Selectivity Estimation for SPARQL Triple Patterns with Shape Expressions

Abdullah Abbas(✉), Pierre Genevès, Cécile Roisin, and Nabil Layaïda

Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
abdullah.abbas.90@gmail.com,
{pierre.geneves,cecile.roisin,nabil.layaida}@inria.fr

Abstract. We optimize the evaluation of conjunctive SPARQL queries, on big RDF graphs, by taking advantage of ShEx schema constraints. Our optimization is based on computing ranks for query triple patterns, which indicates their order of execution. We first define a set of well-formed ShEx schemas, that possess interesting characteristics for SPARQL query optimization. We then define our optimization method by exploiting information extracted from a ShEx schema. The experimentations performed shows the advantages of applying our optimization on the top of an existing state-of-the-art query evaluation system.

1 Introduction

ShEx (Shape Expressions) [12] is a language for expressing constraints on RDF graphs [14]. ShEx schemas can be used to validate RDF documents, generate RDF documents, or communicate expected graph patterns.

In this work we investigate how the evaluation of SPARQL queries [15] on big RDF graphs can be optimized in the presence of ShEx constraints. We propose a method for optimizing the order of evaluation of subqueries, by taking advantage of the information on the data described in ShEx. While SPARQL query optimization by static analysis is important and well-studied, the emergence of constraint languages (such as ShEx) raises new questions on how the knowledge of additional constraints can be effectively leveraged as a part of the static analysis and optimization, especially in the context of big data. In this work, we focus on the logical query structure. We consider SPARQL basic graph patterns (BGPs), and we target the order of execution of triple patterns that aim to minimize the overall execution cost of the query.

Optimization opportunities arise from the presence of joins between query triple patterns. In several situations, the order of execution of triple patterns can be rearranged so that the size of intermediate results for join variables are minimized. Consider the following arbitrary query example with 3 triple patterns and a join on the variable $?x$.

```
SELECT ?x WHERE { ?x :p1000 :a . ?x :p700 :b . ?x :p1 :c . }
```


T. Mikkonen et al. (Eds.): ICWE 2018, LNCS 10845, pp. 195–209, 2018.
https://doi.org/10.1007/978-3-319-91662-0_15

Assume that we know that the triple with predicate : $p1000$ will return 1000 values for $?x$, that of : $p700$ will return 700, and that of : $p1$ will return 1. The join between the first two triple patterns may give up to 700 values which should be reserved in memory for another join with the third triple. A wise choice in this case is to reorder the triple pattern execution, knowing that the third triple is more selective than the other two triples.

Hence, our main purpose in this work is to infer better execution orders for triple patterns, based on the knowledge extracted from a ShEx schema. We implemented our procedure on the top of SPARQLGX, which is one of the most efficient engines for distributed SPARQL evaluation and known to outperform many competitors in the field [6]. SPARQLGX already implements various query optimization techniques including reordering triple patterns [6], but without considering schema constraints. We show that our technique further improves the efficiency of query execution times.

Outline: In Sect. 2 we review the closest related works. In Sect. 3 we introduce some preliminaries necessary for understanding the rest of the paper. In Sect. 4 we define well-formation rules for data-schema pairs that are of interest for our optimization process. In Sect. 5 we define a graph representation of a ShEx schema. In Sect. 6 we formally define our optimization process. In Sect. 7 we report on experimental results with our optimization technique.

2 Related Works

Query optimization for the RDF data model has been studied with various approaches in the literature. Most existing works are based on scanning the data a-priori. The works in [5,9,10] are based on techniques that mainly focus on join optimizations by indexing the data. In [1] the authors provide query optimization a approach based on vertical data partitioning. These works do not consider structured data and data typing.

The works in [7,11] also provide query optimization techniques, by proposing new data representations, and in [8,18] by providing structural summaries or representative schemas. None of these works is based on a given schema, and thus they require an extensive data scan.

For works based on typed data, an approach was proposed in [3] that considers typed XML data trees. Unlike RDF, the tree data-type model of XML allows for extremely efficient subtree pruning. In [13,16] the authors, as in our work, consider query optimization for typed RDF graphs. These works are mainly oriented towards schema violations. In our work, we assume non-violating queries, and we study the effect of reordering which is not studied in the previous works.

In [6] the authors provide a SPARQL query evaluator, SPARQLGX, that relies on a translation of SPARQL queries into executable Spark code that adopts an evaluation strategy based on a storage method and statistics on data. Within the system, optimized joins triple patterns reordering are considered by combining those with common variables, but this reordering does not consider the selectivity of triples based on the structure of the RDF data. Their approach

scales better than the state-of-the-art systems they compare with [6]. For this reason we implemented and tested our optimization technique on the top of this system.

3 Definitions

3.1 SPARQL

SPARQL is an RDF query language and a W3C Recommendation, where RDF is a directed, labeled graph data format for representing information in the web [14] SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions [15].

A SPARQL graph pattern is defined inductively from triple patterns. Given disjoint infinite sets of IRIs - Internationalized Resource Identifiers - (I), blank nodes (B), literals (L), and variables (V), a triple pattern is defined as an instance of $(I \cup B \cup V)(I \cup V)(I \cup B \cup L \cup V)$ denoted by $IBV \times IV \times IBLV$.

In this work we focus on the conjunctive SPARQL fragment, including only BGPs (basic graph patterns), which can be defined abstractly as $q{::}= t \mid q\ AND\ q'$ where t is a triple pattern.

3.2 ShEx

ShEx (or Shape Expressions) is intended to be an RDF constraint language. Logical operators in Shape Expressions such as grouping, conjunction, disjunction and cardinality constraints, are defined to make as closely as possible to their counterparts in regular expressions and grammar languages like BNF [17]. A Shape Expression describes the constraints associated with a subject RDF node as in the following example:

`<Shape1>` { `ex:name xsd:string , ex:phone xsd:string` }

This example shows a definition of a shape in ShEx, where `ex:name` and `ex:phone` are predicates and `xsd:string` is a basic type. This definition means that for an RDF node n to satisfy the requirements of `Shape1`, there must exist exactly two outgoing edges starting from n and labeled with predicates `ex:name` and `ex:phone`. The objects corresponding to these predicates must be of type xsd:string. In general, a ShEx schema allows defining several shapes, and allows using several logical operators, in a way defined formally hereafter.

Abstract Syntax of the Considered ShEx Fragment. Given a finite set of edge labels Σ and a finite set of types Γ, we define a shape expression e over $\Sigma \times \Gamma$ as follows: $e{::}= \epsilon \mid \Sigma \times \Gamma \mid e^{+} \mid (e|e') \mid (e\|e')$ where "|" denotes disjunction, "||" denotes unordered concatenation, and "+" denotes repetition for a positive number of times. $e^{?}$ (optional), e^{*} (unordered Kleene star), $e^{[m;n]}$ (m to n repetitions) can be defined as macros and are also part of the ShEx syntax. In the sequel we write $a{::}t$ as a shorthand for $(a,t) \in \Sigma \times \Gamma$.

A shape expression schema (ShEx), or simply a schema, is a tuple $S = (\Sigma, \Gamma, \delta)$, where Σ is a finite set of edge labels, Γ is a finite set of types, and δ is a type definition function that maps elements of Γ to shape expressions e over $\Sigma \times \Gamma$. If δ is not defined for some type $t \in \Gamma$, the default definition is $\delta(t) = \epsilon$.

We notice that a ShEx shape (or simply a shape) is itself a type. While a shape is considered as a user-defined type, more generally a type may also be a basic built-in type (like xsd:string in the concrete syntaxes of ShEx).

Semantics of ShEx [17]**.** Semantically, an RDF graph is valid against a ShEx schema if it is possible to assign types to the nodes of the graph in a manner that satisfies the type definitions of the schema. We assume a fixed graph $G = (V, E)$ which resembles an RDF graph, and a fixed schema $S = (\Sigma, \Gamma, \delta)$. A typing of G w.r.t. S is a function $\lambda : V \rightarrow 2^{\Gamma}$ that associates with every node of G a set of types. Next, the conditions that a typing needs to satisfy are identified. Given a typing λ and a node $n \in V$ we define the neighborhood-typing of n w.r.t. λ as bag (i.e. multiset) over $\Sigma \times 2^{\Gamma}$ as $neighborTyping^{\lambda}_{G}(n) = \{|a::\lambda(m) \mid (n, a, m) \in E|\}$. We note by $L(e)$ the bag language of a shape expression e, i.e. $L(e)$ is the set of bags allowed by the language of e. Now, λ is a valid typing of S on G if and only if every node satisfies the type definitions of its associated type i.e., for every $n \in V$, $neighborTyping^{\lambda}_{G}(n) \in L(\delta(t))$, for all $t \in \lambda(n)$.

3.3 Our Preliminary Definitions

In accordance with the aforementioned abstract syntax and semantics, we define a shape expression *inclusion relation* which we use in the rest of the paper.

Definition 1. *Given a shape expression e, a predicate p and a ShEx shape s. The inclusion relation $(p, s) \in e$ is defined inductively as follows:*

- $(p, s) \in e$ *if* $e = (p, s)$
- $(p, s) \in e$ *if* $e = e_1^+$ *and* $(p, s) \in e_1$
- $(p, s) \in e$ *if* $e = e_1 \mid e_2$, *and* $(p, s) \in e_1$ *or* $(p, s) \in e_2$
- $(p, s) \in e$ *if* $e = e_1 \mid\mid e_2$, *and* $(p, s) \in e_1$ *or* $(p, s) \in e_2$

We also define the following shape expression *optional* condition.

Definition 2. *Given a shape expression e, a predicate p and a ShEx shape s. The atomic shape expression (p, s) is optional in e, written as $(p, s) \in_{opt} e$ if:*

- $e = e_1 \mid e_2$ *and* $(p, s) \notin e_1$ *and* $(p, s) \in e_2$, *OR*
- $e = e_1 \mid e_2$ *and* $(p, s) \in e_1$ *and* $(p, s) \notin e_2$, *OR*
- $e = e_1 \mid e_2$ *and* $(p, s) \in_{opt} e_1$ *or* $(p, s) \in_{opt} e_2$, *OR*
- $e = e_1 \mid\mid e_2$ *and* $(p, s) \in_{opt} e_1$ *and* $(p, s) \in_{opt} e_2$, *OR*
- $e = e_1^+$, *and* $(p, s) \in_{opt} e_1$

4 Well Formed Data-Schema Pairs

Using RDF as a data format often raises a number of data modelling issues for which choices must be made. The same information might end up being represented in different ways according to the designer choices. Thus, different ShEx schemas – all of which correctly and usefully describe different aspects of the same data graph – might be suggested. Accordingly, we introduce a notion of well-formed data-schema pairs which provide the maximal set of desired information that can be inferred from the ranking procedure described in Sect. 6, without adding new constraints.

The rules for well-formation guarantee that the necessary information needed for our ranking can be deduced from the ShEx schema, yet the ranking procedure is not deterministic. For some shapes, the relations attached to them are not indicative for the selectivity of those shapes. We also define in this section the schema formation rules that makes our shape ranking procedure deterministic.

Definition 3 (Well-Formed Data-Schema Pair). *A data-schema pair* (G, S) *is well-formed if and only if the following rules hold.*

1. ***Cardinality rule:*** *Every m-to-n relation between two schema shapes in* S*, where* $m > n$ *or* m *is not bound, is modelled from the m-sided shape to the n-sided shape.*
2. ***Shape distinction rule:*** *For every 4 schema shapes* $s_1, s_2, so_1, so_2 \in S$ *(not necessarily distinct), and for every 2 predicates* p_a *and* p_b *(not necessarily distinct),* so_1 *and* so_2 *are distinct if the following conditions hold:*
   - $(p_a, so_1) \in \delta(s_1)$
   - $(p_b, so_2) \in \delta(s_2)$
   - O_1 *is the set of nodes of* G *which occur as objects of* p_a *whose subject belongs to the shape* s_1
   - O_2 *is the set of nodes of* G *which occur as objects of* p_b *whose subject belongs to the shape* s_2
   - $O_1 \cap O_2 = \emptyset$

The well-formation rules are particularly interesting because they can be applied on the top of any schema, yet keeping it valid for all the datasets which correspond to it. They are also sufficient and useful for optimizations of real life data-schema examples, although not totally deterministic. A restrictive superset of rules which makes our ranking deterministic is given in the following definition.

Definition 4 (Ranking-Deterministic Data-Schema Pair). *A data-schema pair* (G, S) *is ranking-deterministic if and only if the following rules hold.*

1. ***Well-formedness:*** (G, S) *is well-formed.*
2. ***Cardinality rule:*** *There is no closure cardinality (+,*) in* S*.*
3. ***Shape distinction rule:*** *For every 3 shapes* $s_o, s_1, s_2 \in S$ *(not necessarily distinct), if there exist 2 predicates* p_a *and* p_b *(not necessarily distinct) where* $(p_a, s_o) \in \delta(s_1)$ *and* $(p_b, s_o) \in \delta(s_2)$*, then* s_1 *and* s_2 *refers to the same shape, and* p_a *and* p_b *refers to the same predicate.*

4. ***Data nodes isolation rule:*** *For every data IRI instance d, every 2 shapes $s_1, s_2 \in S$, and every predicate p, if $(p, s_2) \in \delta(s_1)$ and d belongs to the shape s_2, then there exists a data IRI instance d' such that the RDF triple $\langle d', p, d\rangle \in G$. (This states that a data instance shall not be isolated from other data instances unless isolation is required by the given schema.)*

In the following Subsects. 4.1 and 4.2, we give examples and additional descriptions of the well-formedness rules, aiding to understand how they contribute to our ranking procedure.

4.1 Cardinality

Example 1. Assume we want to model a schema describing the relation between students and schools. If we know that the relation in the data between schools and students will be 1-to-many, then the following two schema examples are legitimate, but only the first one is well formed w.r.t. the data.

Schema proposition 1: (Well-Formed)

```
<Student> { :name xsd:string , :school @<School> }
<School> { :name xsd:string }
```

Schema proposition 2: (Not Well-Formed)

```
<Student> { :name xsd:string }
<School> { :name xsd:string, :student @<Student> + }
```

As it is evident from Example 1, the *well-formation cardinality rule* tries to avoid the usage of positive and Kleene closures ($+, *$). Formally, the semantics of the two proposed schemas are different. Schema proposition 1 is more restrictive. Schema proposition 2 misses the restriction that a student should belong to 1 and only 1 school, although it is still an acceptable schema even if this restriction is inherent in the data.

Indeed, the *well-formation cardinality rule* helps us to determine the relative quantity of shape occurrences in the data. For example Schema proposition 1 allows us to know that the $\langle Student\rangle$ instances occur in the data at least as much as $\langle School\rangle$ instances, and normally more.

4.2 Shape Distinction

A shape in a ShEx schema can be as general as allowing any node in any RDF graph to belong to it. The more the shape has restrictions, the more it describes a specific type of nodes. The *well-formation shape distinction rule* puts restrictions on shapes that seem to be too general that they surely miss expressing some constraints that are inherent in the data.

Example 2. Assume we want to model a schema describing the relation between students and researchers to their corresponding schools and research companies. Knowing that schools are not research companies, then the following two schema examples are legitimate, but only the first one is well formed w.r.t. the data.

Schema proposition 1: (Well-Formed)

```
<Student> { :name xsd:string, :school @<School> }
<Researcher>{:name xsd:string, :company @<Company>}
```

Schema proposition 2: (Not Well-Formed)

```
<Student>{:name xsd:string,:school @<Establishment>}
<Researcher>{:name xsd:string,:company @<Establishment>}
```

In Example 2, Schema proposition 2 will not allow us to determine the relative quantity of $\langle Student \rangle$ instances to those of $\langle Establishment \rangle$ instances in the data, while with Schema proposition 1 we are confident that the quantity of $\langle Student \rangle$ instances are more than that of $\langle School \rangle$ instances.

5 Shape Relation Graph

In this section we define a shape graph representation that we use to assign ranks to shapes in Sect. 6. A *shape relation graph* is a graphical representation focusing on the relations existing between shapes in a ShEx document. It is an intermediate structure that will be used later for selectivity estimation analyses.

Definition 5 (Shape Relation Graph). *Given a ShEx document S, we define a shape relation graph $G = \mathcal{SRG}(S)$ as a tuple (N, E) of set of nodes N, each corresponding to a ShEx shape, and an labelled directed relation E between nodes such that:*

- *$E(n_1, x, n_2)$ defines an edge from n_1 to n_2 labeled with x.*
- *Given any two nodes $n_1, n_2 \in N$, and any predicate p, then $E(n_1, p, n_2)$ if and only if $(p, n_2) \in \delta(n_1)$ and $(p, n_2) \notin_{opt} \delta(n_1)$*
- *Given any two nodes $n_1, n_2 \in N$, and any predicate p, then $E(n_1, p^{\epsilon}, n_2)$ if and only if $(p, n_2) \in_{opt} \delta(n_1)$*

Figure 1 shows the *shape relation graph* of a real life schema used in our experimentation (Sect. 7). User-defined types are shown as ovals while built-in types (like xsd:string) and IRIs are shown as rectangles.

Given a *shape relation graph* G, we denote by $\mathcal{R}(G)$ the set of all root nodes (i.e. has no incoming edges) of G, and we denote by $\mathcal{C}(G)$ the set of all cycles in the graph G.

6 Ranking

In order to decide the order of execution of query triple patterns, we assign them ranks inferred from the analysis of the ShEx document. These ranks are based on two main concepts: (1) The hierarchical relations between ShEx shapes, and (2) The predicate distributions among ShEx shapes.

The first concept gives rankings to shapes, and the second concept gives ranking to predicates. The ranking of query triple patterns is based on the product of both rankings together.

6.1 Hierarchical Relations Between ShEx Shapes

In ShEx, the definition of a shape may be based on other shapes defined in the same schema. This notion, called shape inclusion, is explicitly represented by the edges of the *shape relation graph* defined in Sect. 5. Such edge relations between shapes allow us to infer information about the relative frequency of data corresponding to these shapes.

Consider Schema proposition 1 in of Example 1. Representing it as a *shape relation graph*, $\langle School \rangle$ is a child of $\langle Student \rangle$. Each student in the data should have exactly one registered school, and multiple students may be registered in the same school according to the schema. Such a relation between shapes allows us to know that a student instance occurs more in the database than a school instance. Actually the number of schools is at most equivalent to the number of students, where this is a worst case assumption - each student has a unique school. It is evident that this is an extreme case that should not be considered as an average distribution. Thus, it is important to study the hierarchical relations between ShEx shapes. In the example we give the $\langle School \rangle$ shape a priority ranking, since we know that they occur less than the $\langle Student \rangle$ shape, and thus rendering variables corresponding to it more selective.

Concerning cardinality, we notice that a higher cardinality is independent on the actual number of data instances of a shape. For example, if we have `:registeredIn @<School> {1,3}` instead of `:registeredIn @<School>`, that does not necessarily means an increase in the number of schools; the same set of schools may apply in both cases. For the ranking system it is sufficient to consider the relation structure rather than the structure and cardinalities together, and that is why we ignore explicit cardinalities of edges in the *shape relation graph* defined in Sect. 5.

The ranking procedure we propose starts from the root shapes (root nodes). A root shape will have a ranking of 1. Going down through the descendant shapes from the root shape the ranking increases. If there are two (or more) incoming edges to a shape, the lower ranking is transferred. A problem in such a procedure is when there is a cycle between shapes in the graph representation of the schema, that means that the ranking will propagate forever. In such case there is no preference for any of the shapes in the cycles, and all of them must have the same ranking. In some cases, a cycle has an optional relation(s) within it, given by the cardinalities "?", "*", or "{0,n}". In such case, we know that a cut in the cycle can only occur at these points. For asserting the strength of normal relations against such optional relations, the preference for ranking is to actually cut the cycles at these points and apply the ranking system by avoiding such kind of cycles.

Now we formally define all the procedures described, step by step.

Schema Graph Adjustment. First, given the *shape relation graph* G of a ShEx schema, we modify it by detecting optional relations and cycles.

1. For each cycle $C_i(N_i, E_i) \in \mathcal{C}(G)$:
   - For all predicate p, if there exist nodes $n_1, n_2 \in N_i$ such that there exists an edge $E(n_1, p^\epsilon, n_2)$, then remove this edge. Let the new resulting graph be G_{nor}.
2. For each cycle $C_i(N_i, E_i) \in \mathcal{C}(G_{nor})$:
   - Merge all the nodes $x \in N_i$ into a single node c_i.

Schema Shapes Ranking. Now let the output of the *Schema Graph Adjustment* be $G_{adj}(N, E)$. We define the ranking function $\delta_S(x)$ for all $x \in N$ as follows:

1. For each node $r \in \mathcal{R}(G_{adj})$, $\delta_S(r) = 1$
2. For each node $r \in \mathcal{R}(G_{adj})$, apply the procedure $P(r)$ defined next.

Given a node $x \in N$, the procedure $P(x)$ is defined as follows:

- For each $s \in N$, and for each predicate p where there exists an edge $E(x, p, s)$
  1. If $\delta_S(s)$ is not initialized: $\delta_S(s) = \delta_S(x) + 1$
  2. If $\delta_S(x) + 1 < \delta_S(s)$, then $\delta_S(s) = \delta_S(x) + 1$
  3. Apply $P(s)$

Finally, we transmit the cycle rankings to the original nodes.

- For each cycle $C_i(N_i, E_i)$ in G_{nor}:
  - For each node $x \in N_i$, $\delta_S(x) = \delta_S(c_i)$

6.2 Predicate Distributions Among ShEx Shapes

In the previous section we ranked shapes according to their relative frequency of occurrences based on relations between them. Such a ranking is not sufficient for deciding rankings of triple patterns in a query since such ranking is also affected by the uniqueness versus globality of predicates within shapes.

Given a predicate p used in the shapes $s_1, s_2, \ldots, s_n$. The ranking of p within a shape s, denoted as $\delta_P(p, s)$ is defined as follows.

$$\delta_P(p, s) = \frac{\delta_S(s)}{\delta_S(s_1) \times \delta_S(s_2) \times \cdots \times \delta_S(s_n)} \text{ , if } s \in \{s_1, s_2, \ldots, s_n\}$$

$$\delta_P(p, s) = \frac{1}{\delta_S(s_1) \times \delta_S(s_2) \times \cdots \times \delta_S(s_n)} \text{ , otherwise}$$

The previous formula works by reducing the ranking of a predicate when it is more global, i.e. when it is used with more shapes. With more shapes the factors in the denominator will increase and thus reducing the overall ranking. Such predicates are frequent, they are used for many node types, and this means there will be a large set of nodes in the database associated with such predicates. The ranking system tends to leave such predicates to be executed lastly, and that is why the modelled function reduces its ranking. On the other hand, if a predicate

is unique for a certain shape, its ranking tends to be bigger by reducing the number of denominators to only one, which is the shape it corresponds to.

We notice that if a predicate p corresponds to only one shape s_m, then the ranking corresponding to it will be always 1, where this value represents the highest ranking possible.

$$\delta_P(p, s_m) = \frac{\delta_S(s_m)}{\delta_S(s_m)} = 1$$

On the other hand, the lowest possible ranking is when the predicate p is used globally in all the shapes defined in the ShEx document, and particularly when the shape considered for the current ranking is a root node, which have the lowest possible shape ranking of 1, and the denominator is the largest possible which is the product of all the shape rankings.

$$\delta_P(p, s_{root}) = \frac{1}{\prod\limits_{\forall i, \text{ where } s_i \text{ is a shape}} \delta_S(s_i)}$$

6.3 SPARQL Query Triple Rankings

Now our purpose is to rank the triple patterns given a BGP query. Triple patterns with higher ranking will be executed first. Before ranking triples, we need to validate the BGP against the ShEx document, and for each subject in the triple patterns the ShEx validator will decide to which shapes this subject may belong. A subject may belong to multiple shapes at the same time. Thus, for each subject s occurring in the triple patterns we have a set $C(s)$ of candidate shapes for s. For convenience, given a triple pattern t, we define $C(t) = C(s)$ if s is a subject of t. We also define $p(t)$ as the predicate of the triple t.

To define the triple ranking function, we use the two ranking functions δ_S and δ_P defined previously.

Given a BGP B and a triple pattern $t \in B$, we define the ranking of the triple t, denoted by $\delta_T(t)$, as follows:

$$\delta_T(t) = Avg\Big[\delta_S(S_i) \times \delta_P(p(t), S_i)\Big]_{\forall i,\ S_i \in C(t)}$$

For a given triple t, the previous function is the average of the product $\delta_S \times \delta_P$ by considering all the possible candidate shapes for the subject of t.

7 Evaluation

We prepared an experiment in order to show the advantageous effect of our optimization procedure. We generate data according to the Social Network Benchmark (SNB) schema of the Linked Data Benchmark Council (LDBC) [4]. The data and the workload tests are generated by gMark [2]. gMark is a graph and

query workload generator based on an input schema. Our experiment demonstrates 4 different datasets of different sizes, thus we show 4 different charts corresponding to the datasets, and therefore allowing to further comment on the effect of the data size.

7.1 Generated Data

Using the gMark tool, we generated 4 datasets, all according to the LDBC SNB schema, which is represented as a *shape relation graph* in Fig. 1. The datasets sizes are 5M, 30M, 50M, and 100M nodes, corresponding to about 11M, 67M, 113M, and 227M RDF triples respectively.

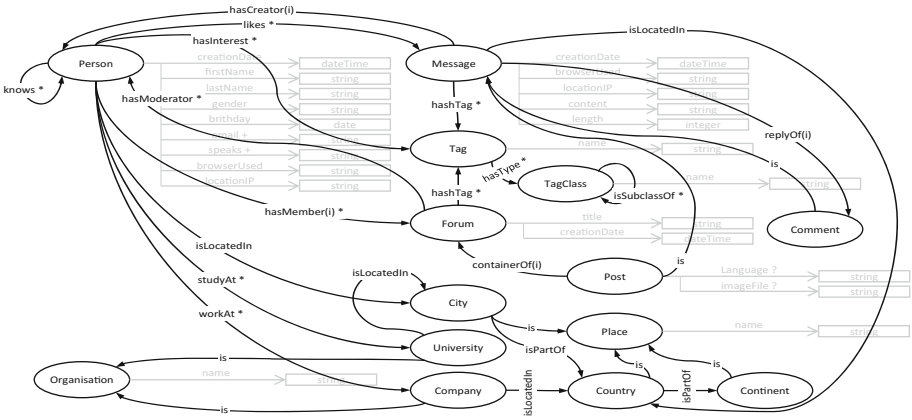

Fig. 1. Shape relation graph (LDBC SNB)

7.2 Workload

Using gMark, we also generated a workload consisting of 12 conjunctive SPARQL queries based on the LDBC SNB schema. We setup the sizes of the queries such that in each query there are between 6 and 10 triple patterns, and there are between 4 and 6 distinguished variables. The choice of the query size is to allow for structures to form within the schema hierarchy, and not to limit it to simple variable relations. Going beyond the size where such hierarchies form is pointless for our evaluation, yet we give a small range to provide a variety of formation choices.

Our purpose is to compare, for each query, the evaluation of the optimized triple patterns order resulting from our method with that of counter part queries, which are just equivalent to the optimized ones but with different order of their triple patterns. In our experiment we generate 50 random permutations for each query of the workload.

7.3 SPARQLGX

An advantage of our optimization technique is that it can be applied on the top of query systems like SPARQLGX [6]. In our experiment we show how the application of our technique further decreases the average run time for SPARQLGX in the presence of a ShEx document.

SPARQLGX, in the current state, has a triple pattern ordering strategy that is based on grouping triple patterns with common join variables together, but the order of the groups themselves is not deterministic; it also depends on their initial ordering given by the input SPARQL query. In our experiments we show that with SPARQLGX alone we obtain improved results, yet using our ranking strategy based on the ShEx information further improves the results.

We define 3 systems that are included in our experiment as follows:

- **S1:** Is SPARQLGX with its ordering strategy turned off (the system itself provides a configuration that stops reordering triple patterns and keeps the original ordering of the query triple patterns).
- **S2:** Is SPARQLGX with its ordering strategy turned on.
- **Optimized:** Is an extension of SPARQLGX. It extends it with the application of our ordering strategy based on the ShEx information.

7.4 Results

The results of our workload query evaluations with the 4 datasets are presented in Fig. 2 which are explained as follows:

- **The blue area** is the runtime range of system **S2** concerning the different permutations of each test query.
- **avg(S2)** marks the average runtime of all the permutations of each test query with system **S2**.
- **avg(S1)** marks the average runtime of all the permutations of each test query with system **S1**.
- Finally, **the green bars** shows the runtime of each input query with our **optimized system** that has a single deterministic triple patterns ordering for each test query.

For each query run we set an evaluation timeout, which is a maximal value result ($t_{timeout}$ = 15, 110, 110, and 130 seconds respectively) near the top of each presented results chart. Some queries timeout, and the query run time is considered equivalent to the timeout value for calculating the average. In our given charts if the average is shown to be at the top of the chart, then this means it is $\geq t_{timeout}$ (timeout value).

To avoid dispensable information in Fig. 2, we do not show the runtime range of the queries with system **S1** (as done with system **S2**) since there are always input permutations that time out for the considered test queries, and thus we only show the average for this system.

The results show a faster execution of some queries (Q1, Q9, Q10, Q11, and Q12), while it preserves or slightly improves the execution time of the other

queries when run using our methodology. Some queries do not show significant improvement due to the structure of the query and its selectivity. For example if a query is asking for a pair of variables signifying the relation between countries and languages, then the result set of such a query is small and constant, since the number of countries and languages is constant; they do not vary even when the dataset size is exponentially increased, and thus such results are expected for some of the generated queries. Actually these kinds of queries are intentionally generated by gMark for benchmarking purposes (check [2]).

Concerning the dataset sizes, it is clear from the charts that our optimization is less evident when the 5M nodes dataset is compared to the bigger datasets. Compared to system **S2** in Fig. 2(a), the optimized orderings of queries Q1, Q10, and Q12 showed a slight improvement. In Fig. 2(b) the improvement is more significant with the latter queries, in addition to the new improvements in queries Q9 and Q11. By further increasing the size of the datasets, the improvement almost stay the same (or precisely it barely increases), which shows a threshold where the gain, although significant, is stabilized.

The average of the improvement of our system (**Optimized**) compared to **avg(S2)** is given as follows:

- **Dataset 5M nodes:** Improvement of queries ranged between 1.2% and 20.5%. The mean improvement of all test queries is 3.8%.
- **Dataset 30M nodes:** Improvement of queries ranged between 1.6% and 87%. The mean improvement of all test queries is 23.5%.
- **Dataset 50M nodes:** Improvement of queries ranged between 1.4% and 84.6%. The mean improvement of all test queries is 25.2%.

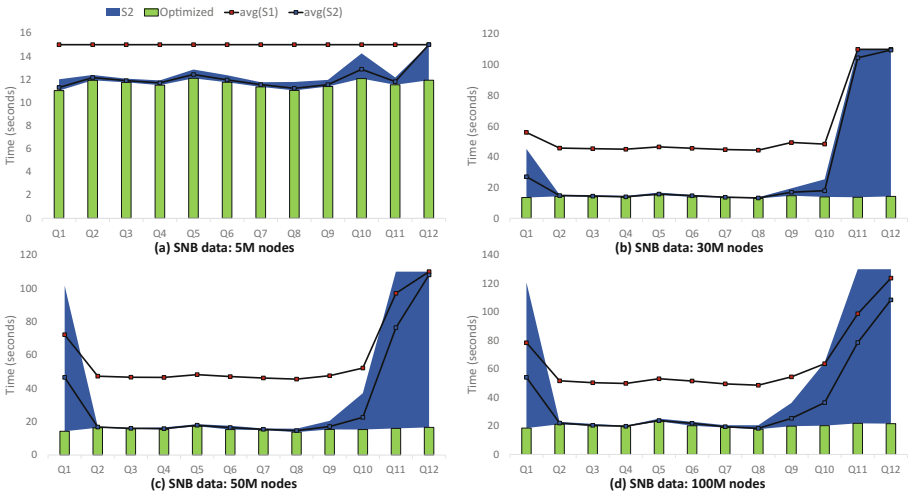

Fig. 2. Comparing ranking-optimized query evaluation with other systems. (Color figure online)

– **Dataset 100M nodes:** Improvement of queries ranged between 1.6% and 85.1%. The mean improvement of all test queries is 25.7%.

8 Conclusion

The proposed query ranking only requires analysis of ShEx schema, an emerging schema language, which is particularly interesting when applied on huge datasets. This is a radically different and promising optimization technique compared to those based on data scans, even with statistical approaches, that are affected by data updates and requires huge data calculations.

We defined a well-formation notion for data-schema pairs that is useful for inferring quantitative information about data instances and we defined a procedure for determining rankings. We implemented a prototype of our system on top of the SPARQLGX query evaluation engine, which is known to outperform many competitors in the field. We compared the rankings found by our system, owing to the analysis of ShEx constraints, to the original reordering method of SPARQLGX in terms of query evaluation times, and with datasets of various sizes. Preliminary experimental results indicate that most rankings found by our system lead to improvements in query execution times. This illustrates the interest of considering ShEx constraints for SPARQL query optimization.

References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable semantic web data management using vertical partitioning. In: Proceedings of the 33rd International Conference on VLDB, VLDB 2007, pp. 411–422. VLDB Endowment (2007)
2. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., Advokaat, N.: gMark: schema-driven generation of graphs and queries. IEEE Trans. Knowl. Data Eng. **29**(4), 856–869 (2017)
3. Benzaken, V., Castagna, G., Colazzo, D., Nguyen, K.: Optimizing XML querying using type-based document projection. ACM Trans. Database Syst. **38**(1), 4:1–4:45 (2013)
4. Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D., Boncz, P.: The LDBC social network benchmark: interactive workload. In: Proceedings of SIGMOD 2015, pp. 619–630. ACM (2015)
5. Goasdoué, F., Kaoudi, Z., Manolescu, I., Quiané-Ruiz, J., Zampetakis, S., et al.: CliqueSquare: efficient Hadoop-based RDF query processing. In: BDA 2013-Journées de Bases de Données Avancées (2013)
6. Graux, D., Jachiet, L., Genevès, P., Layaïda, N.: SPARQLGX: efficient distributed evaluation of SPARQL with Apache spark. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) ISWC 2016, Part II. LNCS, vol. 9982, pp. 80–87. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46547-0_9
7. Joshi, A.K., Hitzler, P., Dong, G.: Logical linked data compression. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 170–184. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38288-8_12

8. Kim, H., Ravindra, P., Anyanwu, K.: Type-based semantic optimization for scalable RDF graph pattern matching. In: Proceedings of WWW 2017, pp. 785–793. International WWW Conference Steering Committee (2017)
9. Lee, K., Liu, L.: Scaling queries over big RDF graphs with semantic hash partitioning. Proc. VLDB Endow. **6**(14), 1894–1905 (2013)
10. Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., Koziris, N.: H2rdf+: high-performance distributed joins over large-scale RDF graphs. In: 2013 IEEE International Conference on Big Data, pp. 255–263, October 2013
11. Pham, M.-D., Passing, L., Erling, O., Boncz, P.: Deriving an emergent relational schema from RDF data. In: Proceedings of WWW 2015, pp. 864–874 (2015)
12. Prud'hommeaux, E., Gayo, J.E.L., Solbrig, H.: Shape expressions: an RDF validation and transformation language. In: Proceedings of SEM 2014, pp. 32–40. ACM (2014)
13. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: Proceedings of the 13th International Conference on Database Theory, ICDT 2010, pp. 4–33. ACM (2010)
14. Schreiber, G., Raimond, Y.: RDF 1.1 primer. W3C note, W3C, June 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/
15. Seaborne, A., Harris, S.: SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
16. Serfiotis, G., Koffina, I., Christophides, V., Tannen, V.: Containment and minimization of RDF/S query patterns. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 607–623. Springer, Heidelberg (2005). https://doi.org/10.1007/11574620_44
17. Staworko, S., Boneva, I., Gayo, J.E.L., Hym, S., Prud'hommeaux, E.G., Solbrig, H.: Complexity and expressiveness of ShEx for RDF. In: ICDT 2015, vol. 31, pp. 195–211 (2015)
18. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.: gStore: answering SPARQL queries via subgraph matching. Proc. VLDB **4**(8), 482–493 (2011)