performance.

4.1 Query Pattern

The queries in $\mathcal{W}$ will be based on instances of the pattern:

$$((a\cdot \downarrow x_1.b)^+) \cdot c[x_1^=] \tag{7}$$

The motivation for the choice of this particular pattern is three-fold:

(1) it contains interactions with data through a register (i.e. an assignment and condition) *inside* of a closure,
(2) it is simple in terms of the number of edge labels it contains and registers it uses, and
(3) its interactions with data apply to nodes that are neither:
  - part of the pairs in the result of the query evaluation problem, or
  - guaranteed to have the same data value associated with them

The first point is important because any pattern that does not interact with data at all, or does so only outside of closures can be modelled as a (C)RPQ [1], and is therefore not an example of the increased expressive power of RQMs. The second point is more practical in that it is possible to find many different instances of a pattern which contains few edge labels and uses few registers in real graph data. Thirdly, a pattern such as $(\downarrow x_1.a\cdot b\cdot c[x_1^=])^+$ would also satisfy the first two points. However, since all its interactions with data pertain to the data values in nodes that are either part of the pairs in the result of the query evaluation problem, or pertain to nodes that have the same data value associated with them (i.e. the nodes with an incoming edge labelled c and an outgoing edge labelled a in the closure), such queries are too selective in practice.

Seventeen concrete combinations of edge labels for a, b and c are obtained from the semantic knowledge graph Yago2s [2]. Example instances are:

- $(\texttt{isLocatedIn}\cdot \downarrow x_1.\texttt{dealsWith})^+ \cdot \texttt{hasCapital}[x_1^=]$
- $(\texttt{owns}\cdot \downarrow x_1.\texttt{isLocatedIn})^+ \cdot \texttt{hasCapital}[x_1^=]$
- $(\texttt{isLocatedIn}\cdot \downarrow x_1.\texttt{owns})^+ \cdot \texttt{isConnectedTo}[x_1^=]$

4.2 Data Graphs

A data graph $G = \langle V, E, \rho \rangle$ is extracted from the semantic knowledge base Yago2s. Yago2s consists of RDF-triples, derived from Wikipedia, WordNet and GeoNames [2].

Let U denote the set of all distinct subjects and objects in Yago2s' RDF-triples. Similarly, let Σ denote the set of all distinct predicates in Yago2s' RDF-triples. We identified $P \subset \Sigma$ as a set of 13 predicates such that for all triples (s, p, o) with $p \in P$ it holds that o is a numerical value. No other predicates in the Yago2s data set could be identified that co-occur with edge labels for (7) and have numerical values. The range of values over all objects o was categorized into three equally sized categories. Thus, the data domain for each $p \in P$ is set to $\{0, 1, 2\}$ where we interpret the data values 0, 1 and 2 as low, medium and high, respectively. Consider an RDF-triple (s, p, o) where $p \in P$. Let $\gamma(o) \in \{0, 1, 2\}$ denote the category that o was assigned to.

Since $|P| = 13$ but the data graph model only allows a single data value per node, we can construct multiple data graphs G_i from Yago2s. Note that the query pattern from (7) requires data values at the source of edges labelled b and at the target of edges labelled c. Hence, for all 13^2 pairs from $P \times P$ we construct a data graph G_i.

Let (q_1, q_2) be an arbitrary pair from $P \times P$. To construct G_i we:

- add a node v to V for every $u \in U$. Let $\eta(u) = v$ denote the node in G that corresponds to u;
- for every triple (s, p, o) with $p \in \Sigma - P$ we add $(\eta(s), p, \eta(o))$ to E;
- for every pair of triples $(s, p, o), (s, b, o')$ where $p \in P$ and b in some instance of (7) we set $\rho(\eta(s)) = \gamma(o)$;
- for every pair of triples $(s, p, o), (s', c, s)$ where $p \in P$ and c in some instance of (7) we set $\rho(\eta(s)) = \gamma(o)$;
- for every $v \in V$ where v is not yet in the domain of ρ we set $\rho(v) = 3$.

Thus the labelling alphabet of each G_i is $\Sigma - P$ and the set of data values $\mathcal{D}$ is $\{0, 1, 2, 3\}$. When checking equality for two data values $d_1, d_2 \in \mathcal{D}$, we will maintain that $d_1 = 3 \vee d_2 = 3 \Rightarrow d_1 \neq d_2$. That is, if neither of the data values were obtained from Yago2s we consider them unknown and therefore unequal.

4.3 Query Workload

The combination of 17 instances of query pattern (7) and the $13^2 = 169$ data graphs allows for a maximum workload size of $17 * 169 = 2873$. However, many of these combinations will yield empty result sets on the query evaluation problem because there existed no RDF-triples in Yago2s that produce data values from $\{0, 1, 2\}$ for the source- or target nodes of edges labelled b or c, respectively. Instead, the workload $\mathcal{W}$ consists of 579 distinct pairs of RQMs and data graphs where the combination of edge labels for a, b and c from $\Sigma - P$ and properties q_1 and q_2 from P resulted in at least one $v \in V$ where $\rho(v) \in \{0, 1, 2\}$.

5 RESULTS

Out of the 579 pairs of regular queries with memory and data graphs (Q, G) only 87 queries Q are simple on G (as shown in Table 1). Because a large majority of the queries is not simple we conclude that, for the given query pattern and data set, the topology of a data graph and RQM are not, by themselves, determining factors in the performance of query plans.

On average, k-register waveplans that are optimal with respect to topology (but not necessarily optimal with respect to data) produce close to 2.5 times the number of edge walks to evaluate a query as do those k-register waveplans that are optimal with respect to topology and data.

	simple	non-simple	total
# of queries	87	492	579
% of queries	15.03%	84.97%	100%

Table 1: The number- and percentage of simple and non-simple queries

	min	max	mean	std
$\varphi(Q)$	1.0	29.08	2.42	5.30

Table 2: A breakdown of the values for $\varphi(Q)$ over $\mathcal{W}$

Moreover, for the worst-case ratio $\varphi(Q)$ observed in $\mathcal{W}$, a plan that is optimal with respect to topology performed just over 29 times more edge walks than a plan for the same query, on the same graph, that is optimal with respect to topology and data. A breakdown of the minimum, maximum, mean and standard deviation of $\varphi(Q)$ over $\mathcal{W}$ is presented in Table 2.

A caveat to the results obtained in this way is the following; the evaluation procedure employs a time-out mechanism whereby a query plan that has produced more edge walks than the best performing (i.e. fewest edge walks producing) plan thus far for the same query is terminated, even if it has not yet completed its evaluation. Hence, the observed ratios are a lower-bound on the actual performance ratios.

We have presented evidence of the orthogonality that exists between optimising query plans for RQMs with respect to topology and data, showing that such optimization involves novel- and non-trivial challenges that go beyond finding an optimal join order for edge labels. Ignoring this orthogonality leads to significant decreases in performance, both on average and in worst-case scenarios.

6 CONCLUDING REMARKS

In our experimental study we found that (1) a large majority of queries is not simple, from which we can conclude that optimising for topology and data are orthogonal problems; (2) on average, plans that are optimal w.r.t. topology (but not necessarily w.r.t. data) perform 2.5 times worse than plans that are optimal overall; and, (3) plans that are optimal w.r.t.topology only can perform up to 29 times worse than plans that are optimal overall

Looking ahead, a main direction of future work is to continue our study of RQM query optimization in the context of a fully-fledged property graph query engine.

REFERENCES

[1] A. Bonifati, G. Fletcher, H. Voigt, and N. Yakovets. 2018. *Querying Graphs.* Morgan & Claypool Publishers.

[2] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artif. Intell.* 194 (Jan. 2013), 28–61.

[3] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. 2016. Querying Graphs with Data. *J. ACM* 63, 2, Article 14 (March 2016), 53 pages.

[4] Thomas Mulder. 2019. *Regular Queries with Memory: From Theory to Practice.* MSc thesis. Technische Universiteit Eindhoven.

[5] Nikolay Yakovets, Parke Godfrey, and Jarek Gryz. 2016. Query planning for evaluating SPARQL property paths. In *Proceedings of the 2016 International Conference on Management of Data.* ACM, 1875–1889.