Annales Univ. Sci. Budapest., Sect. Comp. 42 (2014) 283–296

THE Pros AND Cons OF RDF STRUCTURE INDEXES

Balázs Pinczel, Dávid Nagy, Attila Kiss

(Budapest, Hungary)

Dedicated to Professor András Benczúr on the occasion of his 70^{th} birthday

Communicated by Le Manh Thanh

(Received June 1, 2014; accepted July 1, 2014)

Abstract. During recent years, BigData trends resulted in an increasing number of large datasets. As a result, Semantic Web technologies not only have to face the difficulties stemming from sheer data size, but – arising from its intentions of interconnecting all relevant pieces of knowledge – other problems can appear because of the increased structural complexity of the data. As an answer to these challenges, there are several approaches to support the technologies by utilizing structure indexes in order to increase the efficiency of query evaluation. However, in our experience, the benefits of these methods depend largely on the structural properties of the data. In this paper we summarize our experiments conducted on a parameterizable dataset, with the intention of characterizing the relationship between the structural complexity of the data and the possible benefits of using a structure index.

Key words and phrases: bisimulation, structure indexes, performance measurement, Semantic Web, RDF, SPARQL

1998 CR Categories and Descriptors: H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing — Indexing methods

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013).

This work was completed with the support of the Hungarian and Vietnamese TET (grant agreement no. TET 10-1-2011-0645).

1. Introduction

The aim of the Semantic Web [4] is to move towards a "web of data", where the information is not (only) represented in the form of text-based documents interconnected with hyperlinks, but where the pieces of data are directly connected to each other, forming a complex network of knowledge. The main advantage of such a network is that the information can be readily and more easily interpreted, processed and combined by machines, without the need to perform natural language processing or text mining. A practical, daily-used application is the enrichment of web search engines with semantic information, improving the quality of the search results, compared to the results of a conventional string matching over the "web of documents".

To achieve this vision, the Semantic Web uses the graph-based RDF [5] (Resource Description Framework) as a data model. In RDF, knowledge is modeled as a collection of statements in the form of $(subject, predicate, object)$ triples. The following example shows three triples describing Loránd Eötvös – one statement each for his name, birth date and field – serialized using the Turtle [3] syntax. (The example is part of DBpedia [10], a public dataset containing information extracted from Wikipedia.)

```
dbpedia:Loránd_Eötvös  foaf:name              "Loránd Eötvös" .
dbpedia:Loránd_Eötvös  dbpedia-owl:birthDate  "1848-07-27" .
dbpedia:Loránd_Eötvös  dbpedia-owl:field       dbpedia:Physics .
```

An RDF dataset can also be considered as a graph, where each (s, p, o) triple forms a directed edge with label p, pointing from node s to node o. In this paper, we use a notation which emphasizes the "set-of-triples" approach, yet hints at the graph-like nature as well: $G \subseteq V_G \times L \times V_G$ denotes an RDF graph, given as the set of triples defining its edges, where V_G denotes the vertices of the graph, and L is the set of possible edge labels. Querying of RDF data is done with SPARQL [7], a query language based on graph pattern matching. A pattern can be defined similarly to RDF statements, with the additional possibility of using variables as the components of a triple. This is illustrated in the example query below, which returns the birth date of Loránd Eötvös.

```
SELECT ?birthDate
WHERE {
    ?eotvos  foaf:name              "Loránd Eötvös" .
    ?eotvos  dbpedia-owl:birthDate  ?birthDate .
}
```

Although SPARQL allows a wide variety of additional language constructs to increase expressive power [2] (along with the complexity of evaluation [12,

14]), the basic graph pattern matching constitutes the core of the language. Since this feature implements the ability to impose constraints on the structure of the selected data, this is the area where most structure indexes focus, as well as this paper.

As the technologies of the Semantic Web are spreading, both the number and size of RDF datasets are increasing. This trend introduced the necessity of aiding the evaluation of SPARQL using structure indexes, applying similar techniques to the ones successfully utilized in the world of XML and XPath. However, in our experience, the applicability of this method depends largely on the structural characteristics of the data, which can even lead to situations where using an index actually slows down the evaluation process. In this paper, we summarize our experiments regarding the benefits of utilizing a structure index on a parameterizable dataset, showing that even a minimal amount of structural complexity (found commonly in most real-life datasets) can lead to practically useless indexes.

The rest of the paper is structured as follows. In Section 2, an overview of previous works related to indexing techniques is presented. Section 3 describes the details of our experiments, including the dataset, the indexing technique used, and the evaluated queries. Section 4 is devoted to our findings as results of the experiments, while Section 5 summarizes our conclusions and discusses our plans for future research.

2. Related work

Prior to Semantic Web technologies, structure indexes were applied in the world of semi-structured databases [9, 8] to aid the evaluation of XPath queries over XML data. Here, the data can be viewed as an edge-labeled graph, where queries select vertices that can be reached from a designated root node by walking a path whose sequence of edge-labels satisfy the pattern specified in the path-expression of the query. Bisimulation-based indexes help the evaluation by merging those nodes, which have the same pattern of incoming or outgoing paths. Thus, an index graph is constructed in a such a way, that the evaluation of all query patterns yield the same results on the data graph and the (usually much smaller) index graph.

When it comes to indexing RDF data, there are some approaches which focus on "traditional" (i.e. non-structural) indexing, offering fast lookup of triples while avoiding scans of the whole dataset. Of these, a notable approach is Hexastore [17], which essentially uses six differently ordered versions of the data, according to the six permutations of the *subject*, *predicate*, and *object*

components of the triples.

Also, there are RDF indexes which capture the structural properties of the data graph by some other means than bisimulation, nevertheless these are generally based on similar principles, i.e. extracting common patterns in paths. For example, Matono et al. [11] propose an indexing scheme which builds suffix arrays from the possible paths extracted from subgraphs of the dataset, in order to facilitate answering queries defined by path expressions. Stuckenschmidt et al. [15] also focus on recurring patterns by building a hierarchy of join indexes based on paths, to solve the problem of decomposing queries (and joining retrieved answers) over distributed RDF repositories.

During our experiments we followed the technique described by Tran & Ladwig [16]. They build a structure index based on forward-backward bisimulation, and use a two-step method to evaluate queries defined as basic graph patterns. Further details of this approach are explained in Section 3, where we elaborate on the specifics of our experiments. Similar to our goals, Alzogbi & Lausen [1] also investigate the benefits of structure indexes. They compress the data graph by performing an agglomerative clustering step on partitions obtained by bisimulation. However, their focus is on the size reduction, while we considered the practical aspects of query evaluation as well.

3. Experiments

Motivation. When creating bisimulation-based structure indexes, the quality of the resulting index structure highly depends on the structural complexity of the data graph. To illustrate this, we included in Table 1 the results of indexing two datasets from the opposite ends of the spectrum.

Dataset	**Data nodes**	**Index nodes**	**Query time (without index)**	**Query time (with index)**
SP^{2}Bench	61107	58506	2.031 s	2.547 s
Triangles	90000	3	0.875 s	1.532 s

Table 1. The results of using an index for two datasets with highly different structural characteristics.

The SP^{2}Bench dataset [13] is used as a basis of a SPARQL performance benchmark. It contains an artificial publication database mirroring the characteristics of real-world ones. Our dataset contained 100000 triples generated with the SP^{2}Bench tool. The query applied here is a simple search for the authors and titles of all inproceedings (based on $Q2$ of the SP^{2}Bench queries).

The data has a rather complex structure, owing to the fact that it contains all kinds of publications with different attributes, each citing numerous other publications, etc. Because of this, only a few nodes could be merged based on the similarity of their incoming and outgoing paths, thus the index is almost as large as the original graph. On the other hand, our trivial Triangles dataset shows a high degree of regularity. It consists of 90000 nodes, forming 30000 triangle-shaped isolated subgraphs. For a dataset like this, the index always contains only 3 nodes, regardless of the size of the data graph. Yet, in both cases the use of the index makes the evaluation of the query slower. This anomaly is caused by the fact that – contrary to the world of XML indexes – query evaluation here consists of not only an evaluation on the index graph, but has a second step in order to obey additional non-structural constraints specified in the query (e.g. the equality of node values with given constants). In the case of SP^2Bench – as the index is almost as large as the data – the first step (evaluation on the index graph) is not fast enough to make up for the additional time required by the second step. For Triangles, the first step is reasonably quick, but since each index node corresponds to 30000 data nodes, the search space is not reduced enough, and the slowness of the second step cancels the benefits of the index. This illustrates that either too complex or too simple structures can result in practically useless indexes. More about the details of indexing and the two-step query evaluation can be read in Section 3.2.

3.1. Experimental datasets

To explore the area between the two extremes of Table 1, we used a parameterizable dataset in our experiments, which significantly simplifies the structure of the real-world datasets, yet allows a tunable structural complexity in order to demonstrate its effects on the efficiency of using an index. The data describes a hypothetical social network using the FOAF [6] (Friend of a Friend) vocabulary. For each person in the network we record his name and three additional optional attributes: phone number, e-mail address, and homepage. The subgraph describing a person forms a star shape, with his attributes attached to a central node representing his unique identifier, as shown in the following example.

```
person1  foaf:name      "Person #1" .
person1  foaf:phone     "tel:000001" .
person1  foaf:homepage  <http://example.org/person1> .
person1  foaf:mbox      <mailto:person1@example.org> .
```

The datasets are generated using a randomized process, in which the presence or absence of an optional attribute of a person is controlled with a fixed

probability of 0.8, independently of the existence of his (or anyone's) other attributes. The structural complexity is introduced with edges of the form $(person_1, knows, person_2)$, connecting the centers of two stars, representing an asymmetrical friendship relation between two people. For each person, a single outgoing edge of this type is generated with probability p_{knows} (the parameter of the data model), the target of which is selected uniformly at random from among the other people. Thus, a person either has zero or one outgoing *knows*-edge. Also, $p_{knows} = 0$ corresponds to a dataset consisting of isolated stars, while $p_{knows} = 1$ yields a social network where everyone (asymmetrically) knows exactly one other person. This suffices for our purpose, since our goal with the dataset is not to accurately model real-world social networks, but to simplify and model structural properties of real-world Semantic Web datasets for the purpose of testing the benefits of indexing. As it turns out, even as few as one *knows*-edge per person can be more than enough to render indexing efforts futile.

3.2. Indexing approach

In our experiments, we used the indexing technique and query evaluation procedure introduced by Tran & Ladwig [16]. The index they propose is based on a variation of the forward-backward bisimulation adapted to RDF datasets.

Definition 3.1 (Forward-backward bisimulation of an RDF graph)**.** *Let $G \subseteq V_G \times L \times V_G$ be and RDF graph given as a set of RDF triples, and $R \subseteq V_G \times V_G$ be a binary relation over the vertices of G. Then R is called a forward-backward bisimulation of G if and only if the followings hold.*

1. *For all $s_1, s_2 \in V_G$ and $p \in L$*
   - *$(s_1, s_2) \in R$ and $(s_1, p, o_1) \in G$ implies that there exists $o_2 \in V_G$ such that $(s_2, p, o_2) \in G$ and $(o_1, o_2) \in R$;*
   - *$(s_1, s_2) \in R$ and $(s_2, p, o_2) \in G$ implies that there exists $o_1 \in V_G$ such that $(s_1, p, o_1) \in G$ and $(o_1, o_2) \in R$.*
2. *For all $o_1, o_2 \in V_G$ and $p \in L$*
   - *$(o_1, o_2) \in R$ and $(s_1, p, o_1) \in G$ implies that there exists $s_2 \in V_G$ such that $(s_2, p, o_2) \in G$ and $(s_1, s_2) \in R$;*
   - *$(o_1, o_2) \in R$ and $(s_2, p, o_2) \in G$ implies that there exists $s_1 \in V_G$ such that $(s_1, p, o_1) \in G$ and $(s_1, s_2) \in R$.*

Corollary 3.1. *Let $\sim$ denote a binary relation over the vertices of G, such that for all $v_1, v_2 \in V_G : v_1 \sim v_2$ if and only if there exists a forward-backward*

bisimulation R with $(v_1, v_2) \in R$. Then the relation $\sim$ is itself a forward-backward bisimulation.

Similarly to most bisimulation-based indexing techniques, the index graph itself is constructed by merging the nodes belonging to the same partition according to the equivalence relation $\sim$ defined in Corollary 3.1.

Definition 3.2 (Index graph of an RDF graph)**.** *Let $G \subseteq V_G \times L \times V_G$ be an RDF graph, and $\sim$ its forward-backward bisimulation as defined in Corollary 3.1. Then the index graph $I \subseteq V_I \times L \times V_I$ of data graph G is the graph constructed the following way.*

- *The vertices of I are the equivalence classes of $\sim$.*
- *An index edge $(s^{\sim}, p, o^{\sim})$ with label $p \in L$ exists between index nodes $s^{\sim}, o^{\sim} \in V_I \subseteq \mathcal{P}(V_G)$ if and only if there are vertices $s, o \in V_G$, such that $(s, p, o) \in G$ is an edge of the data graph, and the vertices are members of the corresponding equivalence classes: $s \in s^{\sim}$ and $o \in o^{\sim}$.*

This way, the data graph is compressed by reducing the number of vertices and edges, while the structural patterns of the original graph are still preserved, making it possible to compute graph pattern matches using the compressed index graph. A basic SPARQL graph pattern is evaluated in two steps. In the first step, *index matches* are computed, then in the second step *data matches* are created by combining data elements retrieved in the first step.

Index matches are computed in the first step by treating the index graph as an ordinary data graph, and evaluating the query against it, using standard procedures. Since the nodes in this case represent equivalence classes of the data graph, the evaluation process has a minor difference in how constants are treated: an index node matches a constant of the graph pattern, if the value of the constant is a member of the equivalence class represented by the index node. The results of this step are mappings from variables to index nodes (equivalence classes of data nodes).

The second step in the evaluation has the task of computing data matches based on the index matches of the first step and the information of the data graph. One index match is a mapping from variables to index nodes (i.e. sets of data nodes) which satisfy the structural patterns of the query. For example, when searching for people and their e-mail addresses in our experimental dataset, an index match contains a set of people and a set of e-mail addresses. So basically, in this case, the goal of this step is to pair people with their e-mail addresses. Additionally, false results have to be removed, which only satisfy the structural patterns (i.e. edge labels) from the query, but fail to match node value constants. Tran & Ladwig [16] proposed an efficient join-based iterative

procedure to accomplish this task. During this step "traditional" indexes are used to speed up the lookup of triples from the data graph. As a result of this process, we obtain the final query results as mappings from variables to data nodes. Note that in the case of XPath, a second step like this is unnecessary because *a)* values are only captured at one point in the pattern (at the end of the path), so there are no multiple variables whose index matches have to be combined, and *b)* there are no false results to be removed, because in the basic case all restrictions of the query apply to edge labels, and thus are satisfied during the first step.

There is an opportunity for some optimization in the second step, as shown by Tran & Ladwig [16]. Using the structure index as the basis of evaluation in the first step results in narrowing down the space of possible solutions by keeping only matches that meet the structural requirements specified by the graph pattern. As a consequence, those parts of the query that would not add information to the results – but only serve the purpose of specifying a structural pattern – have already been dealt with and can be removed. Specifically, those tree-shaped parts of the pattern that only consist of variables not appearing in the SELECT list can be pruned away before the second step.

3.3. Measurements

During our experiments, we generated RDF datasets using various p_{knows} values, each dataset describing a hypothetical social network of $n = 100000$ people. Then, we built structure indexes using the method described above. As a basis for evaluating the benefits of utilizing indexes, we used two kinds of metrics: the compression ratio of the index, and the evaluation time of basic queries. For an RDF graph $G \subseteq V_G \times L \times V_G$ and the corresponding index graph $I \subseteq V_I \times L \times V_I$, the compression ratio is defined as $\frac{|V_G|}{|V_I|}$. For the query evaluation we implemented the two-step method described earlier. To gain further insights related to some details of query evaluation, we measured not only the total time, but also the time taken to compute index matches (i.e. the time to complete the first of the two necessary steps). The following two queries were used for evaluation.

Q_1:
```
SELECT *
WHERE {
    ?person foaf:name ?name .
    ?person foaf:phone ?phone .
    ?person foaf:homepage ?homepage .
    ?person foaf:mbox ?mbox .
}
```

Q_2:
```
SELECT ?name
WHERE {
    ?person foaf:name ?name .
    ?person foaf:phone ?phone .
    ?person foaf:homepage ?homepage .
    ?person foaf:mbox ?mbox .
}
```

Both Q_1 and Q_2 focus on specifying the same structural pattern that needs to be satisfied, namely: the person in question must have all optional attributes. The difference is in the SELECT lists: Q_1 asks for the values of all variables, while Q_2 defines a projection to a single variable. This way, the second step of query evaluation for Q_2 can be carried out on a graph pattern that is pruned to contain only the single `?name` node.

4. Results

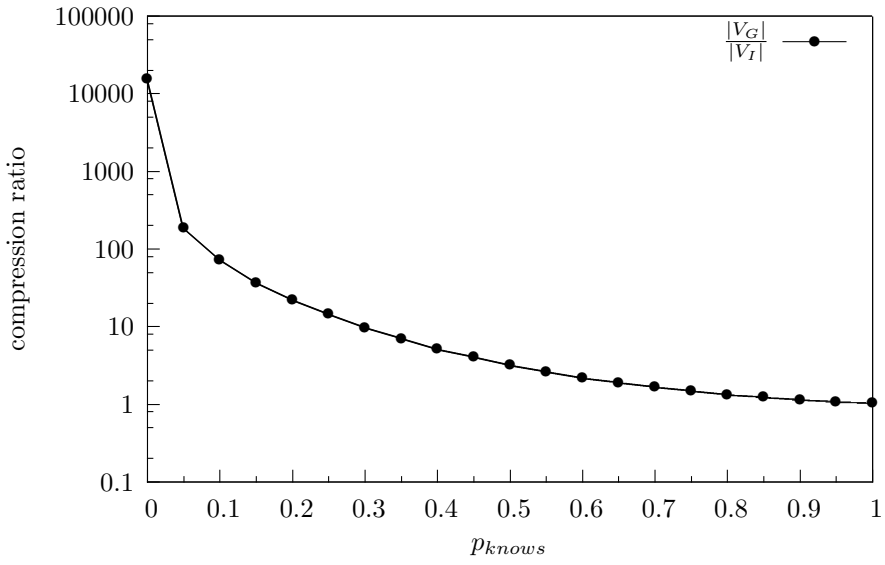

Figure 1. The resulting compression ratio of building a structure index for datasets generated with various p_{knows} parameters.

An important characteristic of a structure index is how the compression ratio changes as the structural complexity increases. To investigate this, we measured the size of the index compared to the size of the graph, for various artificial social network-like datasets generated with gradually increasing p_{knows} values, as described in Section 3. The results can be seen on Figure 1. When the dataset consists of isolated stars ($p_{knows} = 0$), the ratio is the highest (more than 15000). This is not surprising, because in this case the lack of diversity among the possible paths causes the index to contain only 28 nodes regardless of the value of $|V_G|$, thus the ratio can be made arbitrarily large by increasing the number of people in the dataset. For larger p_{knows} values, the ratio rapidly drops (note that the y-axis uses a logarithmic scale): for $p_{knows} = 0.65$ it is already below 2, and it continues to approach the ratio of 1, where the index is practically useless, as computing the results on the compressed graph does not yield a significant advantage.

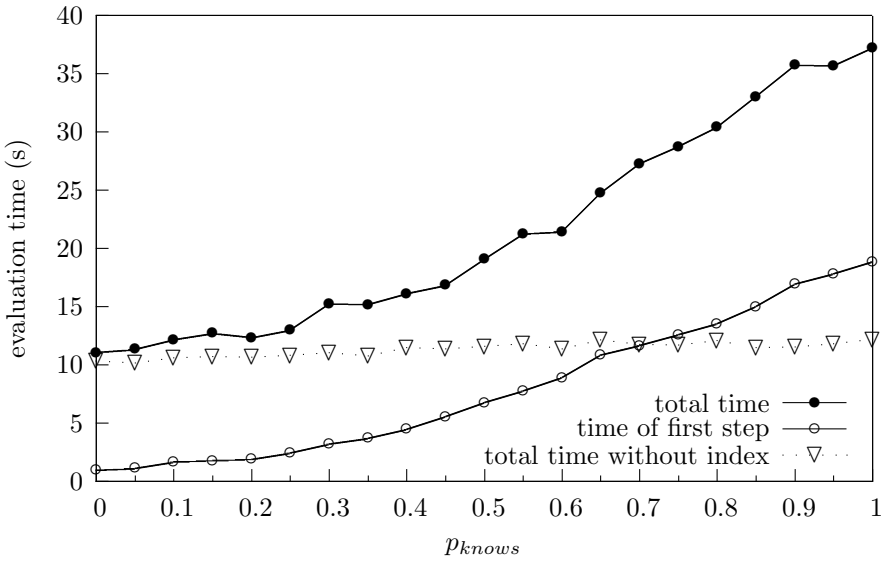

Figure 2. Evaluation time of Q_1 on datasets generated with various p_{knows} parameters.

Figure 2 shows the time needed to evaluate Q_1 on the datasets. Along with the total evaluation time, we included the time spent on computing index matches to indicate how the total time is distributed among the two steps of the evaluation process. As a basis for comparison, the dotted line indicates the time needed to carry out the same task without the use of an index. As

it can be seen, in all cases for this query, the use of an index actually slows down the process, even where the index manages to accomplish an acceptable compression ratio. It is clear from the measurements that (in accordance with the expectations) better compression ratios lead to faster computation of index matches. However, small index sizes (such as the mentioned 28 vertices in the case of $p_{knows} = 0$) mean that a large number of data nodes get assigned to a single index node (in the worst case, more than 15000 on average). As a consequence, the step of computing data matches becomes slower, thereby countering the advantage gained by the outstanding compression ratio. Towards higher p_{knows} values even the first step becomes slower than the naive evaluation time. This can be explained by the fact that while at these values both methods operate on a graph of roughly the same size (see the compression ratios for these values on Figure 1), the index matches are harder to compute, because the matching of a constant requires a membership-test in a set instead of an equality-check.

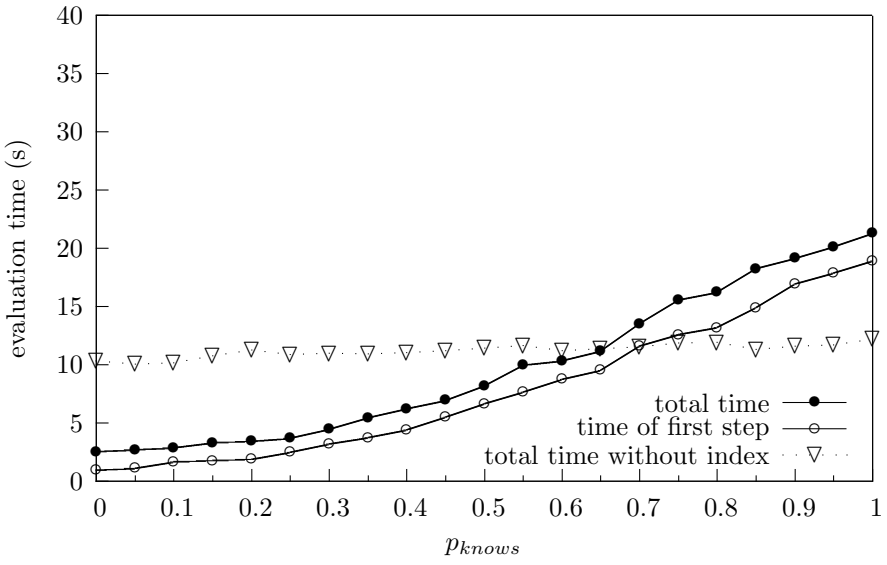

Figure 3. Evaluation time of Q_2 on datasets generated with various p_{knows} parameters.

Figure 3 illustrates how pruning affects the problem of the slow second step. In an extreme case, where all edges of the query can be omitted from the computation of data matches, the evaluation time is dominated by the time of the first step (which is, in turn, governed by the size of the index graph). Thus,

for smaller values of p_{knows}, the utilization of a structure index can beat the naive evaluation time. However, for values around 0.55, the advantage starts to become negligible, and for $p_{knows} \geq 0.7$ the index-using method becomes the slower one.

Summarizing the measurements, we can see that better query evaluation times can only be achieved in strongly limited cases: *a)* we need a prunable query, and additionally *b)* we need a dataset operating with relatively low structural complexity. For our measurements, these meant *a)* an extreme case of only a single node remaining after pruning, and *b)* for a comparable advantage, a value of p_{knows} below 0.55. In our experiences, most real-world applications would have troubles meeting these requirements, especially the latter: even our most extreme test case means only one "friendship"-edge for each entity, whereas real-world datasets usually heavily rely on structural properties similar to this (e.g. there are usually an order of magnitude more "cites"-edges for each paper in a publication database). In principle, the problematic phenomenon could be handled by approaches that ignore certain edge-labels when building the index, such as the Parameterizable Index Graph of Tran & Ladwig [16]. For example, in our case this would mean that we could build the index as if *knows*-edges did not exist, thus accomplishing an outstanding compression ratio. However, for real-world datasets the "guilty" edge-labels may not be as easily identifiable as in the case of our toy dataset. Even if we could manage to find such parameters, there is still the risk of arriving at the opposite extreme, where there are too many data nodes for each index node, as in the case of the Triangles dataset or the Q_1 query for low p_{knows} values.

5. Conclusion and future work

In our paper we investigated the benefits of utilizing structure indexes for RDF graphs, in order to speed up evaluation of SPARQL queries. Our main focus was on the effect of structural complexity. We tested a well-known structure index on a parameterizable artificial dataset describing a hypothetical social network. Our results show that in order to achieve a reasonable advantage using indexes, the dataset and the query must meet some strict requirements, which could make the real-world applications cumbersome. In future research, we would like to further characterize how the structural properties of datasets affect the fruitful applicability of indexes. Our purpose is to design a decision-aiding procedure which could accurately predict whether it would be worth to use a bisimulation-based index for a given dataset and query workload. Our further plans include investigating the problems arising from building and up-

dating indexes for dynamic graphs, where the data and its structure changes over time.

References

[1] **Alzogbi, A. and G. Lausen,** Similar structures inside RDF-graphs, *LDOW2013 - Proc. of WWW2013 Workshop on Linked Data on the Web,* eds. C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas and S. Auer, CEUR Workshop Proceedings **996**, CEUR-WS, 2013.

[2] **Angles, R. and C. Gutierrez,** The expressive power of SPARQL, *ISWC2008 - The Semantic Web,* eds. A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin and K. Thirunarayan, LNCS **5318**, Springer, 2008, 114-129.

[3] **Beckett, D. and T. Berners-Lee, E. Prud'hommeaux and G. Carothers,** RDF 1.1 Turtle - Terse RDF Triple Language, February 2014. `http://www.w3.org/TR/2014/REC-turtle-20140225/.`

[4] **Berners-Lee, T., J. Hendler and O. Lassila,** The Semantic Web, *Scientific American,* **284** (5) (2001), 34-43.

[5] **Cyganak R., D. Wood, M. Lanthaler, G. Klyne, J.J. Carroll and B. McBride,** RDF 1.1. Concepts and abstract syntax, 2014. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.`

[6] **Graves, M., A. Constabaris and D. Brickley,** FOAF: Connecting people on the semantic web, *Cataloging & Classification Quarterly,* **43** (3) (2007), 191-202.

[7] **Harris, S., A. Seaborne and E. Prud'hommeaux,** SPARQL 1.1 Query language, 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.`

[8] **Kiss, A.,** Semi-structured databases, *Algorithms of Informatics,* vol. 2, ch. 20, ed. A. Iványi, mondAt Kiadó, Budapest, 2007, 931-971.

[9] **Kiss, A. and Vu Le Anh,** Combining tree structure indexes with structural indexes in query evaluation on XML data, *Advances in Databases and Information Systems,* eds. J. Eder, Hele-Mai Haav, A. Kalja and J. Penjam, LNCS **3631**, Springer, 2005, 254-267.

[10] **Lehmann, J., R. Isele, M. Jacob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer and C. Bizer,** DBpedia - A large-scale multilingual knowledge base extracted from Wikipedia, *Semantic Web Journal,* 2014. http://svn.aksw.org/papers/2013/SWJ_DBpedia/public.pdf.

[11] **Matono, A., T. Amasaga, M. Yoshikawa and S. Uemura,** An indexing scheme for RDF and RDF schema based on suffix arrays, *Proc. SWDB'03,* 2003, 151-168.

[12] **Pérez, J., M. Arenas and C. Gutierrez,** Semantics and complexity of SPARQL, *ISWC 2006 - The Semantic Web,* eds. I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold and L.M. Aroyo, LNCS **4273**, Springer, 2006, 30-43.

[13] **Schmidt, M., T. Hornung, G. Lausen and C. Pinkel,** SP^2Bench: A SPARQL performance benchmark, CoRR, 2008. http://arxiv.org/abs/0806.4627.

[14] **Schmidt, M., M. Meier and G. Lausen,** Foundations of SPARQL query optimization, *Proc. ICDT'10 - 13th Int. Conf. on Database Theory,* ACM, New York, NY, USA, 2010, 4-33.

[15] **Stuckenschmidt, H., R. Vdovjak, G.-J. Houben and J. Broekstra,** Index structures and algorithms for querying distributed RDF repositories, *Proc. WWW;04 - 13th Int. Conf. on World Wide Web,* ACM, New York, NY, USA, 2004, 631-639.

[16] **Thanh Tran and G. Ladwig,** Structure index for RDF data, *Proc. Semdata'10 - SemDataVLDB,* 2010.

[17] **Weiss, C., P. Carras and A. Bernstein,** Hexastore: Sextuple indexing for semantic web data management, *Proc. of the VLDB Endowment,* **1** (1) (2008), 1008-1019.

B. Pinczel, D. Nagy and A. Kiss
Department of Information Systems
Eötvös Loránd University
Budapest, Hungary
vic@inf.elte.hu
nagydavid@caesar.elte.hu
kiss@inf.elte.hu