Advances in Accessing Big Data with Expressive Ontologies

Ralf Möller[1], Christian Neuenstadt[1],
Özgür L. Özçep[1], and Sebastian Wandelt[2]

[1] Hamburg University of Technology, 21073 Hamburg, Germany
[2] Humboldt-Universität zu Berlin, 10099 Berlin, Germany

Abstract. Ontology-based query answering has to be supported w.r.t. secondary memory and very expressive ontologies to meet practical requirements in some applications. Recently, advances for the expressive DL $\mathcal{SHI}$ have been made in the dissertation of S. Wandelt for concept-based instance retrieval on Big Data descriptions stored in secondary memory. In this paper we extend this approach by investigating optimization algorithms for answering grounded conjunctive queries.[3]

1 Introduction

Triplestores, originally designed to store Big Data in RDF format on secondary memory with SPARQL as a query language, are currently more and more used in settings where query answering (QA) w.r.t. ontologies is beneficial. However, reasoning w.r.t. ontologies in secondary memory is provided for weakly expressive languages only (e.g., RDFS), if at all, and in some cases, query answering algorithms are known to be incomplete. For weakly expressive DL languages, such as DL-Lite, good results for sound and complete query answering w.r.t. large (virtual) Aboxes have already been achieved with OBDA based query rewriting techniques and schema specific mapping rules [1]. However, for expressive, more powerful DLs such as $\mathcal{ALC}$ and beyond only first steps have been made. Solving the problem of Accessing Big Data with Expressive Ontologies (ABDEO) is an important research goal.

A strategy to solve the ABDEO problem is to "summarize" Big Aboxes by melting individuals such that Aboxes fit into main memory [2]. In some situations inconsistencies occur, and summarization individuals must be "refined" (or unfolded) at query answering time in order to guarantee soundness and completeness, a rather expensive operation [3]. Other approaches make use of Abox modularization techniques and try to extract independent modules such that query answering is sound and complete. A first investigation of Abox modularization for answering instance queries w.r.t. the DL $\mathcal{SHIF}$ is presented in [5].

[3] This work has been partially supported by the European Commission as part of the FP7 project Optique (`http://www.optique-project.eu/`).

However, modularization with iterative instance checks over all individuals and modules of an Abox is not sufficient to ensure fast performance [5].[4]

The ABDEO approach presented here is based on a modularization approach developed by Wandelt [15,12,14] for really large Aboxes containing data descriptions for > 1000 universities in terms of LUBM scale measures [6], i.e., datasets in the range of billions of triples. Modules (islands) derived by Wandelt's techniques are usually small in practical applications and can be loaded into main memory such that a standard tableau prover can be used for instance checks. Iteration over all individuals gives sound and complete answers, in principle. Compared to [5], Wandelt (i) proposed extended modularization rules, (ii) implemented incremental ways of computing Abox modularizations, and (iii) investigated new ways to optimize sound and complete concept-based query answering (*instance queries*) with tableau-based reasoning systems for the logic $\mathcal{SHI}$. In particular, "similarities" between modules are detected such that a single instance query on a representative data structure (a so-called one-step node) yields multiple results at a time, and thus, instance checks are saved (the approach is reminiscent of but different from [3], see below or cf. [12] for details). Due to modularization rules, one-step node query answering is sound [12] and in many (well-defined) cases complete for eliminating candidates for a successive iterative instance checking process. In addition, to eliminate candidates, Wandelt and colleagues also investigate complete approximation techniques (see [15] for details).

In this paper we extend Wandelt's modularization based approach for query answering by investigating optimization techniques for answering *grounded conjunctive queries* w.r.t. $\mathcal{SHI}$ ontologies. Grounded conjunctive queries are more expressive from a user's point of view than instance queries. We argue that grounded conjunctive queries substantially narrow down the set of instance checking candidates if selective role atoms mentioned in queries are exploited for generating candidates for concept atoms, such that approximation techniques (to, e.g., DL-Lite) are not required in many cases. We demonstrate our findings using the LUBM benchmark as done, e.g., in [9] and [6]. As an additional extension to Wandelt's work, which uses specific storage layouts for storing Abox data descriptions and internal information in SQL databases, we investigate ontology-based access to existing data stores, namely triplestores, while providing query answering w.r.t. expressive ontologies.

2 Preliminaries

We assume the reader is familiar with description logic languages, ontologies (knowledge bases), inference problems, and optimized tableau-based reasoning algorithms (see, e.g., [11,7]). For the reader's convenience we define conjunctive queries in general, and grounded conjunctive queries in particular (adapted from [10]). In the following we use $\mathbf{AtCon}, \mathbf{Con}$, and $\mathbf{Rol}$ for the sets of atomic

[4] Note that Abox modularization is different from Tbox modularization, as for instance investigated in [4].

concept descriptions, concept descriptions, and role descriptions, respectively, in the ontology.

A *conjunctive query* (CQ) is a first-order query q of the form $\exists \boldsymbol{u}.\psi(\boldsymbol{u}, \boldsymbol{v})$ where ψ is a conjunction of concept atoms $A(t)$ and role atoms $R(t, t')$, with A and R being concept and role names, respectively. The parameters t, t' are variables from $\boldsymbol{u}$ or $\boldsymbol{v}$ or constants (individual names). The variables in $\boldsymbol{u}$ are the existentially quantified variables of q and $\boldsymbol{v}$ are the free variables, also called distinguished variables or answer variables of q. The query q is called a k-ary query iff $|\boldsymbol{v}| = \mathrm{k}$. In a *grounded conjunctive query* (GCQ), $\boldsymbol{u}$ is empty. We only consider grounded conjunctive queries in this paper. We define an operator *skel* that can be applied to a CQ to compute a new CQ in which all concept atoms are dropped.

The query answering problem is defined w.r.t. an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$. Let $Inds(\mathcal{A})$ denote the individuals in A. For $\mathcal{I}$ an interpretation, $q = \psi(\boldsymbol{v})$ a k-ary grounded conjunctive query, and $a_1, \ldots, a_k \in Inds(\mathcal{A})$, we write $\mathcal{I} \models q[a_1, \ldots, a_k]$ if $\mathcal{I}$ satisfies q (i.e., all atoms of q) with variables v_i replaced by $a_i, 1 \leq i \leq k$. A *certain answer* for a k-ary conjunctive query q and a ontology $\mathcal{O}$ is a tuple $(a_1, \ldots, a_k)$ such that $\mathcal{I} \models q[a_1, \ldots, a_k]$ for each model $\mathcal{I}$ of $\mathcal{O}$. We use $cert(q, \mathcal{O})$ to denote the set of all certain answers for q and $\mathcal{O}$. This defines the query answering problem. Given a $\mathcal{SHI}$ ontology $\mathcal{O}$ and a GCQ q, compute $cert(q, \mathcal{O})$. It should be noted that "tree-shaped" conjunctive queries can be transformed into grounded conjunctive queries, possibly with additional axioms in the Tbox [8]. The restriction to grounded conjunctive queries is not too severe in many practical applications.

Grounded conjunctive query answering can be implemented in a naive way by computing the certain answers for each atom and doing a join afterwards. Certain answers for concept atoms can be computed by iterating over $Inds(\mathcal{A})$ with separate instance checks for each individual. Rather than performing an instance check on the whole Abox, which is too large to fit in main memory in many application scenarios, the goal is to do an instance check on a module such that results are sound and complete. More formally, given an input individual a, the proposal is to compute a set of Abox assertions $\mathcal{A}_{isl}$ (a subset of the source Abox $\mathcal{A}$), such that for all atomic (!) concept descriptions A, we have $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models A(a)$ iff $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{isl} \rangle \models A(a)$.

3 Speeding Up Instance Retrieval

In order to define subsets of an Abox relevant for reasoning over an individual a, we define an operation which splits up role assertions in such a way that we can apply graph component-based modularization techniques over the outcome of the split.

Definition 1 (Abox Split). *Given*

- *a role description R,*
- *two distinct named individuals a and b,*

– *two distinct fresh individuals* c *and* d*, and,*
– *an Abox* $\mathcal{A}$*,*

an Abox split *is a function* $\downarrow_{c,d}^{R(a,b)}$: $\mathbf{SA} \rightarrow \mathbf{SA}$*, defined as follows (*$\mathbf{SA}$ *is the set of Aboxes and* $\mathcal{A} \in \mathbf{SA}$*):*

– *If* $R(a,b) \in \mathcal{A}$ *and* $\{c,d\} \nsubseteq Ind(\mathcal{A})$*, then*

$$\begin{aligned}\downarrow_{c,d}^{R(a,b)}(\mathcal{A}) = &\mathcal{A} \setminus \{R(a,b)\} \cup \{R(a,d), R(c,b)\} \cup \\ &\{C(c) \mid C(a) \in \mathcal{A}\} \cup \\ &\{C(d) \mid C(b) \in \mathcal{A}\}\end{aligned}$$

– *Else*

$$\downarrow_{c,d}^{R(a,b)}(\mathcal{A}) = \mathcal{A}.$$

In the following we assume that the Tbox is transformed into a normal form such that all axioms are "internalized" (i.e., on the lefthand side of a GCI there is only $\top$ mentioned. For a formal definition of the normal form of a Tbox, see [12]. Here we use an example to illustrate the idea.

Example 1 (Example for an Extended $\forall$*-info Structure).* Let

$$\begin{aligned}\mathcal{T}_{Ex1} = \{&Chair \sqsubseteq \forall headOf.Department,\\ &\exists memberOf.\top \sqsubseteq Person,\\ &GraduateStudent \sqsubseteq Student\},\\ \mathcal{R}_{Ex1} = \{&headOf \sqsubseteq memberOf\},\end{aligned}$$

then the TBox in normal form is

$$\begin{aligned}\mathcal{T}_{Ex1norm} = \{&\top \sqsubseteq \neg Chair \sqcup \forall headOf.Department,\\ &\top \sqsubseteq \forall memberOf.\bot \sqcup Person,\\ &\top \sqsubseteq \neg GraduateStudent \sqcup Student\}\end{aligned}$$

and the extended $\forall$-info structure for $\mathcal{T}_{Ex1norm}$ and $\mathcal{R}_{Ex1}$ is:

$$extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R) = \begin{cases}\{Department, \bot\} & \text{if } R = headOf,\\ \{\bot\} & \text{if } R = memberOf,\\ \emptyset & \text{otherwise.}\end{cases}$$

Definition 2 (Extended $\forall$-info Structure). *Given a TBox* $\mathcal{T}$ *in normal form and an Rbox* $\mathcal{R}$*, an* extended $\forall$-info structure for $\mathcal{T}$ and $\mathcal{R}$ *is a function* $extinfo^{\forall}_{\mathcal{T},\mathcal{R}}$: $\mathbf{Rol} \rightarrow \wp(\mathbf{Con})$*, such that we have* $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$ *if and only if there exists a role* $R_2 \in \mathbf{Rol}$*, such that* $\mathcal{R} \models R \sqsubseteq R_2$ *and* $\forall R_2.C \in clos(\mathcal{T})$*, where* $clos(\mathcal{T})$ *denotes the set of all concept descriptions mentioned in* $\mathcal{T}$*.*

Now we are ready to define a data structure that allows us to check which concept descriptions are (worst-case) "propagated" over role assertions in $\mathcal{SHI}$-ontologies. If nothing is "propagated" that is not already stated in corresponding Abox assertions, a role assertion is called splittable. This is formalized in the following definition.

Definition 3 ($\mathcal{SHI}$-splittability of Role Assertions). *Given a $\mathcal{SHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a role assertion $R(a, b)$, we say that $R(a, b)$ is* $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$ *iff*

1. *there exists no transitive role R_2 with respect to $\mathcal{R}$, such that $\mathcal{R} \vDash R \sqsubseteq R_2$,*
2. *for each $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$*
   - *$C = \bot$ or*
   - *there is a $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \vDash C_2 \sqsubseteq C$ or*
   - *there is a $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \vDash C \sqcap C_2 \sqsubseteq \bot$*

   and
3. *for each $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R^-)$*
   - *$C = \bot$ or*
   - *there is a $C_2(a) \in \mathcal{A}$ and $\mathcal{T} \vDash C_2 \sqsubseteq C$ or*
   - *there is a $C_2(a) \in \mathcal{A}$ and $\mathcal{T} \vDash C \sqcap C_2 \sqsubseteq \bot$.*

So far, we have introduced approaches to modularization of the assertional part of an ontology. In the following, we use these modularization techniques to define structures for efficient reasoning over ontologies.

We formally define a subset of assertions, called an *individual island*, which is worst-case necessary, i.e. possibly contains more assertions than really necessary for sound and complete instance checking. Informally speaking, we take the graph view of an Abox and, starting from a given individual, follow all role assertions in the graph until we reach a $\mathcal{SHI}$-splittable role assertion. We show that this strategy is sufficient for entailment of atomic concepts. The formal foundations for these subsets of assertions have been set up before, where we show that, under some conditions, role assertions can be broken up while preserving soundness and completeness of instance checking algorithms. First, in Definition 4, we formally define an individual island candidate with an arbitrary subset of the original Abox. The concrete computation of the subset is then further defined below.

Definition 4 (Individual Island Candidate).
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a named individual $a \in Ind(\mathcal{A})$, an individual island candidate*, is a tuple $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$, such that $\mathcal{A}^{isl} \subseteq \mathcal{A}$. Given an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$ and an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ is a* model of ISL_a*, denoted $\mathcal{I} \vDash ISL_a$, if $\mathcal{I} \vDash \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \rangle$. Given an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$, we say that ISL_a* entails a concept assertion $C(a)$*, denoted $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle \vDash C(a)$, if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \vDash ISL_a \implies \mathcal{I} \vDash C(a)$. We say that ISL_a* entails a role assertion $R(a_1, a_2)$*, denoted $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle \vDash R(a_1, a_2)$, if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \vDash ISL_a \implies \mathcal{I} \vDash R(a_1, a_2)$.*

Please note that entailment of concept and role assertions can be directly reformulated as a decision problem over ontologies, i.e., we have $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a\rangle \models C(a) \iff \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}\rangle \models C(a)$. In order to evaluate the quality of an individual island candidate, we define soundness and completeness criteria for individual island candidates.

Definition 5 (Soundness and Completeness for Island Candidates). *Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$ and an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a\rangle$, we say that ISL_a is* sound for instance checking in ontology *$\mathcal{O}$ if for all atomic concept descriptions $C \in$ **AtCon**, $ISL_a \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models C(a)$. ISL_a is* complete for instance checking in ontology *$\mathcal{O}$ if for all atomic concept descriptions $C \in$ **AtCon**, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models C(a) \implies ISL_a \models C(a)$.*

We say that ISL_a is sound for relation checking in ontology *$\mathcal{O}$ if for all role descriptions $R \in$ **Rol** and all individuals $a_2 \in Inds(\mathcal{A})$*

- $ISL_a \models R(a, a_2) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models R(a, a_2)$ *and*
- $ISL_a \models R(a_2, a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models R(a_2, a)$.

ISL_a is complete for relation checking in ontology *$\mathcal{O}$ if for all role descriptions $R \in$ **Rol** and all individuals $a_2 \in Inds(\mathcal{A})$*

- $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models R(a, a_2) \implies ISL_a \models R(a, a_2)$ *and*
- $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models R(a_2, a) \implies ISL_a \models R(a_2, a)$.

We say that ISL_a is sound for reasoning in ontology *$\mathcal{O}$ if ISL_a is sound for instance and relation checking in $\mathcal{O}$. We say that ISL_a is* complete for reasoning in *$\mathcal{O}$ if ISL_a is complete for instance and relation checking in $\mathcal{O}$.*

Definition 6 (Individual Island).
Given an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a\rangle$ for an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$, ISL_a is called individual island for *$\mathcal{O}$ if ISL_a is sound and complete for reasoning in $\mathcal{O}$.*

An individual island candidate becomes an individual island if it can be used for sound and complete reasoning. It is easy to see that each individual island candidate is sound for reasoning since it contains a subset of the original Abox assertions.

In Fig. 1, we define an algorithm which computes an individual island starting from a given named individual a. The set **agenda** manages the individuals which have to be visited. The set **seen** collects already visited individuals. Individuals are visited if they are connected by a chain of $\mathcal{SHI}$-unsplittable role assertions to a. We add the role assertions of all visited individuals and all concept assertions for visited individuals and their direct neighbors.

Theorem 1 shows that the computed set of assertions is indeed sufficient for complete reasoning.

Theorem 1 (Island Computation yields Individual Islands for Ontologies). *Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$ and an individual $a \in Inds(\mathcal{A})$, the algorithm in Fig. 1 computes an individual island $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a\rangle$ for a.*

Input: Ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, individual $a \in Inds(\mathcal{A})$
Output: Individual island $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$
Algorithm:
Let agenda $= \{a\}$
Let seen $= \emptyset$
Let $\mathcal{A}^{isl} = \emptyset$
While agenda $\neq \emptyset$ **do**
Remove a_1 from **agenda**
Add a_1 to **seen**
Let $\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{C(a_1) \mid C(a_1) \in \mathcal{A}\}$
For each $R(a_1, a_2) \in \mathcal{A}$
$\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{R(a_1, a_2) \in \mathcal{A}\}$
If $R(a_1, a_2) \in \mathcal{A}$ is $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$ **then**
$\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{C(a_2) \mid C(a_2) \in \mathcal{A}\}$
else agenda = **agenda** $\cup$ $(\{a_2\} \setminus$ **seen**$)$
For each $R(a_2, a_1) \in \mathcal{A}$
$\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{R(a_2, a_1) \in \mathcal{A}\}$
If $R(a_2, a_1) \in \mathcal{A}$ is $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$ **then**
$\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{C(a_2) \mid C(a_2) \in \mathcal{A}\}$
else agenda = **agenda** $\cup$ $(\{a_2\} \setminus$ **seen**$)$

Fig. 1. Schematic algorithm for computing an individual island.

The proof is given in [12].

For each individual there is an associated individual island, and Abox consistency can be checked by considering each island in turn (islands can be loaded into main memory on the fly). Individual islands can be used for sound and complete instance checks, and iterating over all individuals gives a sound and complete (albeit still inefficient) instance retrieval procedure for very large Aboxes.

Definition 7 (Pseudo Node Successor). *Given an Abox $\mathcal{A}$, a* pseudo node successor *of an individual $a \in Inds(\mathcal{A})$ is a pair $pns^{a,\mathcal{A}} = \langle \mathbf{rs}, \mathbf{cs} \rangle$, such that there is an $a_2 \in Ind(\mathcal{A})$ with*

1. $\forall R \in \mathbf{rs}.(R(a, a_2) \in \mathcal{A} \vee R^-(a_2, a) \in \mathcal{A})$,
2. $\forall C \in \mathbf{cs}.C(a_2) \in \mathcal{A}$, *and*
3. **rs** *and* **cs** *are maximal.*

Definition 8 (One-Step Node).
Given $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual $a \in Inds(\mathcal{A})$, the one-step node of a for $\mathcal{A}$, *denoted $osn^{a,\mathcal{A}}$, is a tuple $osn^{a,\mathcal{A}} = \langle \mathbf{rootconset}, \mathbf{reflset}, \mathbf{pnsset} \rangle$, such that* **rootconset** $= \{C | C(a) \in \mathcal{A}\}$, **reflset** $= \{R | R(a, a) \in \mathcal{A} \vee R^-(a, a) \in \mathcal{A}\}$, *and* **pnsset** *is the set of all pseudo node successors of individual a.*

It should be obvious that for realistic datasets, multiple individuals in an Abox will be mapped to a single one-step node data structure. We associate the corresponding individuals with their one-step node. In addition, it is clear that

one-step nodes can be mapped back to Aboxes. The obvious mapping function is called *Abox*. If $Abox(osn^{a,\mathcal{A}}) \models C(a)$ for a query concept C (a named concept), all associated individuals of $osn^{a,\mathcal{A}}$ are instances of C. It is also clear that not every *one-step node* is complete for determining whether a is not an instance of C. This is the case only if one-step nodes "coincide" with the islands derived for the associated individuals (splittable one-step nodes). Wandelt found that for LUBM in many cases islands are very small, and one-step nodes are indeed complete in the sense that if $Abox(osn^{a,\mathcal{A}}) \not\models C(a)$ then $\mathcal{A} \not\models C(a)$ (for details see [12]). In the following we assume that for instance retrieval, it is possible to specify a subset of Abox individuals as a set of possible candidates. If the set of candidates is small, with some candidates possibly eliminated by one-step nodes, then iterative instance checks give us a feasible instance retrieval algorithm in practice.

4 Answering Grounded Conjunctive Queries

In this section we will shortly describe an implementation of the introduced techniques with a triplestore database. As other groups we use the *Lehigh University Benchmark* or LUBM [6] for evaluating algorithms and data structures. This benchmark is an ontology system designed to test large ontologies with respect to OWL applications. With the LUBM generator, the user can generate n universities each consisting of a random number of departments and individuals. As the number of individuals and the number of assertions increases nearly linear with the number of universities, LUBM is an instrument to test the performance for query answering machines, especially for grounded conjunctive queries in a scalable Abox environment. If a system cannot handle LUBM with, say, a billion triples, it cannot deal with more complex scenarios occurring in future applications, either. The code we used in this paper for evaluating the optimization techiques is written in Java and can be downloaded at `http://www.sts.tu-harburg.de/people/c.neuenstadt/`. We store data in the triplestore AllegroGraph, which provides access to role instances (triples) w.r.t. RDFS plus transitive roles, i.e., role hierarchies and transitive roles are handled by AllegroGraph. Alternatively one could use materialization or query expansion in the OBDA style for role hierarchies. SPARQL is used as a query language for specifying queries to be executed on a particular triplestore database.

4.1 Setting up an AllegroGraph Triplestore

AllegroGraph is run as a server program. In our setting, data is loaded directly into the server, whereas islands as well as one-step nodes are computed by a remote program run on a client computer (we cannot extend the server program easily). In a first step the whole data system has to be set up before we can start query answering. During the setup process, the communication between client and server system consists basically of sending SPARQL queries for data access required in the algorithm shown in Fig. 1 as well as sending AllegroGraph

statements for adding additional triples (or changing existing ones) for storing islands and one-step nodes. Islands are indicated using the subgraph components of AllegroGraph triples (actually, quintuples).

The "similarity" of one-step nodes is deifned using hashvalues with a sufficient bitlength. We first compute a set from each island, compute a hashvalue for it, and store it together with the specific island in the triplestore. Identical hash values allow one to refer to "similar" one-step nodes (with additional checks applied to the collision list as usual for hashing).

Given the concept description C from the query and the named individual a from the tuple, we load the specific one-step node for a from the database and determine whether osn_a entails $C(a)$. Depending on the outcome, three states are possible:

- Osn_a entails $C(a)$, then a is actually an instance of C.
- Osn_a entails $\neg C(a)$ or does not entail $C(a)$ and is splittable, then a is actually not an instance of C.
- Osn_a is not splittable, then the client has to load and check the entire island associated with a to find out whether a actually is an instance of C.

Candidates for concept atoms are determined in our experiments by first doing a retrieval for a query q by executing $skel(q)$ (see above for a definition). Bindings for variables in $skel(q)$ define the candidates for retrieval with concept atoms. By eliminating all skeleton query result tuples that include individuals which do not belong to corresponding concept assertions used in the query, finally all remaining tuples are correct answers to the original conjunctive query.

Wandelt has already investigated the efficiency of Abox modularization techniques for an SQL database server. Here, instead, we work directly on an existing AllegroGraph triplesotore, convert the large ontology step by step into small chunks and compare the generated modules with the local modularization of Sebastian Wandelt on the SQL server [12]. The processing time for one university is about 5000 seconds on AllegroGraph server, where it is like one minute for Wandelt's approach. The modularization of one university takes nearly 200,000 queries. The decrease in performance is based on the huge number of SPARQL mini queries between server and the remote modularization client in the prototype implementation. Thus, only 5 universities are investigated for query answering experiments.

4.2 Evaluating Conjunctive Queries

For evaluating grounded conjunctive queries, LUBM provides 14 predefined test queries, which check several database criteria. We run the tests with an ontology, which uses the description logic language $\mathcal{SHI}$. LUBM queries differ in the amount of input, selectivity and reasoning behaviour for example by relying on inverse roles, role hierarchy, or transitivity inferences.[5] Selectivity basically

[5] In the LUBM dataset, explict assertions about subrelations of degreeFrom are made (e.g., doctoralDegreeFrom). The relation degreeFrom is declared as an inverse to

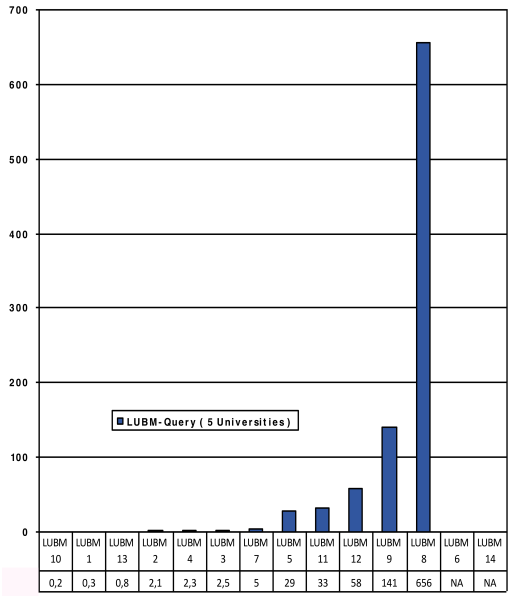

Fig. 2. Runtimes for all queries (in seconds).

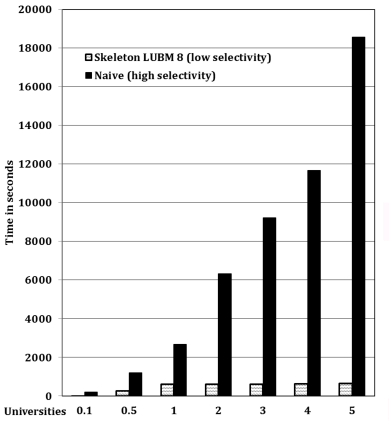

Fig. 3. Comparison between skeleton query and naive approach (numbers for runtimes in seconds).

means that the grounded conjunctive queries being considered have role atoms with large result sets to be reduced by atom queries, which we call less selective (proportion of the instances involved in the skeleton are high compared to those that actually satisfy the query criteria), or automatically excludes a lot of individuals, what we call a highly selective query. Rather than doing a join on the result sets of all atoms in a grounded conjunctive query, role atoms define candidates for concept atoms. Thus for selective queries, candidate sets for concept atoms are smaller. This reduces the number of instance checks that remain if, e.g., one-step node optimizations are not applicable (see above).

The result indicates that the less selective a query is w.r.t. role atoms, the more instance checks we need afterwards, and the more time consuming retrieval is (see Figure 2). Nevertheless, most of the LUBM queries are handled fast, even with the simple implementation for concept atoms with repetitive instance checks. Performance for Query 8 will be increased with an implementation of full one-step node retrieval (with multiple individuals returned at a time, see above). Queries 6 and 14 contain only concept atoms and are not tested here.

To demonstrate that our skeleton query candidate generator is able to significantly improve the results for queries with even low selectivity, we compare the

hasAlumnus. Thus, although, e.g., Query 13 contains a reference to University0 (asking for fillers of hasAlumnus), adding new universities with degreeFrom tuples with University0 on the righthand side causes the cardinality of the set of fillers for hasAlumnus w.r.t. University0 to increase, i.e., having a constant in the query does not mean the result set to be independent of the number of universities.

approach of skeleton queries with the naive approach without skeleton queries in Figure 3. One can directly see the huge performance gain of the skeleton query even for less selective queries. We avoid lots of instance checks and can therefore decrease the answering time by orders of magnitude in many cases.

5 Conclusion

In this work we extended the Abox modularization strategies of Wandelt and colleagues to the efficient use of grounded conjunctive queries on triplestore servers. Results obtained with the techniques discussed in this paper are sound and complete. Note that query engines investigated in [6] are incomplete.

Our prototype needs linear time to add information to the triplestore in a setup phase. Therefore we were not able to run queries on billions of triples. We conclude that island computation needs to be built into the triplestore software iteself and cannot be done from a remote client.

In the average case, the size of the individual island (with respect to the number of assertion in its Abox) is considerably smaller than the original Abox. In our experiments the size is usually orders of magnitudes smaller. Please note that these modularization techniques allow traditional description logic reasoning systems to deal with ontologies which they cannot handle without modularizations (because the data or the computed model abstraction does not fit into main memory).

In addition, the evaluation of the prototype showed how grounded conjuctive queries on triplestore servers w.r.t. expressive ontologies ($\mathcal{SHI}$) can be implemented using only a small size of main memory. The main strategy is to use a skeleton query and try to keep the necessary amount of instance checks in the second step as small as possible. If the number of results for less selective skeleton queries gets larger, the number of instance checks increases rapidly. In some cases it would obviously have been better to reduce the set of possible tuples by considering concept atoms first. This observation has also been made in [9] and, much earlier, in [16] where more elaborate query plan generation techniques are investigated, albeit for main memory systems.

We would like to emphasize that the proposed optimizations can be used for parallel reasoning over ontologies [13]. This will be further investigated in future work such that ABDEO will become possible for practically relevant datasets and ontologies that are more demanding than LUBM.

References

1. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and databases: The DL-Lite approach. *Reasoning Web. Semantic Technologies for Information Systems*, pages 255–356, 2009.
2. J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, and L. Ma. Scalable semantic retrieval through summarization and refinement. In *Proceedings of the National Conference on Artificial Intelligence*, page 299. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

3. Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Edith Schonberg, and Kavitha Srinivas. Scalable highly expressive reasoner (SHER). *J. Web Sem.*, 7(4):357–361, 2009.
4. B.C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. *proc. KR*, 2006, 2006.
5. Y. Guo and J. Heflin. A scalable approach for partitioning owl knowledge bases. In *Proc. of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006)*, 2006.
6. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.
7. V. Haarslev and R. Möller. On the scalability of description logic instance retrieval. *Journal of Automated Reasoning*, 41(2):99–142, 2008.
8. Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic ABoxes. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.
9. Ilianna Kollia, Birte Glimm, and Ian Horrocks. SPARQL query answering over OWL ontologies. In *Proc. of the 8th European Semantic Web Conf. (ESWC 2011)*, Lecture Notes in Computer Science, pages 382–396. Springer, 2011.
10. Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic EL using a relational database system. In Craig Boutilier, editor, *IJCAI*, pages 2070–2075, 2009.
11. R. Möller and V. Haarslev. Tableaux-based reasoning. In S. Staab and R. Studer, editors, *Handbook of Ontologies*, pages 509–528. Springer, 2009.
12. S. Wandelt. *Efficient instance retrieval over semi-expressive ontologies*. PhD thesis, Hamburg University of Technology, 2011.
13. Sebastian Wandelt and Ralf Möller. Distributed island-based query answering for expressive ontologies. In Paolo Bellavista, Ruay-Shiung Chang, Han-Chieh Chao, Shin-Feng Lin, and Peter M. A. Sloot, editors, *Advances in Grid and Pervasive Computing, 5th International Conference, GPC 2010, Hualien, Taiwan, May 10-13, 2010. Proceedings*, volume 6104 of *Lecture Notes in Computer Science*, pages 461–470. Springer, 2010.
14. Sebastian Wandelt and Ralf Möller. Towards Abox modularization of semi-expressive description logics. *Journal of Applied Ontology*, 7(2):133–167, 2012.
15. Sebastian Wandelt, Ralf Möller, and Michael Wessel. Towards scalable instance retrieval over ontologies. *Journal of Software and Informatics*, 2010.
16. Michael Wessel. *Flexible und konfigurierbare Software-Architekturen fr dateninten-sive ontologiebasierte Informationssysteme*. PhD thesis, Technische Universität Hamburg-Harburg, Hamburg, Germany, 2009. ISBN 978-3-8325-2312-1.