Web Semantics: Science, Services and Agents on the World Wide Web 55 (2019) 122–144

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Ontology paper

Reasoning and querying web-scale open data based on DL-Lite$_{\mathcal{A}}$ in a divide-and-conquer way

Zhenzhen Gu [a,*], Songmao Zhang [b], Cungen Cao [a]

[a] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[b] Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences, Beijing, China

HIGHLIGHTS

- We propose to use DL-LiteA techniques to reason and query the web-scale Open Data.
- We provide a divide-and-conquer reasoning and query answering approach for DL-LiteA.
- We build the provided approach from both theoretical and practical perspectives.
- We conduct experiments on large-scale open datasets to verify the provided approach.

ARTICLE INFO

Article history:
Received 3 April 2018
Received in revised form 2 November 2018
Accepted 29 January 2019
Available online 6 February 2019

Keywords:
DL-lite$_{\mathcal{A}}$
Open data
Semantic web
Knowledge base
Query answering
Divide-and-conquer

ABSTRACT

We propose to use DL-Lite$_{\mathcal{A}}$ techniques to reason and query the Web-scale Open Data (knowledge bases) described by Semantic Web standards like RDF and OWL due to the low reasoning complexity and suitable expressivity of the language. When facing the real-life scalability challenge, the actual reasoning and query answering may become infeasible by the following two factors. Firstly, for both satisfiability checking and conjunctive query answering, a polynomial size of queries may need to be answered over the data layers of the corresponding knowledge bases (KBs) w.r.t. the size of the schema knowledge of these KBs. Secondly, for KBs with massive individual assertions, evaluating a single query over the data layers may be highly time-consuming. This impels us to seek for a divide-and-conquer reasoning and query answering approach for DL-Lite$_{\mathcal{A}}$, with the basic idea of partitioning both KBs and queries into smaller chunks and decomposing the original reasoning and query answering tasks into a group of independent sub-tasks such that the overall performance can be improved by taking advantage of parallelization and distribution techniques. The challenge for designing such an approach lies in how to carry out partitioning and reasoning reduction in a sound and complete way. Motivated by hash partitioning of RDF graphs, we expect the smaller KB chunks to have the local feature for both satisfiability checking and simple-query answering. Here simple-queries are the conjunctive queries whose query atoms share a common variable or individual. For query answering, we expect to partition a query into smaller simple-queries and evaluate them over smaller KB chunks. Under these expectations, our divide-and-conquer approach is constructed from both theoretical and practical perspectives. Theoretically, definitions of KB partitions and query partitions are presented, and the sufficient and necessary conditions are identified to determine whether a KB partition holds the desired features. Practically, based on the theoretical results, the concrete ways of partitioning KBs and queries as well as evaluating query partitions over KB partitions are described. Moreover, a strategy of optimizing the procedure of evaluating query partitions over KB partitions is provided to improve the overall query answering performance. To verify our approach, two Web-scale open datasets, DBpedia and BTC 2012 dataset, have been chosen. The empirical results indicate that the provided approach opens new possibilities for realizing performance-critical applications on the Web with both high expressivity and scalability.

1. Introduction

Coined by Tim Berners-Lee, the Semantic Web [1] intends to create a web of data whose content is readable and understandable especially to machines. For this ambition, the World Wide Web

* Corresponding author.
E-mail addresses: guzhenzhen0720@163.com (Z. Gu), smzhang@math.ac.cn (S. Zhang), cgcao@ict.ac.cn (C. Cao).

https://doi.org/10.1016/j.websem.2019.01.003
1570-8268/

(W3C) community directed by Tim Berners-Lee has proposed a series of standards and techniques, including the Recourse Description Framework (RDF) [2], the Web Ontology Language (OWL) [3] and the Linked Data Principles [4,5], so as to describe, share and integrate data and knowledge on the web in a structured and semantically related way. Over the last decade, the Semantic Web technologies have been greatly developed and successfully applied to many domains, such as life sciences and healthcare. During the same period of time, the Open Data, defined as the data that can be freely used, accessed and shared for any purpose, has drawn more and more attention from both research community and industry [6,7]. One of the best practices of the Open Data is the Linked Open Data (LOD) project [8,9], a blend of Linked Data and Open Data. A large number of open datasets have been described and published on the web based on the LOD principles, including the well known DBpedia [10] and Freebase [11]. As the evolving of the information and AI technologies, one can foresee a continual expansion of large-scale and complex open data specified by the Semantic Web standards.

Although the sheer size of the open data in LOD can be viewed as a positive sign for Semantic Web initiatives, it causes performance bottlenecks for systems and tools that provide data managing and query answering services over the LOD datasets. Moreover, as the growth of schema knowledge described by OWL, realizing efficient reasoning and expressive query answering has become a challenging task for both data publishers and end users. Most of the studies in this respect (for example, [12–15]) focused on dealing with the scalability issue whereas inferring knowledge through reasoning is often ignored. Moreover, when reasoning is considered, most of the time only lightweight RDFS reasoning can be supported.

Description Logics (DLs) [16–19] consists of a group of knowledge representation languages in first-order logic that feature the reasoning decidability, forming the logical foundation of Semantic Web, e.g., the DL $\mathcal{SROIQ}$ [20] underlies the expressive ontology language OWL 2 DL [21]. In addition to Semantic Web, DLs have also been studied in application domains like Data Integration [22] and Biomedical Informatics [23]. In the DL community, one of the main research focuses is concerned with the trade-off between expressive power and computational complexity which are two birds that cannot be killed by one stone. Driven by the motivation of processing large-scale individual data, the DL-Lite family [24,25] is specifically tailored to capture basic ontology language constructs while keeping low reasoning complexity.

In the DL-Lite family, DL-Lite$_{\mathcal{A}}$ [26,27], especially designed for Ontology Based Data Access [28–31], is the most expressive language that characterizes FOL-rewritability. This means that in DL-Lite$_{\mathcal{A}}$, both satisfiability checking and conjunctive query answering can be eventually reduced to answering queries over the data layer of the corresponding knowledge base (KB). Concretely, for a DL-Lite$_{\mathcal{A}}$ KB, by schema knowledge reasoning, a set of queries can be constructed such that the KB is satisfiable iff each generated query has an empty answer set over the data layer of this KB. Moreover, for a conjunctive query, again by schema knowledge reasoning, this query can be ultimately rewritten into a set of conjunctive queries such that all the certain answers of this query can be soundly and completely captured by evaluating each rewritten query over the data layer of this KB. Thus DL-Lite$_{\mathcal{A}}$ has low complexity for both satisfiability checking and conjunctive query answering, i.e., AC_0 data complexity and PTime combined complexity [32]. Furthermore, the separation between schema knowledge and data layer reasoning makes it possible to use highly optimized database management systems and engines in practice for query answering. These distinguished features impel us to propose using DL-Lite$_{\mathcal{A}}$ techniques to reason and query the web-scale, open datasets that are described by Semantic Web standards and exceed the expressivity of RDF and RDFS.

To demonstrate the rationality of our proposal, we have conducted a statistical analysis of the Billion Triple Challenge (BTC) 2012 dataset[1] consisting of 1.1 billion RDF triples (without duplicates) obtained by crawling the open Web during May and June 2012. In this dataset, 99% of the triples are individual assertions, calling for techniques with high scalability at data layers such as DL-Lite$_{\mathcal{A}}$. Moreover, the schema knowledge of this dataset contains a total of 344,778 axioms, of which 339,519 (98%) can be captured by DL-Lite$_{\mathcal{A}}$. Furthermore, among the schema knowledge captured by DL-Lite$_{\mathcal{A}}$, there are 15,932 class inclusion axioms in the form of $A \sqsubseteq \exists R.B$, 4625 class disjoint axioms, 1854 inverse property assertions, 614 functional property assertions, and 167 symmetric property assertions, all beyond the expressivity of RDFS.

Unfortunately, even with low reasoning complexity, when facing the real-life scalability challenge, the actual reasoning and query answering realized by DL-Lite$_{\mathcal{A}}$ techniques may become infeasible by the following two factors. Firstly, for both satisfiability checking and conjunctive query answering, polynomial size of queries may need to be answered over the data layer of the corresponding KB, w.r.t. the size of the schema knowledge of this KB. Secondly, for the KBs with massive individual assertions, even aided with highly optimized database management systems, evaluating a single query over the data layers may be very time-consuming. The combination of these two factors makes the situation even worse, calling for new techniques for DL-Lite$_{\mathcal{A}}$ to tackle the scalability and efficiency challenge.

In computer science, divide-and-conquer is a classical algorithm design paradigm with the idea of breaking down a problem into independent, smaller subproblems, which actually is the basis of efficient algorithms for all kinds of problems.[2] This inspires us to design a divide-and-conquer reasoning and query answering approach for DL-Lite$_{\mathcal{A}}$ to meet the scalability and efficiency requirements of real-life applications. The basic idea is to partition both KBs and queries into smaller chunks and decompose the original reasoning and query answering task into a group of independent subtasks. The performance improvement that such a divide-and-conquer approach can achieve embodies in the following three aspects. Firstly, a smaller size of schema knowledge and queries makes a smaller number of queries needed to be answered over the data layer of the corresponding KB. Secondly, a smaller size of the data layer of KB makes the eventual queries be evaluated more efficiently. And finally, the independence among the subtasks enables distribution and parallelization techniques to take effect so as to improve the overall performance.

There are two main challenges for designing such a divide-and-conquer reasoning and query answering approach for DL-Lite$_{\mathcal{A}}$, namely how to partition KBs and queries, and how to carry out reasoning and query answering reduction in a sound and complete way. These two problems are highly interconnected, as in order to ensure soundness and completeness, different KB and query partitioning strategies require different ways of reasoning reduction, and vice versa. In order to tackle the challenges, some desired principles of partitioning or reasoning reduction must be determined before designing the concrete divide-and-conquer approach.

Partitioning via hashing is a well known strategy for RDF graph splitting, and has been applied to real-life large-scale RDF systems [33–35]. The purpose of hash partitioning is to split a RDF graph into subgraphs such that star-queries (queries with only subject–subject joins) can be evaluated over a RDF graph by answering the query over each subgraph independently and then combining the certain answers obtained. Motivated by hash partitioning of RDF graphs, we require the smaller KB chunks to have the

[1] https://km.aifb.kit.edu/projects/btc-2012/.

[2] https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm.

local feature for both satisfiability checking and simple-query answering (abbreviated as SCSQA in this paper). Here, simple-queries extended from star-queries refer to the conjunctive queries whose query atoms share a common variable or individual. The motivation of requiring the local feature of simple-query answering is that we hope "simple queries" can be answered directly without going through query partition. For "non-simple" queries, we intend to partition them into smaller simple-queries and replace answering them over DL-Lite$_\mathcal{A}$ KBs with evaluating the partitioned smaller simple-queries over the KB partitions with the local feature of SCSQA. When a conjunctive query can be evaluated in such a way soundly and completely, we say it is simple-query reducible.

Based on these principles, we devise our divide-and-conquer reasoning and query answering approach from both theoretical and practical points of view, and the contributions are summarized as follows.

1. We formalize the definition of DL-Lite$_\mathcal{A}$ KB partitions, and identify the sufficient and necessary conditions that enable a KB partition to have the local feature of SCSQA. Based on these theoretical results, we present a concrete way of computing such KB partitions.
2. We formalize the definition of conjunctive query partitions and evaluation, and identify the sufficient and necessary conditions that determine whether a conjunctive query is simple-query reducible. Based on these theoretical results, we present concrete ways of partitioning and evaluating simple-query reducible queries as well as non-simple query reducible queries.
3. We design a strategy of optimizing the procedure of evaluating query partitions over KB partitions, with the basic idea of reducing the intermediate results produced by answering sub-queries as many as possible so as to speed up the overall query answering.
4. We conduct experiments on two large datasets, DBpedia and BTC 2012 dataset, to demonstrate the rationality, efficiency and scalability of the provided divide-and-conquer approach. The results show that our reasoning and query answering approach is appealing in consuming Web-scale, real-world open data.

To the best of our knowledge, this is the first study on partitioning DL-Lite$_\mathcal{A}$ KBs for the purpose of reasoning and querying large-scale datasets. The rest of the paper is organized as follows. Section 2 presents the syntax and semantics of DL-Lite$_\mathcal{A}$ and conjunctive queries. We present the problem discussed as well as an overview of our approach in Section 3. DL-Lite$_\mathcal{A}$ KB partitioning and conjunctive query partitioning are studied in Sections 4 and 5, respectively, followed by an optimization strategy for query partition evaluation in Section 6. Afterwards, Section 7 presents the experiments of applying our divide-and-conquer approach to real-world datasets. Related work is introduced in Section 8, and finally Section 9 concludes the paper by giving directions for our further work.

2. Preliminaries

For self-containment, in this section, we briefly present the syntax and semantics of DL-Lite$_\mathcal{A}$ and conjunctive queries [].

Let **I**, **C** and **R** be countably infinite and pairwise disjoint sets of names for DL-Lite$_\mathcal{A}$ individuals, classes and roles. The syntax of DL-Lite$_\mathcal{A}$ is illustrated in the following two definitions.

Definition 1. In DL-Lite$_\mathcal{A}$, basic roles S, general roles R, basic classes B and general classes C are inductively defined as follows:

$$S ::= P \mid P^-, \qquad B ::= A \mid \exists S$$
$$R ::= S \mid \neg S, \qquad C ::= B \mid \exists S.B \mid \neg C$$

Table 1
The interpretation of DL-Lite$_\mathcal{A}$ classes and roles as well as axioms and assertions w.r.t. an interpretation $\mathcal{I}$.

$(P^-)^\mathcal{I}$	$= \{(y,x) \mid (x,y) \in P^\mathcal{I}\}$
$(\neg S)^\mathcal{I}$	$= \Delta^\mathcal{I} \times \Delta^\mathcal{I} - S^\mathcal{I}$
$(\exists S)^\mathcal{I}$	$= \{x \mid \exists y.(x,y) \in S^\mathcal{I}\}$
$(\exists S.B)^\mathcal{I}$	$= \{x \mid \exists y.(x,y) \in S^\mathcal{I} \wedge y \in B^\mathcal{I}\}$
$(\neg B)^\mathcal{I}$	$= \Delta^\mathcal{I} - B^\mathcal{I}$
$\mathcal{I} \models B \sqsubseteq C$	iff $B^\mathcal{I} \subseteq C^\mathcal{I}$
$\mathcal{I} \models S \sqsubseteq R$	iff $S^\mathcal{I} \subseteq R^\mathcal{I}$
$\mathcal{I} \models \mathsf{Fun}(S)$	iff $(x,y) \in S^\mathcal{I}$ and $(x,z) \in S^\mathcal{I}$ implies $y = z$
$\mathcal{I} \models A(a)$	iff $a \in A^\mathcal{I}$
$\mathcal{I} \models P(a,b)$	iff $(a,b) \in P^\mathcal{I}$

where $A \in \mathbf{C}$ and $P \in \mathbf{R}$. A DL-lite$_\mathcal{A}$ axiom takes one of the following forms:

$$S \sqsubseteq R, \quad B \sqsubseteq C, \quad \mathsf{Fun}(S)$$

A DL-Lite$_\mathcal{A}$ individual assertion takes one of the forms:

$$A(a), \quad P(a,b)$$

where $A \in \mathbf{C}$, $P \in \mathbf{R}$ and $a, b \in \mathbf{I}$ are called individuals.

Definition 2. A DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is a tuple where $\mathcal{T}$ is a finite set of DL-Lite$_\mathcal{A}$ axioms and $\mathcal{A}$ a finite set of DL-Lite$_\mathcal{A}$ assertions. Moreover, for each axiom $\mathsf{Fun}(S)$ in $\mathcal{T}$, S and S^- do not occur in the right hands of inclusion axioms ($\sqsubseteq$) in $\mathcal{T}$. $\mathcal{T}$ and $\mathcal{A}$ are respectively called the TBox and ABox of $\mathcal{K}$.

Example 1. By DL-Lite$_\mathcal{A}$, the knowledge "*Every person has a mother which is a woman*" and the fact "*Lucy is a person*" can be formally represented by the following axiom and assertion:

$$Person \sqsubseteq \exists\, hasMother.Woman$$
$$Person(Lucy)$$

where *Person* and *Woman* are classes, *hasMother* is a role and *Lucy* an individual.

Let **V** be a set of variables that is disjoint with **I**, **C** and **R**. The syntax of conjunctive queries is formalized in the definition below.

Definition 3. A query atom has the form $A(x)$ or $P(x,y)$ where $A \in \mathbf{C}$, $P \in \mathbf{R}$ and $x, y \in \mathbf{V} \cup \mathbf{I}$. A conjunctive query Q is an expression of the form:

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow q(\vec{x})$$

where $\alpha_1, \ldots, \alpha_n$ are query atoms, $\vec{x}$ is a tuple of elements in $\mathbf{V} \cup \mathbf{I}$ and each variable in $\vec{x}$ occurs in some α_i where $1 \leq i \leq n$. We define the body and head of query Q as $\mathsf{bd}(Q) = \cup_{i=1}^{n}\{\alpha_i\}$ and $\mathsf{hd}(Q) = \vec{x}$, respectively.

Here, we also mention the notion of *conjunctive queries with inequalities* [26], which are simply conjunctive queries where atoms in the form of $x \neq y$ may occur. For a conjunctive query Q, variables occurring in $\mathsf{hd}(Q)$ are called the *distinguished variables* of Q, whereas variables solely occurring in $\mathsf{bd}(Q)$ are called the *non-distinguished variables* of Q.

Example 2. The question asking for persons and their mothers can be formally represented as the following conjunctive query:

$$Person(?x) \wedge hasMother(?x, ?y) \rightarrow q(?x, ?y)$$

Next, we present the semantics of DL-Lite$_\mathcal{A}$ KBs and conjunctive queries.

Semantics of DL-Lite$_\mathcal{A}$ KBs. An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is a tuple where $\Delta^\mathcal{I}$ is a non-empty set and $\cdot^\mathcal{I}$ is a mapping that maps each element in **I**, **C** and **R** to a distinct element in $\Delta^\mathcal{I}$, a subset of $\Delta^\mathcal{I}$ and a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$, respectively. The interpretation of class and role constructors as well as axioms and assertions are illustrated in Table 1. We say that $\mathcal{I}$ is a model of DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ if $\mathcal{I}$ satisfies all the axioms and assertions in $\mathcal{K}$. The satisfiability and entailment ($\models$) are defined as usual.

For a tuple $\vec{u}$, we use $|\vec{u}|$ and $\vec{u}[i]$ to denote the length and the ith element of $\vec{u}$, respectively. For a tuple $\vec{u}'$ consisting of variables and satisfying $|\vec{u}'| = |\vec{u}|$, we use $[\vec{u}'/\vec{u}]$ to denote a substitution. For a tuple or query O, we use $O[\vec{u}'/\vec{u}]$ to denote the result of replacing every occurrence of $\vec{u}'[i]$ in O with $\vec{u}[i]$ for $1 \leq i \leq |\vec{u}|$. For convenience, for a query Q and tuple $\vec{u}$ such that $|\vec{u}| = |\mathsf{hd}(Q)|$, we use $Q(\vec{u})$ to denote $Q[\mathsf{hd}(Q)/\vec{u}]$. The semantics of conjunctive queries is illustrated as follows.

Semantics of conjunctive queries. For a query Q and interpretation $\mathcal{I}$, a binding π of Q over $\mathcal{I}$ is a function that maps each individual a in Q to $a^\mathcal{I}$ and each variable in Q to an element in $\Delta^\mathcal{I}$. We write $\mathcal{I}, \pi \models Q$ if $\pi(x) \in A^\mathcal{I}$ for each $A(x) \in \mathsf{bd}(Q)$, $(\pi(x), \pi(y)) \in P^\mathcal{I}$ for each $P(x, y) \in \mathsf{bd}(Q)$ and $\pi(x) \neq \pi(y)$ for each inequality atom $x \neq y$ occurring in Q. We write $\mathcal{I} \models Q$ if there exists a binding π of Q over $\mathcal{I}$ such that $\mathcal{I}, \pi \models Q$. For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$, a tuple $\vec{u}$ with length $|\mathsf{hd}(Q)|$ and consisting of names occurring in $\mathcal{K}$ is called a certain answer of Q over $\mathcal{K}$ if for each model $\mathcal{I}$ of $\mathcal{K}, \mathcal{I} \models Q(\vec{u})$ holds. We use $\mathsf{ans}(Q, \mathcal{K})$ to denote the set of all the certain answers of Q over $\mathcal{K}$.

In this paper, we only consider satisfiability checking and conjunctive query answering, as they are two basic and core reasoning problems.

3. Problem description and approach overview

Problem description. In DL-Lite$_\mathcal{A}$, both satisfiability checking and conjunctive query answering can eventually be reduced to answering conjunctive queries over the ABoxes of DL-Lite$_\mathcal{A}$ KBs [26,27]. Concretely, for a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, by TBox reasoning, a set $\mathbb{Q}$ of conjunctive queries with or without inequalities can be constructed such that $\mathcal{K}$ is satisfiable iff evaluating each query in $\mathbb{Q}$ over $\mathcal{A}$ returns an empty set, i.e., $\cup_{Q \in \mathbb{Q}} \mathsf{ans}(Q, (\emptyset, \mathcal{A})) = \emptyset$. Moreover, if $\mathcal{K}$ is satisfiable then for each conjunctive query Q, again by TBox reasoning, Q can be rewritten into a set of conjunctive queries $\mathbb{Q}$ such that all the certain answers of Q over $\mathcal{K}$ can be captured by evaluating each query in $\mathbb{Q}$ over $\mathcal{A}$. To realize such query rewriting for DL-Lite$_\mathcal{A}$, one of the classical algorithms is `PerfectRef` [26,27], i.e.,

$$\mathsf{ans}(Q, \mathcal{K}) = \cup_{Q' \in \mathtt{PerfectRef}(Q, \mathcal{T})} \mathsf{ans}(Q', (\emptyset, \mathcal{A}))$$

holds. Thus DL-Lite$_\mathcal{A}$ has low complexity for both satisfiability checking and conjunctive query answering, and the separation between TBox and ABox reasoning enables the exploration of highly optimized database management systems to store ABox assertions and evaluate queries over them.

Even with these efficiency features, when processing real-life, Web-scale open datasets described by Semantic Web standards, the following two factors may make the actual DL-Lite$_\mathcal{A}$ reasoning and query answering tasks infeasible. (1) For both satisfiability checking and conjunctive query answering, a polynomial size of queries may need to be answered over the ABox of the corresponding KB w.r.t. the size of the TBox. Suppose a KB uses the DBpedia ontology (with 233,227 axioms) as its TBox. For satisfiability checking, according to the algorithm `Consistent` [27], 2963,132 queries need to be answered over the ABox of this KB. Furthermore, suppose a conjunctive query contains the classes *dbo:Person* and *dbo:Place*[3] from the DBpedia ontology. By TBox reasoning, this query will be rewritten into more than $1144 \times 1391 = 1591{,}304$ queries, as the two classes respectively have 1144 and 1391 implied sub-classes. (2) For KBs with a Web-scale ABox, even though taking advantage of highly optimized database management systems, evaluating a single query can be time-consuming. For example, in our experimental setting as will be described in Section 7, asking for the persons labeled with "Alan Turing":

$$dbo : Person(?x) \wedge rdfs{:}label(?x, Alan\ Turing@en) \rightarrow q(?x)$$

over the 275,710,447 individual assertions in DBpedia took a total of 52 minutes. Thus efficient techniques are needed for DL-Lite$_\mathcal{A}$ to tackle the scalability and efficiency challenges in performance-critical, Web-scale applications.

The divide-and-conquer approach. In this paper, we propose to cater to the requirement of scalability and efficiency from a divide-and-conquer perspective. Our basic idea is to partition both KBs and queries into smaller chunks, and decompose the original reasoning tasks into a group of independent sub-tasks so that the queries eventually needed to be evaluated can be reduced and quickly answered and distribution and parallelization techniques can take effect to improve the overall performance. The difficulty in realizing such an approach lies in that the partitioning and reasoning reduction shall be carried out in a sound and complete way. Motivated by hash partitioning of RDF graphs, we expect the smaller KB chunks to have the local feature for both satisfiability checking and simple-query answering (SCSQA). Here, simple queries, defined as follows, are extended from star-queries to denote the conjunctive queries whose query atoms sharing a common individual or variable.

Definition 4. We say that conjunctive query Q is a *simple-query* if there exists a variable or individual x occurring in each query atom of Q.

Both star-queries and simple-queries enforce the existence of a variable or individual shared by all their query atoms, whereas for star-queries this shared entity occurs solely at the subject position, which is not required for simple-queries. This means that all star-queries are simple-queries, but not vice versa.

By Definition 4, we can get that the following three queries are all simple-queries, whereas the first is also a star-query and the second and third are not

$$Person(?x) \wedge birthDate(?x, ?y) \wedge member(MIT, ?x) \rightarrow q(?x, ?y)$$
$$subject(?x, ?y) \wedge boarder(?y, ?z) \rightarrow q(?x, ?y, ?z)$$
$$director(?x, ?y) \wedge writer(?z, ?y) \wedge Film(?y) \rightarrow q(?x, ?y, ?z)$$

However, the following conjunctive query asking for persons and the prizes they won as well as the comments of the corresponding prizes is not a simple-query, since there does not exist a variable or individual that is shared by all of its query atoms:

$$Person(?x) \wedge prize(?x, ?y) \wedge comment(?y, ?z) \rightarrow q(?x, ?y, ?z)$$

The motivation of requiring the smaller KB chunks to have the local feature of simple-query answering is to hope that "simple" queries can be evaluated directly without going through query partitioning. Note that, a *simple-query with inequalities* is a simple-query in which inequality atoms $x \neq y$ may occur. On the other hand, for conjunctive query answering, we expect to partition a conjunctive query into smaller simple-queries and evaluate these simple-queries over the KB chunks with the local feature of SCSQA. If a conjunctive query can be answered in such a way soundly and completely, we say it is simple-query reducible.

[3] *dbo* denotes the abbreviation of the namespace of the DBpedia ontology.

Based on these expectations on KB and query partitions, our divide-and-conquer reasoning and query answering approach is built by gradually discussing the following three problems P1 — P3 from both theoretical and practical perspectives:

- P1: DL-Lite$_\mathcal{A}$ KB partitioning;
- P2: Conjunctive query partitioning and evaluation of query partitions over KB partitions;
- P3: Optimization of the procedure of evaluating query partitions over KB partitions.

The first problem P1 is handled by discussing the following three subproblems:

- P11: The definition of DL-Lite$_\mathcal{A}$ KB partitions;
- P12: The sufficient and necessary conditions that enable a KB partition to have the local feature of SCSQA;
- P13: The concrete way of computing the desired DL-Lite$_\mathcal{A}$ KB partitions.

Note that there can be varied ways to define DL-Lite$_\mathcal{A}$ KB partitions, which will make the conditions of detecting as well as the concrete ways of computing the KB partitions with the desired feature totally different. Thus the subproblem P11 shall be studied firstly. Before giving the concrete ways of computing KB partitions, we shall identify the conditions that enable a KB partition to have the local feature of SCSQA. Thus P12 should be discussed secondly. Lastly, from the practical point of view, based on the theoretical results, concrete ways of computing the desired KB partitions must be given, i.e., the subproblem P13.

Analogously to P1, the second problem P2 is also decomposed into the following three subproblems P21–P23:

- P21: The definition of conjunctive query partitions;
- P22: The sufficient and necessary conditions that determine whether a conjunctive query is simple-query reducible;
- P23: The concrete ways of partitioning and evaluating simple-query reducible conjunctive queries and non-simple query reducible conjunctive queries.

The subproblem P21 must be handled firstly. The reason is similar to P11. In order to evaluate as many conjunctive queries in the desired way as possible, we should decide whether all conjunctive queries are simple-query reducible, and if not, the sufficient and necessary conditions of deciding whether a conjunctive query is simple-query reducible must be found out. Thus P22 should be discussed secondly. Lastly, from the practical point of view, based on the theoretical results, the concrete ways of partitioning and evaluating simple-query reducible conjunctive queries and non-simple query reducible conjunctive queries must be provided, i.e., the subproblem P23.

Finally, problem P3 is discussed from the overall efficiency perspective. In the scenario of both KB and query partitioning, generating certain answers of the original query needs to merge the certain answers of all the sub-queries. This goes to compute the Cartesian product of the certain answer sets of sub-queries, which can become extremely time-consuming when these certain answer sets are of large size. Thus our optimization strategy endeavors to reduce the number of intermediate results of sub-query evaluation.

To summary, our work in the paper does not contradict with the fact that DL-Lite$_\mathcal{A}$ features low complexity. Rather, we take advantage of DL-Lite$_\mathcal{A}$ and devise a divide-and-conquer approach so that Web-scale, real-world datasets can be queried by taking reasoning into account in an efficient, sound and complete way.

4. DL-Lite$_\mathcal{A}$ KB partitioning

In this section, we discuss the first problem P1, i.e., the DL-Lite$_\mathcal{A}$ KB partitioning, of our divide-and-conquer approach. As described in Section 3, we subsequently present (P11) the definition of KB partitions, (P12) the conditions of detecting KB partitions with the desired feature, and (P13) the concrete way of computing the desired KB partitions.

4.1. Formalization of DL-Lite$_\mathcal{A}$ KB partitioning

No matter what the KB partition is, each smaller KB chunk will contain a smaller size of ABox. For a smaller sub-ABox, obviously, not all the axioms in the original KB are relevant in terms of the considered reasoning tasks. For example, consider the following KB:

$$(\{A \sqsubseteq B, P \sqsubseteq S\},\ \{A(a), P(b, c)\})$$

For the smaller chunk just containing the assertion $A(a)$, considering the axiom $A \sqsubseteq B$ is sufficient.

Undoubtedly, for a fixed ABox, the smaller the size of the TBoxes, the smaller the number of queries needed to be answered over the ABox for both satisfiability checking and conjunctive query answering, w.r.t. a given TBox and a given query rewriting algorithm. Based on this basic observation, before giving the definition of DL-Lite$_\mathcal{A}$ KB partition, we first provide a way of computing a smaller TBox for a smaller ABox in the definition below.

Definition 5. For a DL-Lite$_\mathcal{A}$ KB $(\mathcal{T}, \mathcal{A})$ and $\mathcal{A}' \subseteq \mathcal{A}$, we use $\mathcal{T}|_{\mathcal{A}'}$ to denote a subset of $\mathcal{T}$ constructed by the following steps. Let $\mathcal{T}|_{\mathcal{A}'} = \emptyset$.

1. For each axiom $\alpha \in \mathcal{T}$, add α to $\mathcal{T}|_{\mathcal{A}'}$ if (1) or (2) holds. (1) α has the form $\gamma \sqsubseteq \neg\eta$ or Fun(S) and each name in α occurs in $(\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')$; (2) α has the form $\gamma \sqsubseteq \eta$ and the name in γ occurs in $(\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')$;
2. Iterate step 1, until $\mathcal{T}|_{\mathcal{A}'}$ do not change anymore.

The correctness of the procedure illustrated in Definition 5 in terms of satisfiability checking and conjunctive query answering can be guaranteed by the following lemma which indicates that for subset $\mathcal{A}'$, considering axioms in $\mathcal{T}|_{\mathcal{A}'}$ is sufficient for the considered reasoning tasks.

Lemma 1. *For a DL-Lite$_\mathcal{A}$ KB $(\mathcal{T}, \mathcal{A})$ and a subset $\mathcal{A}'$ of $\mathcal{A}$, we can get that:*

1. *$(\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')$ is satisfiable iff $(\mathcal{T}, \mathcal{A}')$ is satisfiable;*
2. *$\mathsf{ans}(Q, (\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')) = \mathsf{ans}(Q, (\mathcal{T}, \mathcal{A}'))$ for each conjunctive query Q.*

Proof. By $\mathcal{T}|_{\mathcal{A}'} \subseteq \mathcal{T}$, the ($\Leftarrow$) direction of 1 and the ($\subseteq$) direction of 2 hold trivially. Next, we prove the other directions of 1 and 2. For convenience, we use $\mathcal{K}_1$ and $\mathcal{K}_2$ to denote $(\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')$ and $(\mathcal{T}, \mathcal{A}')$, respectively.

(1. $\Rightarrow$) $\mathcal{K}_1$ is satisfiable, so it has a canonical model $\mathcal{I}$ [27].[4] Next, we show $\mathcal{I}$ is also a model of $\mathcal{K}_2$. $\mathcal{K}_1$ and $\mathcal{K}_2$ have the same ABox, so $\mathcal{I}$ satisfies all the assertions in $\mathcal{K}_2$. For each axiom $A \sqsubseteq C$ in $\mathcal{K}_2$, if it occurs in $\mathcal{K}_1$ then $\mathcal{I} \models A \sqsubseteq C$ holds, otherwise, we know A is not used as class in $\mathcal{K}_2$, so $A^\mathcal{I} = \emptyset$, thus $\mathcal{I} \models A \sqsubseteq C$ holds. Other types of axioms in $\mathcal{K}_2$ can be proved analogously. So $\mathcal{I}$ is also a model of $\mathcal{K}_2$. Thus $\mathcal{K}_2$ is satisfiable. (2. $\supseteq$) If $\mathcal{K}_1$ is not satisfiable, this direction holds trivially. Next, we assume $\mathcal{K}_1$ is satisfiable. By (1. $\Rightarrow$), $\mathcal{K}_2$ is

[4] For each DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$, there exists a canonical interpretation $\mathcal{I}$ such that $\mathcal{K}$ is satisfiable iff $\mathcal{I} \models \mathcal{K}$, and if $\mathcal{K}$ is satisfiable then for each conjunctive query Q, $\vec{u} \in \mathsf{ans}(Q, \mathcal{K})$ iff $\mathcal{I} \models Q(\vec{u})$. If $\mathcal{K}$ is satisfiable, $\mathcal{I}$ is called a canonical model of $\mathcal{K}$.

satisfiable. According to (1. $\Rightarrow$) and $\mathcal{T}|_{\mathcal{A}'} \subseteq \mathcal{T}$, we can get that each canonical model of $\mathcal{K}_1$ is also a model of $\mathcal{K}_2$ and vice versa. Thus this direction holds. □

For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and an assertion α in $\mathcal{A}$, we say α is a *redundant assertion* in $\mathcal{K}$ if there exists another individual assertion α' in $\mathcal{K}$ such that α can be entailed by $\mathcal{T}$ and α', that is, if $(\mathcal{T}, \{\alpha'\}) \models \alpha$. As shown in the example below, due to redundant individual assertions in a KB, the TBox constructed by the way described in Definition 5 may not be the *least* TBox (measured by set containment or set size) for a given sub-ABox.

Example 3. Consider the DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where:

$$\mathcal{T} = \{A \sqsubseteq B, P \sqsubseteq S\}, \quad \mathcal{A} = \{A(a), B(a), P(b, c)\}$$

In $\mathcal{K}$, $B(a)$ is a redundant assertion, since it is entailed by $A \sqsubseteq B$ and $A(a)$. For the following sub-ABox $\mathcal{A}'$, a sub-TBox $\mathcal{T}|_{\mathcal{A}'}$ can be constructed by Definition 5:

$$\mathcal{A}' = \{A(a), B(a)\}, \quad \mathcal{T}|_{\mathcal{A}'} = \{A \sqsubseteq B\}$$

Obviously, $(\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')$ and $(\emptyset, \mathcal{A}')$ coincide in terms of the considered reasoning tasks, thus $\mathcal{T}|_{\mathcal{A}'}$ is not the least TBox for $\mathcal{A}'$.

How to compute the least TBox for a given sub-ABox is beyond the scope of this paper, and we leave it for our future work.

Based on Definition 5 and Lemma 1, the definition of DL-Lite$_\mathcal{A}$ KB partitions as well as the partitions with the local feature of satisfiability checking and simple-query answering is formally illustrated in the definition below.

Definition 6. For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we say that the set:

$$\mathcal{S} = \{(\mathcal{T}_1, \mathcal{A}_1), \ldots, (\mathcal{T}_n, \mathcal{A}_n)\}$$

of DL-Lite$_\mathcal{A}$ KBs is a partition of $\mathcal{K}$ if $\mathcal{A} = \cup_{i=1}^{n} \mathcal{A}_i$ and $\mathcal{T}_i = \mathcal{T}|_{\mathcal{A}_i}$ for $1 \leq i \leq n$. We say that $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$ iff the following two conditions hold:

1. $\mathcal{K}$ is satisfiable iff each KB in $\mathcal{S}$ is satisfiable;
2. If $\mathcal{K}$ is satisfiable then $\mathsf{ans}(Q, \mathcal{K}) = \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(Q, \mathcal{K}')$ for each simple-query Q.

For generality, we do not require the ABoxes of the KBs in a partition to be pairwise disjoint. Moreover, in the perspective of efficiency, $\mathcal{T}|_{\mathcal{A}_i}$ is used as the TBox of the sub-KB $(\mathcal{T}_i, \mathcal{A}_i)$ rather than the original TBox $\mathcal{T}$.

4.2. Conditions of detecting SCSQA-local KB partitions

By analyzing the procedures of satisfiability checking and conjunctive query rewriting [26,27], we found that in DL-Lite$_\mathcal{A}$, both satisfiability checking and simple-query answering can eventually be reduced to evaluating simple-queries over the ABoxes of DL-Lite$_\mathcal{A}$ KBs. This conclusion is explicitly illustrated in the following two lemmas.

Lemma 2. *For each DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, there exists a set $\mathbb{Q}$ of simple-queries such that $\mathcal{K}$ is satisfiable iff $\cup_{Q \in \mathbb{Q}} \mathsf{ans}(Q, (\emptyset, \mathcal{A})) = \emptyset$.*

Proof. Let $\mathbb{Q}$ be the set of queries corresponding to the negative closure $\mathtt{cln}(\mathcal{T})$ of $\mathcal{T}$ [27] and the functional role assertions $\mathsf{Fun}(P)$ in $\mathcal{T}$. For example, if $\mathtt{cln}(\mathcal{T})$ contains the axioms $A \sqsubseteq \neg B$ and $\mathsf{Fun}(P)$ then $\mathbb{Q}$ contains the query $A(?x) \wedge B(?x) \rightarrow q()$ and $P(?x, ?y) \wedge P(?x, ?z) \wedge ?y \neq ?z \rightarrow q()$, where A, B and P are names. By the constructing approach in [27], we can get that $\mathbb{Q}$ is a set of simple-queries and $\mathcal{K}$ is satisfiable iff $\mathsf{ans}(Q, (\emptyset, \mathcal{A})) = \emptyset$ for each $Q \in \mathbb{Q}$. So this lemma holds. □

Lemma 3. *For each satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and simple-query Q, there exists a set $\mathbb{Q}$ of simple-queries such that $\mathsf{ans}(Q, \mathcal{K}) = \cup_{Q' \in \mathbb{Q}} \mathsf{ans}(Q', (\emptyset, \mathcal{A}))$.*

Proof. By the query rewritten procedure in [27], it is not difficult to find out that for each simple-query Q, by applying an inclusion axiom in $\mathcal{T}$ over a query atom in Q or unifying two query atoms in Q, the resultant query is still a simple-query. So, we can get that $\mathbb{Q} = \mathtt{PerfectRef}(Q, \mathcal{T})$ is a set of simple-queries. $\mathsf{ans}(Q, \mathcal{K}) = \cup_{Q' \in \mathbb{Q}} \mathsf{ans}(Q', (\emptyset, \mathcal{A}))$ holds. So this lemma holds. □

All the query atoms in a simple-query share a common individual or variable. Thus, by Lemmas 2–3, we can conclude that in the scenario of KB partitioning, for the local feature of satisfiability checking and simple-query answering, the individual assertions occurring in the original KB and sharing a common individual should be put in an identical sub-KB, i.e., these assertions cannot be distributed into different sub-KBs.

Proposition 1. *For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a partition $\mathcal{S}$ of $\mathcal{K}$, if for each $U \subseteq \mathcal{A}$ satisfying that all the assertions in U share a same individual, there exists $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $U \subseteq \mathcal{A}'$, then $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$.*

Proof. We first prove the local feature of satisfiability checking. If $\mathcal{K}$ is satisfiable then each KB in $\mathcal{S}$ is satisfiable. Suppose all the KBs in $\mathcal{S}$ are satisfiable. Next, we show $\mathcal{K}$ is satisfiable. Assume (A1) $\mathcal{K}$ is not satisfiable. By Lemma 2, we can get that there exists a simple-query Q such that $\mathsf{ans}(Q, (\emptyset, \mathcal{A})) \neq \emptyset$. So there exists $U \subseteq \mathcal{A}$ satisfying that all the assertions in U share a common individual and $\mathsf{ans}(Q, (\emptyset, U)) \neq \emptyset$. By the condition in this proposition, we can get that there exists $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $U \subseteq \mathcal{A}'$. Obviously, $(\mathcal{T}, \mathcal{A}')$ is not satisfiable. Then by Lemma 1, we can conclude that $(\mathcal{T}', \mathcal{A}')$ is not satisfiable. This contradicts with the supposition that all the KBs in $\mathcal{S}$ are satisfiable. So (A1) does not hold. Thus $\mathcal{K}$ is satisfiable.

Suppose $\mathcal{K}$ is satisfiable. Next, we prove the local feature of simple-query answering. Let Q be an arbitrary simple-query. $\cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(Q, \mathcal{K}') \subseteq \mathsf{ans}(Q, \mathcal{K})$ holds trivially. Assume (A2) there exists $\vec{u} \in \mathsf{ans}(Q, \mathcal{K}) - \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(Q, \mathcal{K}')$. Then there exists $Q' \in \mathtt{PerfectRef}(Q, \mathcal{T})$ so that $\vec{u} \in \mathsf{ans}(Q', (\emptyset, \mathcal{A}))$. By the proof of Lemma 3, we know Q' is a simple-query. So there exists $U \subseteq \mathcal{A}$ satisfying that all the assertions in U share a common individual and $\vec{u} \in \mathsf{ans}(Q', (\emptyset, U))$. By the condition in this lemma, we can get that there exists $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $U \subseteq \mathcal{A}'$. So $\vec{u} \in \mathsf{ans}(Q, (\mathcal{T}, \mathcal{A}'))$ holds. Then by Lemma 1, we can further obtain that $\vec{u} \in \mathsf{ans}(Q, (\mathcal{T}', \mathcal{A}'))$. This contradicts with (A2). So (A2) does not hold. Thus $\mathsf{ans}(Q, \mathcal{K}) \subseteq \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(Q, \mathcal{K}')$ holds. Hence $\mathsf{ans}(Q, \mathcal{K}) = \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(Q, \mathcal{K}')$ holds. Therefore, $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. □

Example 4. Consider the following DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and its two partitions $\mathcal{S}_1$ and $\mathcal{S}_2$:

$$\mathcal{K} = (\{A \sqsubseteq B\}, \{A(a), P(a, b), B(c)\})$$
$$\mathcal{S}_1 = \{(\{A \sqsubseteq B\}, \{A(a), P(a, b)\}), (\{\}, \{B(c)\})\}$$
$$\mathcal{S}_2 = \{(\{A \sqsubseteq B\}, \{A(a)\}), (\{\}, \{P(a, b), B(c)\})\}$$

All the assertions in $\mathcal{K}$ sharing the same individual are contained in the same sub-KB in $\mathcal{S}_1$. Thus, by Proposition 1, we can conclude that $\mathcal{S}_1$ is a SCSQA-local partition of $\mathcal{K}$. On the other hand, $\mathcal{S}_2$ is not a SCSQA-local partition of $\mathcal{K}$, since it does not have the local feature of the following simple-query Q:

$$A(?x) \wedge P(?x, ?y) \rightarrow q(?x, ?y)$$

from the fact that $\mathsf{ans}(Q, \mathcal{K}) = \{(a, b)\}$ but $\cup_{\mathcal{K}' \in \mathcal{S}_2} \mathsf{ans}(Q, \mathcal{K}') = \emptyset$. Even though $\mathcal{S}_2$ doe not satisfy the condition in Proposition 1, the conclusion that $\mathcal{S}_2$ is not a SCSQA-local partition of $\mathcal{K}$ cannot be obtained, since Proposition 1 is just a sufficient condition.

Note that the condition presented in Proposition 1 cannot be used as a necessary condition due to the redundant individual assertions in DL-Lite$_\mathcal{A}$ KBs, as illustrated by the following example.

Example 5. Let $\mathcal{K}'$ be the KB obtained by adding the assertion $B(a)$ to the KB in Example 4. In $\mathcal{K}'$, $B(a)$ is a redundant assertion since it is entailed by $A \sqsubseteq B$ and $A(a)$. Consider the following partition $\mathcal{S}$ of $\mathcal{K}'$:

$$\mathcal{S} = \{(\{A \sqsubseteq B\},\ \{A(a), P(a,b)\}),\ (\{\},\ \{B(a), B(c)\})\}$$

It is not difficult to validate that $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. However, $\mathcal{S}$ does not satisfy the condition in Proposition 1, since the assertions containing a are distributed in different sub-KBs in $\mathcal{S}$. Actually, if a partition $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}'$, no matter where to put $B(a)$ among the sub-KBs in $\mathcal{S}$, the resultant partition is still a SCSQA-local partition. This means that the positioning of redundant assertions does not affect the result of KB partitioning.

Before showing the situations that make the condition in Proposition 1 to become sufficient and necessary, we first provide a sufficient and necessary condition for individual assertion entailment checking.

Lemma 4. *For a satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and individual assertion α, then $\mathcal{K} \models \alpha$ iff there exists $\alpha' \in \mathcal{A}$ such that α and α' share a same individual and $(\mathcal{T}, \{\alpha'\}) \models \alpha$.*

Proof. Let Q be the query $\alpha \rightarrow q()$. Then it holds trivially that $\mathcal{K} \models \alpha$ iff $\mathsf{ans}(Q, \mathcal{K}) \neq \emptyset$. It is not difficult to obtain that each query in $\mathtt{PerfectRef}(Q, \mathcal{T})$ has only one query atom and this atom contains all the individuals occurring in α. Thus, we can get that $\mathcal{K} \models \alpha$ iff there exists $\alpha' \in \mathcal{A}$ that shares an individual with α and $\cup_{Q' \in \mathtt{PerfectRef}(Q,\mathcal{T})}\mathsf{ans}(Q', (\emptyset, \{\alpha'\})) \neq \emptyset$. Then we can further obtain that $\mathcal{K} \models \alpha$ iff there exists $\alpha' \in \mathcal{A}$ such that α and α' share an individual and $(\mathcal{T}, \{\alpha'\}) \models \alpha$. So this lemma holds. □

As illustrated in the theorem below, if a DL-Lite$_\mathcal{A}$ KB does not contain redundant individual assertions, then the condition in Proposition 1 can be used as a sufficient and necessary condition.

Theorem 1. *For a satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ without redundant individual assertions, then a partition $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$ iff for each subset $U \subseteq \mathcal{A}$ such that all the assertions in U share a common individual, there exists $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $U \subseteq \mathcal{A}'$.*

Proof. By Proposition 1, the ($\Leftarrow$) direction holds. Next, we show the ($\Rightarrow$) direction. $\mathcal{K}$ is satisfiable then all the KBs in $\mathcal{S}$ are satisfiable. Assume (A) there exists $\mathrm{U} \subseteq \mathcal{A}$ such that all the assertions in U share a common individual and there does not exist $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $\mathrm{U} \subseteq \mathcal{A}'$. Let Q be the query $\wedge_{\alpha \in \mathrm{U}}\alpha \rightarrow q()$. Obviously Q is a simple-query, and $\mathsf{ans}(Q, \mathcal{K}) = \{()\}$, i.e., Q is true over $\mathcal{K}$. $\mathcal{K}$ is a KB without redundant individual assertions. Then by Lemma 4, we can further obtain that for each $\alpha \in \mathrm{U}$, $(\mathcal{T}, \mathcal{A} - \{\alpha\}) \not\models \alpha$. Thus by the assumption (A), we can get that there does not exist $\mathcal{K}' \in \mathcal{S}$ such that $\mathcal{K}' \models \alpha$ for each $\alpha \in \mathrm{U}$. So for each $\mathcal{K}' \in \mathcal{S}$, $\mathsf{ans}(Q, \mathcal{K}') = \emptyset$, i.e., Q' is false over $\mathcal{K}'$. This contradicts with the fact that $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. Thus the assumption (A) does not hold. So the ($\Rightarrow$) direction of this theorem holds. Therefore this theorem holds. □

The KB in Example 4 is satisfiable and does not contain redundant individual assertions. Thus by Theorem 1, we can directly obtain that the partition $\mathcal{S}_2$ in Example 4 is not a SCSQA-local partition of this KB.

Redundant individual assertions in a KB do not affect entailment. Based on this, we can obtain a more general sufficient and necessary condition for detecting SCSQA-local partitions, shown in the theorem below which indicates that a partition is a SCSQA-local partition of a DL-Lite$_\mathcal{A}$ KB iff all the assertions sharing a common individual and entailed by the original KB can be entailed by a same sub-KB.

Theorem 2. *A partition $\mathcal{S}$ of a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is a SCSQA-local partition of $\mathcal{K}$ iff for each individual assertion set U satisfying that all the assertions in U share a common individual and $\mathcal{K} \models \alpha$ for each $\alpha \in U$, there exists $\mathcal{K}' \in \mathcal{S}$ such that $\mathcal{K}' \models \alpha$ for each $\alpha \in U$.*

Proof. ($\Rightarrow$) If $\mathcal{K}$ is not satisfiable then some KBs in $\mathcal{S}$ are not satisfiable. Then this direction holds trivially. Next, we assume $\mathcal{K}$ is satisfiable. So all the KBs in $\mathcal{S}$ are satisfiable. Suppose (S) there exists an individual assertion set U satisfying that all the assertions in U share a common individual and are entailed by $\mathcal{K}$, but there does not exist $\mathcal{K}' \in \mathcal{S}$ such that $\mathcal{K}' \models \alpha$ for each $\alpha \in \mathrm{U}$. Let Q be the query $\wedge_{\alpha \in \mathrm{U}}\alpha \rightarrow q()$. Then Q is a simple-query and $\mathsf{ans}(Q, \mathcal{K}) = \{()\}$. By (S), we can get that $\cup_{\mathcal{K}' \in \mathcal{S}}\mathsf{ans}(Q, \mathcal{K}') = \emptyset$. This contradicts with the fact that $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. So (S) does not hold. Thus this direction holds.

($\Leftarrow$) If $\mathcal{K}$ is satisfiable then all the KBs in $\mathcal{S}$ are satisfiable. Suppose all the KBs in $\mathcal{S}$ are satisfiable. Assume (A1) $\mathcal{K}$ is not satisfiable. Let $a \in \boldsymbol{I}$ and $A \in \boldsymbol{C}$ be two names not occurring in $\mathcal{K}$. Then $\mathcal{K} \models A(a)$ holds. All the KBs in $\mathcal{S}$ are satisfiable. So $\mathcal{K}' \not\models A(a)$ holds for each $\mathcal{K}' \in \mathcal{S}$. This contradicts with the condition in the theorem. So (A1) does not hold. Thus $\mathcal{K}$ is satisfiable. Hence $\mathcal{S}$ has the local feature of satisfiability checking.

Assume $\mathcal{K}$ is satisfiable. Next, we prove the local feature of simple-query answering. Let $\mathcal{R}$ be a subset of $\mathcal{A}$ satisfying that for each $\alpha \in \mathcal{R}$ there exists $\alpha' \in \mathcal{A} - \mathcal{R}$ such that $(\mathcal{T}, \{\alpha'\}) \models \alpha$, and for each $\alpha' \in \mathcal{A} - \mathcal{R}$ there does not exist $\alpha'' \in \mathcal{A} - \mathcal{R} - \{\alpha'\}$ such that $(\mathcal{T}, \{\alpha''\}) \models \alpha'$. Thus $\mathcal{K}_o = (\mathcal{T}, \mathcal{A} - \mathcal{R})$ is a KB without redundant individual assertions. Let $\mathcal{S}'$ be the partition obtained by dropping all the assertions in $\mathcal{R}$ from the KBs in $\mathcal{S}$. Next, we show $\mathcal{S}'$ is a SCSQA-local partition of $\mathcal{K}_o$. Assume (A2) there exists $\mathrm{U} \subseteq \mathcal{A} - \mathcal{R}$ satisfying that all the assertions in U share a common individual and there does not exist $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $\mathrm{U} \subseteq \mathcal{A}'$. From $\mathrm{U} \subseteq \mathcal{A} - \mathcal{R}$ and Lemma 4, we can get that there does not exist $\mathcal{K}' \in \mathcal{S}$ such that $\mathcal{K}' \models \alpha$ for each $\alpha \in \mathrm{U}$. This contradicts with the condition in the theorem. So (A2) does not hold. This means that all the assertions in $\mathcal{A} - \mathcal{R}$ sharing a common individual are contained in the same sub-KB in $\mathcal{S}$. $\mathcal{S}'$ is obtained by dropping all the assertions in U from the KBs in $\mathcal{S}$. So all the assertions in $\mathcal{A} - \mathcal{R}$ sharing a common individual are contained in the same sub-KB in $\mathcal{S}'$. Thus by Theorem 1, $\mathcal{S}'$ is a SCSQA-local partition of $\mathcal{K}_o$. Let Q be an arbitrary simple-query. Then $\mathsf{ans}(Q, \mathcal{K}_o) = \cup_{\mathcal{K}' \in \mathcal{S}'}\mathsf{ans}(Q, \mathcal{K}')$ holds. Obviously, $\mathsf{ans}(Q, \mathcal{K}_o) = \mathsf{ans}(Q, \mathcal{K})$ and $\cup_{\mathcal{K}' \in \mathcal{S}'}\mathsf{ans}(Q, \mathcal{K}') \subseteq \cup_{\mathcal{K}' \in \mathcal{S}}\mathsf{ans}(Q, \mathcal{K}')$ hold. Thus $\mathsf{ans}(Q, \mathcal{K}) \subseteq \cup_{\mathcal{K}' \in \mathcal{S}}\mathsf{ans}(Q, \mathcal{K}')$ holds. The relation $\cup_{\mathcal{K}' \in \mathcal{S}}\mathsf{ans}(Q, \mathcal{K}') \subseteq \mathsf{ans}(Q, \mathcal{K})$ holds trivially. So $\mathcal{S}$ has the local feature of simple-query answering.

Therefore $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. □

Example 6. Consider the KB $\mathcal{K}$ and its partitions $\mathcal{S}_1$ and $\mathcal{S}_2$ in Example 4 again. By Theorem 2, we can directly obtain that $\mathcal{S}_1$ is a SCSQA-local partition of $\mathcal{K}$, while $\mathcal{S}_2$ is not.

4.3. A concrete way of computing SCSQA-local KB partitions

Based on the theoretical results in Section 4.2, a SCSQA-local partition $\mathcal{S}$ of a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ can be obtained by first computing a partition of the ABox then generating a TBox for each sub-ABox. Concretely:

1. Compute a partition $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of $\mathcal{A}$ by the following two steps:

a. For each individual a of $\mathcal{K}$, compute the set $\mathsf{U}_{\{a\}}$ consisting of all the assertions occurring in $\mathcal{A}$ and containing a;
b. Let U be the set consisting of all the assertion sets computed in step 1.a. Compute a partition $\mathsf{U}_1, \ldots, \mathsf{U}_n$ of U. For each U_i, let $\mathcal{A}_i = \cup_{\mathsf{U}_{\{a\}} \in \mathsf{U}_i} \mathsf{U}_{\{a\}}$.

2. For each $1 \leq i \leq n$, compute the sub-TBox $\mathcal{T}|_{\mathcal{A}_i}$ for $\mathcal{A}_i$. Let $\mathcal{S} = \cup_{i=1}^{n} \{(\mathcal{T}|_{\mathcal{A}_i}, \mathcal{A}_i)\}$.

For the local feature of satisfiability checking and simple-query answering, the assertions sharing a common individual should be put in the same sub-KB. Thus, in step 1, we first group the assertions in $\mathcal{A}$ based on the individuals they share (step 1.a). In order to avoid the KB partitions containing too many sub-KBs, we choose to merge some assertion groups to generate the ABox of a sub-KB (step 1.b). Note that, the different ways of merging assertion groups, i.e., the different ways of partitioning U in step 1.b, lead to different SCSQA-local partitions of $\mathcal{K}$. Undoubtedly, in the terms of distribution and parallelization, the partitions generating more balanced and smaller sized sub-ABoxes are more likely to achieve the efficiency improvement.

For a desired size *asize* of ABoxes in a KB partition, in the above step 1.b, a partition of U that generates sub-ABoxes sized no more than *asize* can be realized by the following two steps:

b1. Compute a tuple (order) $\mathsf{O} = \mathsf{U}_1, \ldots, \mathsf{U}_m$ of the elements in U such that $|\mathsf{U}_i| \leq |\mathsf{U}_{i+1}|$ for $1 \leq i \leq m-1$.
b2. Continue the following procedure P, until $|\mathsf{O}| = 0$, i.e., O does not contain any element.

P. Generate a sub-ABox $\mathcal{A}'$. Let $\mathcal{A}' = \emptyset$. For i from $|\mathsf{O}|$ to 1, if $|\mathcal{A}' \cup \mathsf{O}[i]| \leq$ *asize* then let $\mathcal{A}' = \mathcal{A}' \cup \mathsf{O}[i]$; otherwise for j from 1 to $i-1$, if $|\mathcal{A}' \cup \mathsf{O}[j]| \leq$ *asize* then let $\mathcal{A}' = \mathcal{A}' \cup \mathsf{O}[j]$; otherwise break the first loop. Remove all the elements used to generate $\mathcal{A}'$ from O.

Theoretical studies on computing balanced and smaller-sized sub-KBs will be discussed in our future work.

Example 7. Consider the KB $\mathcal{K}$ in Example 4 again. $\mathcal{K}$ totally contains three different individuals a, b and c. Thus three individual assertions $\mathsf{U}_{\{a\}}$, $\mathsf{U}_{\{b\}}$ and $\mathsf{U}_{\{c\}}$ which respectively consist of all the assertions sharing a, b and c can be computed:

$$\mathsf{U}_{\{a\}}\colon \{A(a), P(a,b)\},\quad \mathsf{U}_{\{b\}}\colon \{P(a,b)\},\quad \mathsf{U}_{\{c\}}\colon \{B(c)\}$$

Different ways of partition $\{\mathsf{U}_{\{a\}}, \mathsf{U}_{\{b\}}, \mathsf{U}_{\{c\}}\}$ will lead to different SCSQA-local partitions of $\mathcal{K}$. For example, for these two partitions $\{\mathsf{U}_{\{c\}}\}, \{\mathsf{U}_{\{a\}}, \mathsf{U}_{\{b\}}\}$ and $\{\mathsf{U}_{\{b\}}, \mathsf{U}_{\{c\}}\}, \{\mathsf{U}_{\{a\}}\}$, the following two SCSQA-local partitions $\mathcal{S}_1$ and $\mathcal{S}_2$ can be obtained:

$$\mathcal{S}_1 = \{(\{\}, \{B(c)\}),\ (\{A \sqsubseteq B\}, \{A(a), P(a,b)\})\}$$
$$\mathcal{S}_2 = \{(\{\}, \{B(c), P(a,b)\}),\ (\{A \sqsubseteq B\}, \{A(a, P(a,b))\})\}$$

Example 7 indicates that putting the assertions sharing a common individual into the same sub-KB may cause a role individual assertion occurring in two different sub-KBs in a SCSQA-local partition, such as the role assertion $P(a,b)$ in the partition $\mathcal{S}_2$ in Example 7. This is because a role individual assertion usually contains two different individuals. Anyhow, the sum of the sizes of the ABoxes of the sub-KBs in a partition will not exceed twice the size of the ABox of the original KB.

Moreover, based on the above two steps 1–2, we can further obtain the complexity of computing SCSQA-local partitions of DL-Lite$_{\mathcal{A}}$ KBs, shown in the following theorem.

Theorem 3. *For a DL-Lite$_{\mathcal{A}}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a SCSQA-local partition of $\mathcal{K}$ can be computed in $O(|\mathcal{T}|^2)$ w.r.t. $|\mathcal{T}|$ and in $O(|\mathcal{A}|)$ w.r.t. $|\mathcal{A}|$.*

Proof. This theorem holds by the following two factors. (1) All the assertion sets whose elements share a common individual of $\mathcal{K}$ can be computed in time $O(|\mathcal{A}|)$. (2) For each $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{T}|_{\mathcal{A}'}$ can be computed in $O(|\mathcal{T}|^2)$. □

Low computational complexity makes the proposed approach appealing for partitioning a large-scale DL-Lite$_{\mathcal{A}}$ KB into smaller independent sub-KBs which have the local feature of satisfiability checking and simple-query answering.

5. Conjunctive query partitioning and evaluation

In this section, we discuss the second problem P2, i.e., conjunctive query partitioning and evaluation, of the divide-and-conquer approach. As discussed in Section 3, this problem is decomposed into these three sub-problems: (P21) the definition of conjunctive query partitions; (P22) the conditions of detecting simple-query reducible queries; and (P23) the concrete ways of partitioning and evaluating simple-query reducible queries and non-simple query reducible queries. Here, we only consider satisfiable DL-Lite$_{\mathcal{A}}$ KBs.

5.1. Formalization of conjunctive query partitioning

The definition of conjunctive query partitions as well as simple-query partitions of conjunctive queries is illustrated in the definition below.

Definition 7. For a conjunctive query Q, we say that the set $\mathcal{Q} = \cup_{i=1}^{n} \{q_i\}$ of conjunctive queries is a partition of Q if:

1. $\mathsf{bd}(Q) = \cup_{i=1}^{n} \mathsf{bd}(q_i)$, and $\mathsf{bd}(q_i) \cap \mathsf{bd}(q_j) = \emptyset$ for $1 \leq i, j \leq n$ and $i \neq j$;
2. For each $1 \leq i \leq n$, $\mathsf{hd}(q_i)$ consists of all the variables occurring in $\mathsf{bd}(q_i)$ that also occur either in $\mathsf{hd}(Q)$ or in $\cup_{k=1,k\neq i}^{n} \mathsf{bd}(q_k)$.

We say that $\mathcal{Q}$ is a simple-query partition of Q iff all the queries in $\mathcal{Q}$ are simple-queries.

For the sake of efficiency, we hope that each query atom in the original query is evaluated only once. Therefore, in the first condition of Definition 7, we require the bodies of sub-queries are pairwise disjoint. The second condition of Definition 7 implies that the head of each sub-query consists of two variable parts. The *first part* are the variables occurring both in the body of this sub-query and the head of the original query, while the *second part* are those shared by the body of this sub-query and the bodies of some other sub-queries. This means that the heads of some sub-queries may contain variables solely occurring in the body of the original query. The motivation of such a design is to guarantee that the same variables present in different sub-queries are bound to the same values. As shown in the example below, without the second variable part, we cannot use the certain answers of the sub-queries to generate the certain answers of the original query.

Example 8. Consider the following conjunctive query Q, conjunctive query set $\mathcal{Q}$ and DL-Lite$_{\mathcal{A}}$ KB $\mathcal{K}$:

$$Q\ :\ P(?x, ?y) \wedge S(?x, ?y) \rightarrow q(?x)$$
$$\mathcal{Q} = \{q_1\colon P(?x, ?y) \rightarrow q(?x),\quad q_2\colon S(?x, ?y) \rightarrow q(?x)\}$$
$$\mathcal{K} = (\{\},\ \{P(a,b), S(a,c)\})$$

Obviously, (a) is both a certain answer of q_1 and q_2 over $\mathcal{K}$. However, (a) is not a certain answer of Q over $\mathcal{K}$. Thus the certain answers of q_1 and q_2 cannot be used to generate the certain answers of Q. This is because $?y$ may be bound to different values to obtain the certain answers of q_1 and q_2, respectively.

The sub-queries in a query partition will be evaluated independently. Thus the unique way of deciding whether the variables shared by different sub-queries are bound to the same values is to put the shared variables to the heads of the corresponding sub-queries.

The concrete way of merging the certain answers of the sub-queries to generate the certain answers of the original query is presented in the definition below.

Definition 8. For a partition $\mathcal{Q} = \cup_{i=1}^{n}\{q_i\}$ of a conjunctive query Q and partition $\mathcal{S}$ of a DL-Lite$_\mathcal{A}$ KB, we define:

$$\mathsf{ans}(Q, \mathcal{Q}, \mathcal{S}) = \{\vec{x}[\vec{x}_1/\vec{u}_1, \ldots, \vec{x}_n/\vec{u}_n] | (\vec{u}_1, \ldots, \vec{u}_n) \in \Lambda_1 \times \cdots \times \Lambda_n\}$$

where $\vec{x} = \mathsf{hd}(Q)$, for each $1 \leq i \leq n$, $\vec{x}_i = \mathsf{hd}(q_i)$ and $\Lambda_i = \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(q_i, \mathcal{K}')$, and for each $1 \leq i, j \leq n$, $1 \leq k \leq |\vec{u}_i|$ and $1 \leq l \leq |\vec{u}_j|$, if $\vec{x}_i[k] = \vec{x}_j[l]$ then $\vec{u}_i[k] = \vec{u}_j[l]$.

The correctness of this way of merging the certain answers of sub-queries can be guaranteed by the lemma below.

Lemma 5. *Given a conjunctive query Q and satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$, then for each partition $\mathcal{Q}$ of Q and partition $\mathcal{S}$ of $\mathcal{K}$, $\mathsf{ans}(Q, \mathcal{Q}, \mathcal{S}) \subseteq \mathsf{ans}(Q, \mathcal{K})$ holds.*

Proof. Let $\vec{u} \in \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$. Suppose $\mathcal{Q} = \cup_{i=1}^{n}\{q_i\}$. Then there exists $\vec{u}_i \in \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(q_i, \mathcal{K}')$ for each $q_i \in \mathcal{Q}$ such that $\vec{u} = \mathsf{hd}(Q)[\mathsf{hd}(q_1)/\vec{u}_1, \ldots, \mathsf{hd}(q_n)/\vec{u}_n]$ holds. For each $q_i \in \mathcal{Q}$, $\vec{u}_i \in \mathsf{ans}(q_i, \mathcal{K})$ holds trivially. Next, we show $\vec{u} \in \mathsf{ans}(Q, \mathcal{K})$. Let $\mathcal{I}$ be a canonical model of $\mathcal{K}$. Then there exists a binding π_i of $q_i(\vec{u}_i)$ over $\mathcal{K}$ such that $\mathcal{I}, \pi_i \models q_i(\vec{u}_i)$. From $\pi_1 - \pi_n$, a binding π of $Q(\vec{u})$ over $\mathcal{I}$ can be constructed by the following settings. For each individual a, let $\pi(a) = a$. For each variable x in $Q(\vec{u})$, there exists q_i containing x. If x occurs in the lth position of $\mathsf{hd}(q_i)$ then let $\pi(x) = \vec{u}_i[k]$ else let $\pi(x) = \pi_i(x)$. By the construction of π and $\vec{u}$, it holds trivially that $\mathcal{I}, \pi \models Q(\vec{u})$. So $\vec{u} \in \mathsf{ans}(Q, \mathcal{K})$. So this lemma holds. □

Finally, we formalize the notion of *simple-query reducible conjunctive queries*. By the semantics of conjunctive queries, we know that distinguished variables and non-distinguished variables of conjunctive queries are interpreted in different ways. Concretely, distinguished variables can solely be bound to names, whereas non-distinguished variables can be bound to not only names but also anonymous individuals implied to exist. Based on this, we say that a conjunctive query is simple-query reducible iff it has a simple-query partition in which each sub-query does not contain any non-distinguished variables of the original query as distinguished variables, i.e., the semantics of the variables of the original query are preserved.

Definition 9. We say that a conjunctive query Q is simple-query reducible iff Q has a simple-query partition $\mathcal{Q}$ such that $\mathsf{hd}(q) \subseteq \mathsf{hd}(Q)$ holds for each $q \in \mathcal{Q}$.

The lemma below indicates that if a query is simple-query reducible then answering this query over a satisfiable DL-Lite$_\mathcal{A}$ KB can be soundly and completely realized by evaluating a simple-query partition of this query over a SCSQA-local partition of this KB.

Lemma 6. *For a conjunctive query Q, if Q is simple-query reducible then Q has a simple-query partition $\mathcal{Q}$ such that for each satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and each SCSQA-local partition $\mathcal{S}$ of $\mathcal{K}$, $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$ holds.*

Proof. If Q is simple-query reducible then by Definition 9, Q has a simple-query partition $\mathcal{Q} = \cup_{i=1}^{m}\{q_i\}$ such that $\mathsf{hd}(q_i) \subseteq \mathsf{hd}(Q)$ for each $q_i \in \mathcal{Q}$. Next, we show the equation (E) $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$ holds. Let $\vec{u} \in \mathsf{ans}(Q, \mathcal{K})$, and let $\vec{u}_i = \mathsf{hd}(q_i)[\mathsf{hd}(Q)/\vec{u}]$ for each $q_i \in \mathcal{Q}$. $\mathsf{hd}(q_i) \subseteq \mathsf{hd}(Q)$ holds for each $q_i \in \mathcal{Q}$. So we can get that $\vec{u}_1 - \vec{u}_m$ are tuples of names and $\vec{u} = \mathsf{hd}(Q)[\mathsf{hd}(q_1)/\vec{u}_1, \ldots, \mathsf{hd}(q_m)/\vec{u}_m]$ holds. For each $q_i \in \mathcal{Q}$, $\vec{u}_i \in \mathsf{ans}(q_i, \mathcal{K})$ holds trivially. $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. So $\vec{u}_i \in \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{ans}(q_i, \mathcal{K}')$ holds. Thus $\vec{u} \in \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$ holds. Hence $\mathsf{ans}(Q, \mathcal{K}) \subseteq \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$. Then by Lemma 5, equation (E) holds. Therefore this lemma holds. □

5.2. Conditions of detecting simple-query reducible queries

Not all conjunctive queries are simple-query reducible, as shown in the following example. This indicates that there exist queries and satisfiable DL-Lite$_\mathcal{A}$ KBs such that evaluating any simple-query partitions of these queries over any SCSQA-local partitions of these KBs cannot obtain all the certain answers of the original queries over the original KBs.

Example 9. Consider the following conjunctive query Q:

Q: $A(?x) \wedge P(?x, ?y) \wedge B(?y) \rightarrow q()$

Q has a total of three simple-query partitions as follows:

$$\mathcal{Q}_1 = \{A(?x) \wedge P(?x, ?y) \rightarrow q(?y),\ \ B(?y) \rightarrow q(?y)\}$$
$$\mathcal{Q}_2 = \{A(?x) \rightarrow q(?x),\ \ P(?x, ?y) \wedge B(?y) \rightarrow q(?x)\}$$
$$\mathcal{Q}_3 = \{A(?x) \rightarrow q(?x),\ \ P(?x, ?y) \rightarrow q(?x, ?y),\ \ B(?y) \rightarrow q(?y)\}$$

By Definition 9, Q is not simple-query reducible, since each $\mathcal{Q}_i$ contains sub-queries using non-distinguished variables of Q, for instance $?y$, as distinguished variables. Consider the following DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$:

$$\mathcal{K} = (\{C \sqsubseteq \exists S,\ \exists S^- \sqsubseteq A,\ A \sqsubseteq \exists P,\ \exists P^- \sqsubseteq B\},\ \{C(a)\})$$

Obviously, $\mathcal{S} = \{\mathcal{K}\}$ is a SCSQA-local partition of $\mathcal{K}$, and $\mathsf{ans}(Q, \mathcal{Q}_i, \mathcal{S}) = \emptyset$ for each $1 \leq i \leq 3$. However, $\mathsf{ans}(Q, \mathcal{K}) = \{()\}$, i.e., Q is true over $\mathcal{K}$. This means that if a conjunctive query Q is not simple-query reducible then there may exist a KB such that evaluating any simple-query partition of Q cannot obtain all the certain answers of Q over the KB.

Next, we provide a sufficient and necessary condition to decide whether a conjunctive query is simple-query reducible. By the definition of query partitions (Definition 5), we can get that for a conjunctive query Q and a simple-query partition $\mathcal{Q}$ of Q, if all the atoms sharing a common non-distinguished variable of Q are put into the same sub-query in $\mathcal{Q}$, then all the queries in $\mathcal{Q}$ will not contain any non-distinguished variables of Q as distinguished variables, i.e., the heads of the queries in $\mathcal{Q}$ will not contain the variables solely occurring in the body of Q. Based on this, we can obtain a more intuitive condition of deciding whether a conjunctive query is simple-query reducible without computing and checking any simple-query partition of this query.

Theorem 4. *A conjunctive query Q is simple-query reducible iff there do not exist three query atoms $P(?x, ?y)$, α and α' of Q such that $?x$ and $?y$ are non-distinguished variables of Q, α contains $?x$ but not $?y$ and α' contains $?y$ but not $?x$.*

Proof. ($\Rightarrow$) Assume (A) Q contains three atoms $P(?x, ?y)$, α and α' of Q such that $?x$ and $?y$ are non-distinguished variables of Q, α contains $?x$ but not $?y$ and α' contains $?y$ but not $?x$. Let $\mathcal{Q}$ be an arbitrary simple-query partition of Q. Then by the definition of simple-queries, these three query atoms cannot occur in a same sub-query in $\mathcal{Q}$. This means $?x$ or $?y$ is contained in the head of some sub-queries in $\mathcal{Q}$. Then by Definition 9, we can get that Q is not simple-query reducible. This contradicts with the fact that Q is simple-query reducible. So the assumption (A) does not hold. So this direction hold.

($\Leftarrow$) For each non-distinguished variable $?x$ of Q, let $U_{\{?x\}}$ be the set consisting of all the query atoms of Q that contains $?x$. By the condition, we can get that for every two different non-distinguished variables $?x$ and $?y$ of Q, $U_{\{?x\}} \subseteq U_{\{?y\}}$ or $U_{\{?x\}} \cap U_{\{?y\}} = \emptyset$ holds. If $U_{\{?x\}} \subseteq U_{\{?y\}}$ holds then merge $U_{\{?x\}}$ and $U_{\{?y\}}$ into one atom set $U_{\{?x\}} \cup U_{\{?y\}}$. Then for each query atom set U, generate a sub-query q such that $\mathsf{bd}(q) = \mathsf{U}$ and $\mathsf{hd}(q)$ consists of all the variables occurring in U that also occur in $\mathsf{hd}(Q)$ or $\mathsf{bd}(Q) - \mathsf{U}$. For each query atom α of Q that does not contain any non-distinguished variable of Q, generate a sub-query q such that $\mathsf{bd}(q) = \{\alpha\}$ and $\mathsf{hd}(q)$ consists of all the variables occurring in α that also occur in $\mathsf{hd}(Q)$ or $\mathsf{bd}(Q) - \{\alpha\}$. Let $\mathcal{Q}$ be the set consisting of all the generated sub-queries. Then $\mathcal{Q}$ is a simple-query partition of Q and each non-distinguished variable of Q does not occur in the heads of the queries in $\mathcal{Q}$. So Q is simple-query reducible. □

By Theorem 4, we can directly conclude that the conjunctive queries with no more than one non-distinguished variable are all simple-query reducible. Moreover, based on Theorem 4, we can further obtain the complexity of detecting whether a conjunctive query is simple-query reducible.

Theorem 5. *For a conjunctive query Q, the complexity of deciding whether Q is simple-query reducible is $O(|Q|)$.*

Proof. Based on Theorem 4, this theorem holds trivially. □

5.3. Concrete ways of partitioning and evaluating conjunctive queries

After presenting the conditions of determining simple-query reducible conjunctive queries, here, we present the concrete ways of partitioning and evaluating simple-query reducible queries and non-simple query reducible queries.

5.3.1. Partitioning and evaluating simple-query reducible queries

There may exist multiple simple-query partitions of a simple-query reducible conjunctive query, and those that can capture all the certain answers of the original query should be identified first before we can provide a concrete way for computing such partitions.

For a simple-query reducible conjunctive query, Lemma 6 proves the existence of simple-query partitions that can capture all the certain answers of the original query. The theorem below further illustrates which partitions can fulfill this task.

Theorem 6. *For a simple-query reducible conjunctive query Q, let $\mathcal{Q}$ be any simple-query partition of Q such that $\mathsf{hd}(q) \subseteq \mathsf{hd}(Q)$ for each $q \in \mathcal{Q}$. Then $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$ holds, for any satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and any SCSQA-local partition $\mathcal{S}$ of $\mathcal{K}$.*

Proof. For Q and $\mathcal{Q}$, $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$ can be proved analogously to the proof of Lemma 6. □

Example 10. Consider the following conjunctive query Q and its partition $\mathcal{Q}$:

$$Q : A(?x) \wedge P(?x, ?y) \wedge B(?y) \rightarrow q(?x)$$
$$\mathcal{Q} = \{A(?x) \rightarrow q(?x),\ P(?x, ?y) \wedge B(?y) \rightarrow q(?x)\}$$

By Theorem 4, Q is simple-query reducible. Moreover $\mathcal{Q}$ is a simple-query partition of Q satisfying that all the queries in $\mathcal{Q}$ do not use any non-distinguished variable of Q as distinguished variable. So, evaluating Q over any satisfiable DL-Lite$_\mathcal{A}$ KB can be realized by answering $\mathcal{Q}$ over any SCSQA-local partition of this KB.

For a simple-query reducible conjunctive query Q, a simple-query partition $\mathcal{Q}$ of Q satisfying that $\mathsf{hd}(q) \subseteq \mathsf{hd}(Q)$ for each $q \in \mathcal{Q}$ can be obtained by the following two steps.

1. Compute a partition U of the query atom set $\mathsf{bd}(Q)$ by the following two steps. Let $\mathsf{A} = \mathsf{bd}(Q)$ and $\mathsf{U} = \emptyset$.
   a. For each non-distinguished variable x of Q, compute the atom set $U_{\{x\}}$ consisting of all the query atoms in A containing x, and let $\mathsf{U} = \mathsf{U} \cup \{U_{\{x\}}\}$ and $\mathsf{A} = \mathsf{A} - U_{\{x\}}$. Then for every two different non-distinguished variables x and y of Q occurring in the same atom in Q, merge $U_{\{x\}}$ and $U_{\{y\}}$ into one set $U_{\{x\}} \cup U_{\{y\}}$ in U, i.e., replacing $U_{\{x\}}$ and $U_{\{y\}}$ with $U_{\{x\}} \cup U_{\{y\}}$.
   b. For each distinguished variable or individual x of Q, compute the atom set $U_{\{x\}}$ consisting of all the query atoms in A containing x, and let $\mathsf{U} = \mathsf{U} \cup \{U_{\{x\}}\}$ and $\mathsf{A} = \mathsf{A} - U_{\{x\}}$.
2. For each query atom set $\mathsf{U} \in \mathsf{U}$ satisfying that $|\mathsf{U}| > 0$, generate a sub-query q such that $\mathsf{bd}(q) = \mathsf{U}$ and $\mathsf{hd}(q)$ consisting of all the variables not only occurring in U but also occurring either in $\mathsf{hd}(Q)$ or in $\mathsf{bd}(Q) - \mathsf{U}$, and add q to $\mathcal{Q}$.

As shown in the example below, the order of steps 1.a and 1.b is crucial in guaranteeing that generated sub-queries do not use some non-distinguished variables of Q as distinguished variables.

Example 11. Consider the simple-query reducible conjunctive query Q in Example 10 again. Q has solely one non-distinguished variable $?y$ and distinguished variable $?x$. Then by the above step 1, two query atom sets $U_{\{?y\}}$ and $U_{\{?x\}}$ can be successively obtained after step 1:

$$U_{\{?y\}} = \{P(?x, ?y), B(?y)\},\quad U_{\{?x\}} = \{A(?x)\}$$

Then two sub-queries can be generated and the simple-query partition $\mathcal{Q}$ of Q in Example 10 can be obtained. However, if we reverse the order of steps 1.a and 1.b, the following two query atom sets will be generated by step 2:

$$U_{\{?x\}} = \{A(?x), P(?x, ?y)\},\quad U_{\{?y\}} = \{B(?y)\}$$

Then the following simple-query partition $\mathcal{Q}'$ of Q can be generated:

$$\mathcal{Q}' = \{A(?x) \wedge P(?x, ?y) \rightarrow q(?x, ?y),\quad B(?y) \rightarrow q(?y)\}$$

The non-distinguished variable $?y$ of Q is used as distinguished variable of the queries in $\mathcal{Q}'$. Thus, $\mathcal{Q}'$ cannot be used to evaluate over SCSQA-local partitions of DL-Lite$_\mathcal{A}$ KBs.

Moreover, based on the above two steps, we can further obtain the complexity of computing the desired simple-query partitions of simple-query reducible conjunctive queries.

Theorem 7. *For a simple-query reducible conjunctive query Q, a simple-query partition $\mathcal{Q}$ of Q satisfying that $\mathsf{hd}(q) \subseteq \mathsf{hd}(Q)$ for each $q \in \mathcal{Q}$ can be obtained in time $O(|Q|)$.*

Proof. By the above two steps, this theorem holds trivially. □

5.3.2. Partitioning and evaluating non-simple query reducible queries

As shown in Example 9, if a conjunctive query is not simple-query reducible, then some certain answers of this query over a satisfiable DL-Lite$_\mathcal{A}$ KB cannot be obtained by evaluating any simple-query partition of this query over any SCSQA-local partition of this KB. Thus, for completeness, non-simple query partitions need to be evaluated.

By the semantics of conjunctive queries, one can observe that if a certain answer of a conjunctive query over a DL-Lite$_\mathcal{A}$ KB is obtained by binding some non-distinguished variables to anonymous elements, then in the scenario of KB and query partitioning, this certain answer can solely be obtained by evaluating a partition

of this query in which the sub-queries do not contain those non-distinguished variables as distinguished variables. Based on this observation, for a given set of non-distinguished variables of a conjunctive query, the following definition illustrates a way of computing a partition of this query which can capture all the certain answers of this query obtained by binding these non-distinguished variables to anonymous elements.

Definition 10. For a conjunctive query Q and set NV consisting of some non-distinguished variables of Q, we use $Q|_{SQ}^{NV}$ to denote a partition of Q satisfying the following two conditions:

1. Each variable in NV solely occurs in one query in $Q|_{SQ}^{NV}$;
2. For each non-simple query $q \in Q|_{SQ}^{NV}$ and for every two different atoms α and α' in q, there exists a sequence $\beta_1, \ldots, \beta_n$ of atoms in q such that $\beta_1 = \alpha$, $\beta_n = \alpha'$ and β_i and β_{i+1} share a variable in NV for $1 \leq i \leq n-1$.

The first restriction in Definition 10 indicates that all the variables in NV are still used as non-distinguished variables in the sub-queries in $Q|_{SQ}^{NV}$, and the second condition illustrates that if a sub-query in $Q|_{SQ}^{NV}$ is not a simple-query then its query atoms must be connected by the variables in NV. In the scenario of KB partitioning, the second condition is crucial in guaranteeing that $Q|_{SQ}^{NV}$ can capture all the certain answers of Q obtained by binding all the variables in NV to anonymous elements.

The correctness of Definition 10 can be guaranteed by the theorem below which indicates that the certain answers of Q obtained by binding all the variables in NV to anonymous element can be captured by $Q|_{SQ}^{NV}$ and by enumerating the situations of binding different non-distinguished variables to anonymous elements, all the certain answers of Q can be obtained.

Theorem 8. *Given a non-simple query reducible conjunctive query Q, then for any satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and any SCSQA-local partition $\mathcal{S}$ of $\mathcal{K}$, we can get that:*

$$\mathsf{ans}(Q, \mathcal{K}) = \cup_{NV' \in 2^{NV}} \mathsf{ans}(Q, Q|_{SQ}^{NV'}, \mathcal{S})$$

where NV is the set consisting of all the non-distinguished variables of Q.

Proof. By Lemma 5, the ($\supseteq$) direction of the equation in this theorem holds trivially. Next, we show the ($\subseteq$) direction. Let $\vec{u} \in \mathsf{ans}(Q, \mathcal{K})$. Suppose $\mathcal{S} = \cup_{i=1}^{n}\{\mathcal{K}_i\}$. $\mathcal{K}$ is satisfiable. So all the KBs in $\mathcal{S}$ are satisfiable. For each $\mathcal{K}_i$, let $\mathcal{I}_i$ be a canonical model of $\mathcal{K}_i$. Assume (A) $\mathcal{I}_i$ and $\mathcal{I}_j$ do not share any anonymous element, i.e., $(\Delta^{\mathcal{I}_i} - \boldsymbol{I}) \cap (\Delta^{\mathcal{I}_j} - \boldsymbol{I}) = \emptyset$, for $i \neq j$. Based on $\mathcal{I}_1 - \mathcal{I}_n$, a model $\mathcal{I}$ of $\mathcal{K}$ can be constructed by setting $\Delta^{\mathcal{I}} = \cup_{i=1}^{n}\Delta^{\mathcal{I}_i}$, $a^{\mathcal{I}} = a$ for each $a \in \boldsymbol{I}$ and $r^{\mathcal{I}} = \cup_{i=1}^{n} r^{\mathcal{I}_i}$ for each $r \in \boldsymbol{C} \cup \boldsymbol{R}$. Then there exists a binding π of $Q(\vec{u})$ over $\mathcal{I}$ such that $\mathcal{I} \models Q(\vec{u})$ holds. Let $NV' = \{x | x \in NV \wedge \pi(x) \notin \boldsymbol{I}\}$, i.e., NV' is the set of all the non-distinguished variables that are bound to anonymous elements. Next, we show (C) $\vec{u} \in \mathsf{ans}(Q, Q|_{SQ}^{NV'}, \mathcal{S})$. Suppose $Q|_{SQ}^{NV'} = \cup_{k=1}^{m}\{q_k\}$. Let $\vec{u}_k$ be a tuple such that $|\vec{u}_k| = |\mathsf{hd}(q_k)|$ and for each $1 \leq l \leq |\vec{u}_k|$, if $\mathsf{hd}(q_k)[l]$ occurs in the jth position of $\mathsf{hd}(Q)$ then $\vec{u}_k[l] = \vec{u}[j]$ otherwise $\vec{u}_k[l] = \pi(\mathsf{hd}(q_k)[l])$. By the construction of these tuples, we can get that $\vec{u}_1 - \vec{u}_m$ are tuples of names and (E) $\vec{u} = \mathsf{hd}(Q)[\mathsf{hd}(q_1)/\vec{u}_1, \ldots, \mathsf{hd}(q_m)/\vec{u}_m]$ holds. For each $\vec{u}_k$, next we show (C1) $\vec{u}_k \in \cup_{i=1}^{n}\mathsf{ans}(q_k, \mathcal{K}_i)$ holds. $\vec{u}_k \in \mathsf{ans}(q_k, \mathcal{K})$ holds trivially. If q_k is a simple-query then (C1) holds, since $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$. Otherwise, let π_k be a function that maps each individual a in $q_k(\vec{u}_k)$ to a and each variable x in $q_k(\vec{u}_k)$ to $\pi(x)$. Then by the second condition of Definition 10 and the assumption (A), we can get that there exists $\mathcal{I}_i$ such that $\mathcal{I}_i, \pi_k \models q_k(\vec{u}_k)$ holds. So $\vec{u}_k \in \mathsf{ans}(q_k, \mathcal{K}_i)$ holds. Thus (C1) holds. Then by (E), $\vec{u} \in \mathsf{ans}(Q, Q|_{SQ}^{NV'}, \mathcal{S})$ holds. Hence the ($\subseteq$) direction of the equation in this theorem holds. Therefore this theorem holds. □

For a non-simple query reducible conjunctive query Q and set NV of some non-distinguished variables of Q, a partition $\mathcal{Q}$ of Q satisfying the two conditions in Definition 10, i.e., the partition $Q|_{SQ}^{NV}$, can be obtained by the following two steps:

1. Compute a partition U of $\mathsf{bd}(Q)$ by the following two steps. Let $\mathsf{A} = \mathsf{bd}(Q)$:
   a. Compute an undirected graph G with node set A. For every two atoms α and α' in A, there exists an edge between α and α' iff they share a variable in NV.
   b. For each connected component l of G, generate an atom set U consisting of all the nodes in l and set $\mathsf{U} = \mathsf{U} \cup \{\mathsf{U}\}$ and $\mathsf{A} = \mathsf{A} - \mathsf{U}$.
   c. For each variable or individual x occurring in A, generate an atom set U consisting of all the atoms in A containing x and set $\mathsf{U} = \mathsf{U} \cup \{\mathsf{U}\}$ and $\mathsf{A} = \mathsf{A} - \mathsf{U}$.
2. For each atom set $\mathsf{U} \in \mathsf{U}$, generate a sub-query q such that $\mathsf{bd}(q) = \mathsf{U}$ and $\mathsf{hd}(q)$ consisting of all the variables in U that also occur either in $\mathsf{hd}(Q)$ or in $\mathsf{bd}(Q) - \mathsf{U}$, and add q to $\mathcal{Q}$.

Based on the above two steps, we can further obtain the complexity of computing the partition $Q|_{SQ}^{NV}$.

Lemma 7. *For a conjunctive query Q and set NV of some non-distinguished variables of Q, $Q|_{SQ}^{NV}$ can be computed in time $O(|Q|^2)$.*

Proof. According to the concrete way of computing $Q|_{SQ}^{NV}$, this lemma holds trivially. □

Example 12. Consider the conjunctive query Q in Example 9 again. Q has two non-distinguished variables $?x$ and $?y$. By enumerating the subsets of $\{?x, ?y\}$, four partitions of Q can be computed:

$$Q|_{SQ}^{\{?x\}} = \{A(?x) \wedge P(?x, ?y) \rightarrow q(?y),\ B(?y) \rightarrow q(?y)\}$$
$$Q|_{SQ}^{\{?y\}} = \{A(?x) \rightarrow q(?x),\ P(?x, ?y) \wedge B(?y) \rightarrow q(?x)\}$$
$$Q|_{SQ}^{\{?x,?y\}} = \{A(?x) \wedge P(?x, ?y) \wedge B(?y) \rightarrow q()\}$$
$$Q|_{SQ}^{\{\}} = Q|_{SQ}^{\{?x\}} \text{ or } Q|_{SQ}^{\{\}} = Q|_{SQ}^{\{?y\}}$$

By Theorem 8, all the certain answers of Q over any satisfiable DL-Lite$_\mathcal{A}$ KB can be obtained by evaluating the above four partitions of Q over any SCSQA-local partition of this KB.

Finally, note that for a non-simple query reducible query with n non-distinguished variables, Theorem 8 indicates that as many as 2^n query partitions need to be answered over a SCSQA-local partition of a DL-Lite$_\mathcal{A}$ KB to obtain all the certain answers of this query over the KB. Therefore, compared with simple-query reducible queries, more time may be spent in evaluating non-simple query reducible queries.

6. Optimizing the procedure of evaluating query partitions

In this section, we present the last problem P3, i.e., optimizing the procedure of evaluating query partitions over KB partitions.

Although query partitioning can reduce the final number of queries needed to be evaluated over the ABox of a DL-Lite$_\mathcal{A}$ KB significantly,[5] a large amount of intermediate results generated by

[5] For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a conjunctive query Q, the classical way of answering Q over $\mathcal{K}$ is by rewriting, i.e., first rewrites Q into a group of conjunctive queries according to the axioms in $\mathcal{T}$ and then evaluates the rewritten conjunctive queries over $\mathcal{A}$ directly where reasoning is no longer needed. Suppose that we partition Q into sub-queries $q_1, \ldots, q_m$ and q_i has n_i rewritten queries for $1 \leq i \leq m$. The number of conjunctive queries eventually needed to be evaluated over $\mathcal{A}$ is $n_1 + \cdots + n_m$; whereas without query partitioning, answering Q over $\mathcal{K}$ needs to evaluate no less than $n_1 \times \cdots \times n_m$ queries over $\mathcal{A}$. This says that query partitioning can reduce the final number of conjunctive queries answered over the ABoxes of KBs.

evaluating partitioned sub-queries may also slow down the overall query answering procedure. According to the work of Sideway Information Passing [36–38], we realize an optimization strategy specific for DL-Lite$_\mathcal{A}$ to reduce the intermediate results as many and as early as possible by transferring values between sub-queries. As illustrated in the example below, when answering a sub-query in a partition of a conjunctive query, we will transfer the corresponding values obtained from the certain answers of the sub-queries that have already been evaluated to this sub-query before it is evaluated, i.e., materializing some variables of this sub-query when their values are known. By this way, we can avoid computing the certain answers of this sub-query which do not coincide with the previous sub-queries.

Example 13. Consider the following conjunctive query Q and its partition $\mathcal{Q}$:

$$Q:\ R(a, ?x) \wedge P(?x, ?y) \wedge S(?y, ?z) \rightarrow q(?x, ?y, ?z)$$

$$\begin{aligned}\mathcal{Q} \quad &= \{q_1\colon R(a, ?x) \wedge P(?x, ?y) \rightarrow q(?x, ?y),\\ &\qquad q_2\colon S(?y, ?z) \rightarrow q(?y, ?z)\}\end{aligned}$$

Let $\mathcal{S}$ be a partition of a DL-Lite$_\mathcal{A}$ KB. After evaluating q_1 over the KBs in $\mathcal{S}$, suppose we obtain the certain answer set $\Lambda = \cup_{i=1}^{3}\{(a_i, b_i)\}$. If we transfer the bindings of $?y$ to q_2 before it is evaluated, i.e., replacing evaluating q_2 with answering the following three conjunctive queries:

$$\begin{aligned}q_2^{b_1}:&\quad S(b_1, ?z) \rightarrow q(b_1, ?z)\\ q_2^{b_2}:&\quad S(b_2, ?z) \rightarrow q(b_2, ?z)\\ q_2^{b_3}:&\quad S(b_3, ?z) \rightarrow q(b_3, ?z)\end{aligned}$$

then we can avoid to computing the certain answers of q_2 which do not bind $?y$ to b_1, b_2 or b_3.

For a function ξ, we use $\mathsf{dm}(\xi)$ and $\mathsf{rg}(\xi)$ to denote the domain and range of ξ, respectively. For a substitution $[\vec{x}/\vec{u}]$ and tuple $\vec{y}$, we use $[\vec{x}/\vec{u}]|_{\vec{y}}$ to denote a function ξ such that $\mathsf{dm}(\xi)$ consists of all the elements occurring both in $\vec{x}$ and $\vec{y}$ and for each $x \in \mathsf{dm}(\xi)$, if x occurs in the ith position of $\vec{x}$ then $\xi(x) = \vec{u}[i]$. The formalization as well as the correctness of the sub-query value transfer procedure in the situation of DL-Lite$_\mathcal{A}$ and conjunctive query partitioning is illustrated in the proposition below.

Proposition 2. *For a partition $\mathcal{Q} = \cup_{i=1}^{n}\{q_i\}$ of a conjunctive query Q and a partition $\mathcal{S}$ of a satisfiable DL-Lite$_\mathcal{A}$ KB, the following equation holds:*

$$\begin{aligned}&\mathsf{ans}(Q, \mathcal{Q}, \mathcal{S}) =\\ &\{\vec{x}[\vec{x}_1/\vec{u}_1, \ldots, \vec{x}_n/\vec{u}_n]|(\vec{u}_1, \ldots, \vec{u}_n) \in \Lambda_1|_{\mathcal{B}_1} \times \cdots \times \Lambda_n|_{\mathcal{B}_n}\}\end{aligned}$$

where $\vec{x} = \mathsf{hd}(Q)$, $\vec{x}_i = \mathsf{hd}(q_i)$ for $1 \le i \le n$, and

$$\mathcal{B}_k = \begin{cases}\emptyset & \text{if } k = 1;\\ \{[\vec{x}_1/\vec{u}_1, \ldots, \vec{x}_{k-1}/\vec{u}_{k-1}]|_{\vec{x}_k}| & \\ \quad (\vec{u}_1, \ldots, \vec{u}_{k-1}) \in \Lambda_1|_{\mathcal{B}_1} \times \cdots \times \Lambda_{k-1}|_{\mathcal{B}_{k-1}}\} & \text{if } k > 1.\end{cases}$$

$$\Lambda_k|_{\mathcal{B}_k} = \begin{cases}\cup_{\mathcal{K}\in\mathcal{S}}\mathsf{ans}(q_1, \mathcal{K}) & \text{if } k = 1;\\ \cup_{\mathcal{K}\in\mathcal{S}}\cup_{\xi\in\mathcal{B}_k}\mathsf{ans}(q_k\xi, \mathcal{K}) & \text{if } k > 1.\end{cases}$$

Proof. For a conjunctive query q and satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$, the following two conclusions (C1) and (C2) can be trivially proved. (C1) For each $\vec{u} \in \mathsf{ans}(q, \mathcal{K})$ and function ξ such that $\mathsf{dm}(\xi)$ consists of some variables in $\mathsf{hd}(q)$ and for each $x \in \mathsf{dm}(\xi)$ if x occurs in the ith position of $\mathsf{hd}(q)$ then $\xi(x) = \vec{u}[i]$, then $\vec{u} \in \mathsf{ans}(q\xi, \mathcal{K})$ holds. (C2) for each function ξ such that $\mathsf{dm}(\xi)$ consists of some variables in $\mathsf{hd}(q)$ and $\mathsf{rg}(\xi) \subseteq \boldsymbol{I}$, then $\mathsf{ans}(q\xi, \mathcal{K}) \subseteq \mathsf{ans}(q, \mathcal{K})$ holds. Let $\vec{u} \in \mathsf{ans}(Q, \mathcal{Q}, \mathcal{S})$. Then there exists $\vec{u}_i \in \cup_{\mathcal{K}\in\mathcal{S}}\mathsf{ans}(q_i, \mathcal{K})$ for $q_i \in \mathcal{Q}$ such that $\vec{u} = \vec{x}[\vec{x}_1/\vec{u}_1, \ldots, \vec{x}_n/\vec{u}_n]$ holds. Let $\xi_i = [\vec{x}_1/\vec{u}_1, \ldots, \vec{x}_{i-1}/\vec{u}_{i-1}]|_{\vec{x}_i}$ for $1 < i \le n$. Then by (C1) and mathematical induction, it can be proved that $\xi_i \in \mathcal{B}_i$ and $\vec{u}_i \in \Lambda_i|_{\mathcal{B}_i}$ hold. So $\vec{u}$ is an element of the set of the right hand of the equation in this proposition. So the ($\subseteq$) direction of the equation in this proposition holds. By (C2), $\Lambda_k|_{\mathcal{B}_k} \subseteq \cup_{\mathcal{K}\in\mathcal{S}}\mathsf{ans}(q_k, \mathcal{K})$ holds for each $1 \le k \le n$. Thus the ($\supseteq$) direction of the equation in this proposition holds. Therefore this proposition holds. □

In Proposition 2, $\mathcal{B}_k$ denotes the set of bindings of some variables of q_k obtained from the certain answers of the sub-queries evaluated before q_k.

By observing the conjunctive query rewriting procedure in DL-Lite$_\mathcal{A}$, we found that the transferred variable bindings do not affect the query rewriting procedure. This means that when evaluating a sub-query q over a sub-KB $\mathcal{K}$ with a set $\mathcal{B}$ of variable bindings, the conjunctive queries obtained by first transferring the bindings in $\mathcal{B}$ to q then rewriting the resultant queries over the TBox of $\mathcal{K}$ are equivalent to the conjunctive queries obtained by first rewriting q over the TBox of $\mathcal{K}$ then transferring the bindings in $\mathcal{B}$ to the rewritten queries. Thus, we actually need one time of query rewriting rather than $|\mathcal{B}|$ times. So, the procedure of evaluating q over $\mathcal{K}$ with the variable binding set $\mathcal{B}$ can be further optimized. The formalization and correctness are illustrated in the following lemma and proposition.

Lemma 8. *For a conjunctive query q, DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$ and set $\mathcal{B}$ of functions ξ such that $\mathsf{dm}(\xi) \subseteq \mathsf{hd}(q)$ and $\mathsf{rg}(\xi) \subseteq \boldsymbol{I}$, the following equation holds:*

$$\begin{aligned}&\cup_{\xi\in\mathcal{B}}\mathtt{PerfectRef}(q\xi, \mathcal{T}) =\\ &\cup_{\xi\in\mathcal{B}}\{q'\xi|q' \in \mathtt{PerfectRef}(q, \mathcal{T})\}\end{aligned}$$

Proof. Let ξ be an arbitrary function in $\mathcal{B}$. This lemma can be proved by showing the following equation holds:

$$\mathtt{PerfectRef}(q\xi, \mathcal{T}) = \{q'\xi|q' \in \mathtt{PerfectRef}(q, \mathcal{T})\}$$

Suppose the rewriting procedure of q over $\mathcal{T}$ by PerfectRef is Pro1:

$$\mathsf{Pro1}:\quad (S_1, q_1) \xrightarrow[\alpha_1,\ \beta_1]{I_1,\ \alpha_1} I_1\, (S_2, q_2) \xrightarrow[\alpha_2,\ \beta_2]{I_2,\ \alpha_2} I_2 \cdots \xrightarrow[\alpha_n,\ \beta_n]{I_n,\ \alpha_n} I_n\, (S_{n+1})$$

where $S_1 = \{q\}$, $q_1 = q$, $q_i \in S_i$, α_i and β_i are query atoms of q_i, I_i is a inclusion axiom in $\mathcal{T}$, $l_i \in \{0, 1\}$, and

$$S_{i+1} = \begin{cases}S_i \cup \{q_i[\alpha_i/\mathsf{gr}(\alpha_i, I_i)]\} & \text{if } l_i = 0;\\ S_i \cup \{\tau(reduce(q_i, \alpha_i, \beta_i))\} & \text{if } l_i = 1.\end{cases}$$

Here, $q_i[\alpha_i/\mathsf{gr}(\alpha_i, I_i)]$ denotes the query obtained by applying the inclusion axiom I_i over q_i based on the atom α_i, and $\tau(reduce(q_i, \alpha_i, \beta_i))$ denotes the query obtained by unifying the two query atoms α_i and β_i into one query atom. Next, we show the following procedure Pro2 is a rewriting procedure of $q\xi$:

$$\mathsf{Pro2}:\quad (S'_1, q'_1) \xrightarrow[\alpha'_1,\ \beta'_1]{I_1,\ \alpha'_1} I_1\, (S'_2, q'_2) \xrightarrow[\alpha'_2,\ \beta'_2]{I_2,\ \alpha'_2} I_1 \cdots \xrightarrow[\alpha'_n,\ \beta'_n]{I_n,\ \alpha'_n} I_1\, (S'_{n+1})$$

where $S'_i = \{q\xi|q \in S_i\}$, $q'_i = q_i\xi$, $\alpha'_i = \alpha_i\xi$ and $\beta_i = \beta_i\xi$. This can be proved by showing the following relation (R) holds:

$$S'_{i+1} = \begin{cases}S'_i \cup \{q'_i[\alpha_i/\mathsf{gr}(\alpha'_i, I_i)]\} & \text{if } l_i = 0;\\ S'_i \cup \{\tau(reduce(q'_i, \alpha'_i, \beta'_i))\} & \text{if } l_i = 1.\end{cases}$$

R can be trivially validated by mathematical induction. Thus this lemma holds. □

Based on Lemma 8, we can get that evaluating a sub-query over a sub-KB with a set of variable bindings can be realized by first rewriting this sub-query over the TBox of this sub-KB, transferring the variable bindings to the rewritten queries and evaluating the finally obtained queries over the ABox of this sub-KB.

Proposition 3. *For a conjunctive query Q, satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and set $\mathcal{B}$ of functions ξ such that $\mathsf{dm}(\xi) \subseteq \mathsf{hd}(q)$ and $\mathsf{rm}(\xi) \subseteq \boldsymbol{I}$, the following equation holds:*

$$\cup_{\xi\in\mathcal{B}}\mathsf{ans}(q\xi, \mathcal{K}) = \cup_{q'\in\mathsf{PerfectRef}(q,\mathcal{T})}\cup_{\xi\in\mathcal{B}}\mathsf{ans}(q'\xi, (\emptyset, \mathcal{A}))$$

Proof. By the query rewriting feature of DL-Lite$_\mathcal{A}$, we can get:

$$\cup_{\xi\in\mathcal{B}}\mathsf{ans}(q\xi, \mathcal{K}) = \cup_{\xi\in\mathcal{B}}\cup_{q'\in\mathsf{PerfectRef}(q\xi,\mathcal{T})}\mathsf{ans}(q', (\emptyset, \mathcal{A}))$$

Then by Lemma 8, this proposition holds directly. □

Example 14. Consider the queries and KBs in Example 13 again. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a KB in the partition $\mathcal{S}$. Then evaluating the sub-query q_2 over $\mathcal{K}$ with the three variable bindings ξ_1: $?y \to b_1$, ξ_2: $?y \to b_2$ and ξ_3: $?y \to b_3$ can be realized by the following three steps. Firstly, rewrite q_2 over $\mathcal{T}$. After this step, suppose we obtain the following two queries:

q_2^1 : $S_1(?y, ?z) \to q(?y, ?z)$
q_2^2 : $S_2(?z, ?y) \to q(?y, ?z)$

Secondly, transfer the bindings $\xi_1 - \xi_3$ to q_2^1 and q_2^2. After this step, the following queries can be computed:

$q_2^{1_1}$: $S_1(b_1, ?z) \to q(b_1, ?z)$
$q_2^{1_2}$: $S_1(b_2, ?z) \to q(b_1, ?z)$
$q_2^{1_3}$: $S_1(b_3, ?z) \to q(b_1, ?z)$
$q_2^{2_1}$: $S_2(?z, b_1) \to q(b_1, ?z)$
$q_2^{2_2}$: $S_2(?z, b_2) \to q(b_2, ?z)$
$q_2^{2_3}$: $S_2(?z, b_3) \to q(b_3, ?z)$

Finally, evaluating the above six conjunctive queries over $\mathcal{A}$ can obtain all the certain answers of evaluating these three queries $q_2\xi_1$, $q_2\xi_2$ and $q_2\xi_3$ over $\mathcal{K}$.

The result presented in Proposition 3 can be used to optimize the procedure of computing the components $\cup_{\mathcal{K}\in\mathcal{S}}\cup_{\xi\in\mathcal{B}_{i+1}}\mathsf{ans}(q_{i+1}\xi, \mathcal{K})$ in Proposition 2.

Finally, note that for the value transferring strategy to work, the order of evaluating the sub-queries in a partition of a conjunctive query is crucial. In Example 13, evaluating $\mathcal{Q}$ in the order of q_1, q_2 would be much more efficient than in the order of q_2, q_1, since evaluating q_2 is more time costly with a large number of certain answers obtained. For two sub-queries q and q' in a partition of a conjunctive query, if q and q' share some variables and q contains more individuals or has more query atoms, then q should be evaluated before q', as evaluating q may return fewer certain answers. The experiments presented in the subsequent section demonstrate that this value transferring strategy can significantly improve the query answering performance. Nevertheless, there are side-effects as transferring values between sub-queries in a partition destroys their independence. Moreover, as illustrated by the results of evaluating NQ_1^{dbp} in the experiments, if all the sub-queries in a partition can be evaluated efficiently and have less number of certain answers, this strategy may slow down the procedure of evaluating this query partition.

7. Experiments

In this section, we conduct experiments to demonstrate the efficiency of the provided approach as well as the rationality of our proposal of using DL-Lite$_\mathcal{A}$ techniques to consume the Web-scale Open Data. For this purpose, we have implemented a prototype system[6] in Java, where classical algorithms `Consistent` and `PerfectRef` in [24] are adopted for checking the satisfiability of DL-Lite$_\mathcal{A}$ KBs and rewriting conjunctive queries over DL-Lite$_\mathcal{A}$ TBoxes, respectively. Moreover, MySQL version 5.5 is used for storing the ABoxes of DL-Lite$_\mathcal{A}$ KBs. Concretely, for each ABox $\mathcal{A}$, a database $\mathcal{DB}$ is automatically created consisting of an unary table for every class in $\mathcal{A}$ and a binary table for every role in $\mathcal{A}$ to store their individual assertions. Furthermore, we adopt multithread techniques to check the satisfiability of and answer a single query over the sub-KBs in a KB partition in a parallel way. Note that the concrete choices of the specific reasoning and query answering algorithms and database engines as well as the concrete ways of managing the ABoxes of DL-Lite$_\mathcal{A}$ KBs in relational databases can affect the actual performance of both satisfiability checking and conjunctive query answering. Neverthless, running a comparison between different choices is unnecessary for the purpose of this paper, as our main goal is to detect the performance difference between reasoning and query answering with and without KB and query partitioning, as well as to reveal the benefit of KB and query partitioning for large-scale, performance-critical applications.

We ran the prototype system on two Web-scale, real-world open datasets described by Semantic Web standards, DBpedia_200 dataset[7] extended with DBpedia ontology[8] (totally contains 0.278 billion RDF triples) and BTC 2012 dataset (totally contains 1.1 billion RDF triples). The machine used was a server with 64-bit Windows 2010 operating system, Intel (R) Xeon (R) CPU E5-2640 v4 2.40 GHz, 64 RAM, 20 kernels and 40 logical processors. The Java VM used was 1.6.0.22 where the main memory was limited to 50G.

7.1. The DL-Lite$_\mathcal{A}$ KB construction and SCSQA-local partition computation

In this subsection we report the results of constructing DL-Lite$_\mathcal{A}$ KBs from the two datasets and computing SCSQA-local partitions of the obtained DL-Lite$_\mathcal{A}$ KBs.

DL-Lite$_\mathcal{A}$ adopts unique name assumption, i.e., every two different individuals are unequal. Therefore, we materialize the name equivalence relations (the assertions of the OWL vocabulary `owl:sameAs`) in the two datasets by first grouping the names based on their equivalence relationship and then for each group, choosing one surrogate and replacing all the equivalent names with the surrogate. The schema knowledge described in the DBpedia_200 dataset does not exceed the expressivity of DL-Lite$_\mathcal{A}$. After materializing the name equivalence relations, a DL-Lite$_\mathcal{A}$ KB $\mathcal{K}_{dbp}$ with 233,227 axioms and 275,710,447 individual assertions is constructed from the DBpedia_200 dataset. For the BTC 2012 dataset, by dropping 5,259 axioms that cannot be captured by DL-Lite$_\mathcal{A}$, a DL-Lite$_\mathcal{A}$ KB $\mathcal{K}_{btc}$ with 339,519 axioms and 1040,455,793 individual assertions is constructed. We randomly choose two numbers, 28,571,044 and 52,022,789, to respectively limit the size of the ABoxes of the sub-KBs for $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$. As a result, as listed in Table 2, a SCSQA-local partition $\mathcal{S}_{dbp}$ of $\mathcal{K}_{dbp}$ with 14 sub-KBs and a SCSQA-local partition $\mathcal{S}_{btc}$ of $\mathcal{K}_{btc}$ with 30 sub-KBs have been computed using the concrete ways described in Section 4.3, within 1.7 and 4.5 hours, respectively. One can see that the provided approach for computing SCSQA-local partitions of DL-Lite$_\mathcal{A}$ KBs has generated balanced-sized sub-KBs.

[6] Source code can be found at https://github.com/Lucy321456/DLLitePartition.
[7] http://benchmark.dbpedia.org/.
[8] http://wiki.dbpedia.org/downloads-2016-04.

Table 2
Statistics of the sub-KBs in $\mathcal{S}_{dbp}$ (a) and in $\mathcal{S}_{btc}$ (b), where ‖ denotes the size of set and M denotes million.

(a)

KB	$\mathcal{K}_1^{dbp}$	$\mathcal{K}_2^{dbp}$	$\mathcal{K}_3^{dbp}$	$\mathcal{K}_4^{dbp}$	$\mathcal{K}_5^{dbp}$	$\mathcal{K}_6^{dbp}$	$\mathcal{K}_7^{dbp}$
|TBox|	7440	41,408	54,806	62,625	65,486	67,937	68,853
|ABox|	28.57M	28.57M	28.57M	28.57M	2.85M	28.57M	28.57M
KB	$\mathcal{K}_8^{dbp}$	$\mathcal{K}_9^{dbp}$	$\mathcal{K}_{10}^{dbp}$	$\mathcal{K}_{11}^{dbp}$	$\mathcal{K}_{12}^{dbp}$	$\mathcal{K}_{13}^{dbp}$	$\mathcal{K}_{14}^{dbp}$
|TBox|	79,946	95,469	77,925	30,061	57,637	49,523	59,622
|ABox|	28.57M	28.57M	28.57M	28.57M	28.57M	28.57M	24.69M

(b)

KB	$\mathcal{K}_1^{btc}$	$\mathcal{K}_2^{btc}$	$\mathcal{K}_3^{btc}$	$\mathcal{K}_4^{btc}$	$\mathcal{K}_5^{btc}$	$\mathcal{K}_6^{btc}$	$\mathcal{K}_7^{btc}$	$\mathcal{K}_8^{btc}$	$\mathcal{K}_9^{btc}$	$\mathcal{K}_{10}^{btc}$
|TBox|	1573	5587	5137	3248	64	6143	6728	6585	6864	7894
|ABox|	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M
KB	$\mathcal{K}_{11}^{btc}$	$\mathcal{K}_{12}^{btc}$	$\mathcal{K}_{13}^{btc}$	$\mathcal{K}_{14}^{btc}$	$\mathcal{K}_{15}^{btc}$	$\mathcal{K}_{16}^{btc}$	$\mathcal{K}_{17}^{btc}$	$\mathcal{K}_{18}^{btc}$	$\mathcal{K}_{19}^{btc}$	$\mathcal{K}_{20}^{btc}$
|TBox|	6346	7354	526	526	456	240	173	17	17	4893
|ABox|	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M
KB	$\mathcal{K}_{21}^{btc}$	$\mathcal{K}_{22}^{btc}$	$\mathcal{K}_{23}^{btc}$	$\mathcal{K}_{24}^{btc}$	$\mathcal{K}_{25}^{btc}$	$\mathcal{K}_{26}^{btc}$	$\mathcal{K}_{27}^{btc}$	$\mathcal{K}_{28}^{btc}$	$\mathcal{K}_{29}^{btc}$	$\mathcal{K}_{30}^{btc}$
|TBox|	513	5101	4922	5474	4794	749	140	108	7291	2662
|ABox|	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	52.02M	5.58M

Table 3
Results of satisfiability checking, where `Time` denotes minutes used to check satisfiability, `False` and `True` respectively denote non-satisfiable and satisfiable, ++ denotes out of memory error, and `Query` denotes the number of queries needed to be answered over the ABox of the corresponding KB for satisfiability checking.

(a)

	$\mathcal{K}_{dbp}$	$\mathcal{S}_{dbp}$
Time	6.44	**0.07**
Result	False	False

	$\mathcal{K}_{btc}$	$\mathcal{S}_{btc}$
Time	++	**0.15**
Result	++	False

(b)

KB	$\mathcal{K}_{dbp}$
Query	**2,963,132**

SubKB	$\mathcal{K}_1^{dbp}$	$\mathcal{K}_2^{dbp}$	$\mathcal{K}_3^{dbp}$	$\mathcal{K}_4^{dbp}$	$\mathcal{K}_5^{dbp}$	$\mathcal{K}_6^{dbp}$	$\mathcal{K}_7^{dbp}$
Query	219,683	**402,145**	398,597	384,536	374,923	378,539	340,026

KB	$\mathcal{K}_{btc}$
Query	++

SubKB	$\mathcal{K}_8^{dbp}$	$\mathcal{K}_9^{dbp}$	$\mathcal{K}_{10}^{dbp}$	$\mathcal{K}_{11}^{dbp}$	$\mathcal{K}_{12}^{dbp}$	$\mathcal{K}_{13}^{dbp}$	$\mathcal{K}_{14}^{dbp}$
Query	318,287	305,123	230,454	110,242	191,799	121,188	117,887

SubKB	$\mathcal{K}_1^{btc}$	$\mathcal{K}_2^{btc}$	$\mathcal{K}_3^{btc}$	$\mathcal{K}_4^{btc}$	$\mathcal{K}_5^{btc}$	$\mathcal{K}_6^{btc}$	$\mathcal{K}_7^{btc}$	$\mathcal{K}_8^{btc}$	$\mathcal{K}_9^{btc}$	$\mathcal{K}_{10}^{btc}$
Query	10,247	159,466	120,526	28,194	96	190,437	246,762	220,563	257,384	253,381

SubKB	$\mathcal{K}_{11}^{btc}$	$\mathcal{K}_{12}^{btc}$	$\mathcal{K}_{13}^{btc}$	$\mathcal{K}_{14}^{btc}$	$\mathcal{K}_{15}^{btc}$	$\mathcal{K}_{16}^{btc}$	$\mathcal{K}_{17}^{btc}$	$\mathcal{K}_{18}^{btc}$	$\mathcal{K}_{19}^{btc}$	$\mathcal{K}_{20}^{btc}$
Query	197,857	253,784	2,298	2,276	1,833	853	455	0	0	127,928

SubKB	$\mathcal{K}_{21}^{btc}$	$\mathcal{K}_{22}^{btc}$	$\mathcal{K}_{23}^{btc}$	$\mathcal{K}_{24}^{btc}$	$\mathcal{K}_{25}^{btc}$	$\mathcal{K}_{26}^{btc}$	$\mathcal{K}_{27}^{btc}$	$\mathcal{K}_{28}^{btc}$	$\mathcal{K}_{29}^{btc}$	$\mathcal{K}_{30}^{btc}$
Query	4,050	129,035	132,648	154,402	167,010	5,819	124	122	**309,665**	25,976

7.2. The performance of satisfiability checking

In this subsection we report the effect of KB partitioning on the performance of satisfiability checking. The results of checking the satisfiability of the KBs in the partition $\mathcal{S}_{dbp}$ ($\mathcal{S}_{btc}$) with parallelization are compared with the KB $\mathcal{K}_{dbp}$ ($\mathcal{K}_{btc}$) without partitioning in Table 3.

As shown by Table 3(a), without KB partitioning, checking the satisfiability of $\mathcal{K}_{dbp}$ totally cost 6 minutes, and for $\mathcal{K}_{btc}$, after 6 hours an out-of-memory error was reported. That $\mathcal{K}_{btc}$ contains a large number of disjoint axioms and inclusion axioms causes the failure in completing generating the queries needed to be evaluated over its ABox. Conversely, facilitated by KB partitioning which eventually enables parallelization, satisfiability checking of $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ has been successfully completed in **0.07** and **0.15** minutes, respectively. Whenever one sub-KB in a SCSQA-local partition is identified to be unsatisfiable, one can conclude that the original KB is unsatisfiable, thus the procedure can be terminated. For these two KBs, the performance improvement owes much to the smaller size of the sub-KBs in the KB partitions. For satisfiability checking, smaller TBoxes make less number of conjunctive queries needed to be answered over the ABoxes and thus less time is used to compute these queries. This can be clearly seen from Table 3(b). Without KB partitioning, checking the satisfiability of $\mathcal{K}_{dbp}$ may need to evaluate **2963,132** queries over its ABox. On the other hand, for the sub-KBs in $\mathcal{S}_{dbp}$, no more than **402,145** of these **2963,132** queries need to be evaluated over their ABoxes.

Note that Table 3(b) also indicates that if $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ are satisfiable, the performance improved by KB partitioning equipped with parallelization will be much more remarkable. If $\mathcal{K}_{dbp}$ is satisfiable then all the **2963,132** queries need to be evaluated over the ABox of $\mathcal{K}_{dbp}$. However, for each sub-KB in $\mathcal{S}_{dbp}$, there are still no more than **402,145** of the **2963,132** queries needed to be answered over the ABox of this sub-KB. Moreover, the big difference between the numbers of queries needed to be answered over the ABoxes implies that even using other database engines, similar results should be obtained.

Overall, the results in Table 3 indicate that KB partitioning equipped with parallelization can significantly improve the performance of checking the satisfiability of Web-scale DL-Lite$_{\mathcal{A}}$ KBs. When considering the two datasets as RDF/RDFS graphs, the results we obtained about their satisfiability cannot be discovered, showing the power of using DL-Lite$_{\mathcal{A}}$ techniques to consume large-scale open data.

Table 4
Tested simple-queries and non simple-queries over DBpedia_200 dataset and BTC 2012 dataset. Prefixes for the abbreviated names used in these queries are available in the Appendix of this paper.

SQ_1^{dbp}	$dbo:Person(?x) \rightarrow q(?x)$	SQ_1^{btc}	$foaf:Person(?x) \rightarrow q(?x)$
SQ_2^{dbp}	$rdfs:label(?x,'Alan\ Turing@en') \rightarrow q(?x)$	SQ_2^{btc}	$rdfs:label(?x,'Tim\ Berners-Lee') \rightarrow q(?x)$
SQ_3^{dbp}	$dbo:Person(?x) \wedge rdfs:label(?x,'Alan\ Turing@en') \rightarrow q(?x)$	SQ_3^{btc}	$foaf:Person(?x) \wedge rdfs:label(?x,'Tim\ Berners-Lee') \rightarrow q(?x)$
SQ_4^{dbp}	$skos:subject(dbr:Alan\_Turing,?x) \wedge skos:broader(?x,?y) \wedge rdfs:label(?x,?z) \rightarrow q(?x,?y,?z)$	SQ_4^{btc}	$skos\#subject(fb:01z0hl,?x) \wedge skos\#broader(?x,?y) \wedge rdfs:comment(?x,?z) \rightarrow q(?x,?y,?z)$
SQ_5^{dbp}	$dbp:workInstitutions(dbr:Alan\_Turing,?x) \wedge dbo:Organisation(?x) \wedge rdfs:comment(?x,?y) \rightarrow q(?x,?y)$	SQ_5^{btc}	$swrc:affiliation(sc:tim-berners-lee,?x) \wedge dbo:Organisation(?x) \wedge rdfs:label(?x,?y) \rightarrow q(?x,?y)$
SQ_6^{dbp}	$skos:subject(?x,dbr:Category:Computer\_pioneers) \wedge foaf:name(?x,?y) \wedge rdfs:comment(?x,?z) \rightarrow q(?x,?y,?z)$	SQ_6^{btc}	$skos:subject(?x,dbr:Love) \wedge dbp:name(?x,?y) \wedge rdfs:label(?x,?z) \rightarrow q(?x,?y,?z)$
SQ_7^{dbp}	$dbo:Person(?x) \wedge dbp:birthPlace(?x,dbr:China) \wedge dbo:birthDate(?x,?y) \wedge foaf:name(?x,?z) \wedge dbo:deathDate(?x,?d) \rightarrow q(?x,?y,?z,?d)$	SQ_7^{btc}	$foaf:Person(?x) \wedge dbp:prizes(?x,bdr:Nobel\_Prize\_in\_Physics) \wedge dbo:field(?x,?y) \wedge dbo:knownFor(?x,?z) \wedge dbo:almaMater(?x,?d) \rightarrow q(?x,?y,?z,?d)$
SQ_8^{dbp}	$dbo:Film(?x) \wedge dbo:writer(?x,dbr:Robert\_Thoeren) \wedge dbo:director(?x,?y) \wedge rdfs:comment(?x,?z) \wedge foaf:primaryTopic(?d,?x) \rightarrow q(?x,?y,?z,?d)$	SQ_8^{btc}	$dbo:Film(?x) \wedge dbo:writer(?x,dbr:Robert\_Thoeren) \wedge dbo:director(?x,?y) \wedge rdfs:label(?x,?z) \wedge foaf:primaryTopic(?d,?x) \rightarrow q(?x,?y,?z,?d)$
NQ_1^{dbp}	$skos:subject(dbr:Alan\_Turing,?x) \wedge skos:subject(?y,?x) \wedge dbp:knownFor(?y,?z) \wedge dbp:prizes(?y,?d) \rightarrow q(?x,?y,?z,?d)$	NQ_1^{btc}	$dbp:prizes(fb:01w7np,?x) \wedge dbp:prizes(?y,?x) \wedge rdfs:label(?y,?z) \rightarrow q(?x,?y,?z)$
NQ_2^{dbp}	$dbo:writer(?x,dbr:Robert\_Thoeren) \wedge dbo:director(?x,?y) \wedge rdfs:label(?y,?z) \rightarrow q(?x,?y,?z)$	NQ_2^{btc}	$dbo:director(?x,dbr:George\_Lucas) \wedge dbo:writer(?x,?y) \wedge rdfs:label(?y,?z) \rightarrow q(?x,?y,?z)$
NQ_3^{dbp}	$rdfs:label(?x,'Alan\ Turing@en') \wedge dbo:influencedBy(?y,?x) \wedge rdfs:label(?y,?z) \rightarrow q(?x,?y,?z)$	NQ_3^{btc}	$rdfs:label(?x,'Alan\ Turing@en') \wedge dbo:influencedBy(?y,?x) \wedge rdfs:label(?y,?z) \rightarrow q(?x,?y,?z)$
NQ_4^{dbp}	$dbo:Work(?x) \wedge dbo:writer(?x,dbr:Robert\_Thoeren) \wedge dbo:director(?x,?y) \wedge foaf:primaryTopic(?z,?x) \wedge dbo:birthDate(?y,?d) \rightarrow q(?x,?y,?z,?d)$	NQ_4^{btc}	$dbo:Work(?x) \wedge dbo:writer(?x,dbr:Robert\_Thoeren) \wedge dbo:director(?x,?y) \wedge rdfs:comment(?x,?z) \wedge foaf:primaryTopic(?d,?x) \wedge rdfs:comment(?y,?e) \rightarrow q(?x,?y,?z,?d,?e)$
NQ_5^{dbp}	$dbo:Person(?x) \wedge skos:subject(?x,dbr:Category:Computer\_pioneers) \wedge rdfs:comment(?x,?y) \wedge dbo:doctoralAdvisor(?x,?z) \wedge dbo:Scientist(?z,?d) \rightarrow q(?x,?y,?z,?d)$	NQ_5^{btc}	$foaf:Person(?x) \wedge dbp:prizes(?x,fb:0dt39) \wedge rdfs:comment(?x,?y) \wedge dbo:doctoralAdvisor(?x,?z) \wedge rdfs:label(?z,?e) \rightarrow q(?x,?y,?z,?e)$
NQ_6^{dbp}	$dbo:Organisation(?x) \wedge dbo:employer(?y,?x) \wedge dbo:Scientist(?y) \wedge dbp:knownFor(?y,dbr:Turing\_Award) \wedge odp:hasLocation(?x,?z) \wedge dbo:PopulatedPlace(?z) \rightarrow q(?x,?y,?z)$	NQ_6^{btc}	$dbo:EducationalInstitution(?x) \wedge dbo:almaMater(?y,?x) \wedge foaf:Person(?y) \wedge dbp:prizes(?y,fb:0dt39) \wedge dbo:field(?y,?z) \wedge dbo:knownFor(?y,?d) \wedge rdfs:label(?d,?e) \rightarrow q(?x,?y,?z,?d,?e)$

7.3. The performance of query answering

In this subsection we present the effect of KB partitioning, query partitioning and sub-query value transfer on the performance of conjunctive query answering. For this task, we respectively design 8 simple-queries SQ_1^{dbp}-SQ_8^{dbp} and 6 non-simple queries NQ_1^{dbp}-NQ_6^{dbp} for $\mathcal{K}_{dbp}$, and 8 simple-queries SQ_1^{btc}-SQ_8^{btc} and 6 non-simple queries NQ_1^{btc}-NQ_6^{btc} for $\mathcal{K}_{btc}$. These queries, as listed in Table 4, are designed by taking into account all the following three aspects, (1) whether they contain a small or large number of query atoms, (2) whether there will be a small or large number of rewritten queries over the original KBs, and (3) whether they can be evaluated efficiently or inefficiently over the original KBs. The query answering results are measured by `Time`, `Ans` and `RCQ`, where `Time` and `Ans` respectively denote the total seconds used for answering a query and the total number of certain answers finally obtained, and `RCQ` denotes the total number of rewritten queries. The concrete formulas of computing `RCQ` in different situations are listed in Table 5. For example, in the situation of query answering solely with query partitioning, `RCQ` denotes the sum of the number of the rewritten queries of each sub-query. Theoretically, if a KB $\mathcal{K}$ is not satisfiable, then for each query Q, $\mathsf{ans}(Q,\mathcal{K})$ is the set of all the tuples with length $|\mathsf{hd}(Q)|$ and consisting of the individuals in $\mathcal{K}$. Thus, we assume that all the KBs in $\mathcal{S}_{dbp}$ and in $\mathcal{S}_{btc}$ and KBs $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ are satisfiable, with the motivation of answering queries through rewriting.

We first show the effect of KB partitioning on the performance of simple-query answering. Evaluating a simple-query over a satisfiable DL-Lite$_\mathcal{A}$ KB can be soundly and completely realized by first answering this query over each sub-KB in a SCSQA-local partition of this KB independently and then merging the certain answers obtained. The results of evaluating the tested simple-queries over the KBs in $\mathcal{S}_{dbp}$ ($\mathcal{S}_{btc}$) with parallelization and over the KB $\mathcal{K}_{dbp}$ ($\mathcal{K}_{btc}$) without KB partitioning are shown in Table 6.

Table 5
Formulas for computing `RCQ` w.r.t. a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T},\mathcal{A})$, a partition $\cup_{i=1}^{n}\{(\mathcal{T}_i,\mathcal{A}_i)\}$ of $\mathcal{K}$, a conjunctive query Q and a partition $\cup_{i=1}^{m}\{q_i\}$ of Q.

`RCQ`	`Strategy`
$\lvert\mathsf{PerfectRef}(\mathcal{T},Q)\rvert$	Without KB partition and query partition
$\sum_{k=1}^{m}\lvert\mathsf{PerfectRef}(\mathcal{T},q_i)\rvert$	Solely with query partition
$\max_{i=1}^{n}\{\lvert\mathsf{PerfectRef}(\mathcal{T}_i,Q)\rvert\}$	Solely with KB partition
$\max_{i=1}^{n}\{\sum_{k=1}^{m}\lvert\mathsf{PerfectRef}(\mathcal{T}_i,q_k)\rvert\}$	With both KB partition and query partition

In general, the values of `Time` listed in Table 6(a) and (b) indicate that no matter a simple-query has a large or small number of rewritten queries over the original KB, such as SQ_7^{dbp}, SQ_2^{dbp}, SQ_8^{btc} and SQ_2^{btc}, or it can be evaluated efficiently or inefficiently over the original KB, such as SQ_2^{dbp}, SQ_3^{dbp}, SQ_5^{btc} and SQ_3^{btc}, in the situation of KB partitioning equipped with parallelization, this simple-query can be answered in an extremely efficient way. Analogously to satisfiability checking, such performance improvement owes much to the smaller size of the sub-KBs and parallelization enabled by partitioning. Firstly, smaller TBoxes generate smaller number of rewritten conjunctive queries and less time is thus used to compute these rewritten queries, w.r.t. a query rewriting algorithm and a conjunctive query. This can be seen by comparing the 1st row with the 2nd row of Table 6(a) and (b), respectively. Take SQ_7^{dbp} as

Table 6
Results of evaluating the tested simple-queries over the corresponding KBs with and without KB partitioning, where the better values of RCQ and Time are given in bold and where — denotes a query cannot be evaluated in 30 minutes.

(a)

		SQ_1^{dbp}	SQ_2^{dbp}	SQ_3^{dbp}	SQ_4^{dbp}	SQ_5^{dbp}	SQ_6^{dbp}	SQ_7^{dbp}	SQ_8^{dbp}
RCQ	$\mathcal{K}_{dbp}$	1,145	1	1,145	1	503	1	18,289	124
	$\mathcal{S}_{dbp}$	**378**	1	**378**	1	**253**	1	**377**	**12**
Time	$\mathcal{K}_{dbp}$	54.66	16.02	–	14.30	101.34	–	–	–
	$\mathcal{S}_{dbp}$	**29.97**	**4.08**	**201.86**	**1.40**	**45.74**	**824.03**	**30.51**	**49.44**
Ans	$\mathcal{K}_{dbp}$	846,391	5	–	728	25	–	–	–
	$\mathcal{S}_{dbp}$	846,391	5	1	728	25	162	70	7

(b)

		SQ_1^{btc}	SQ_2^{btc}	SQ_3^{btc}	SQ_4^{btc}	SQ_5^{btc}	SQ_6^{btc}	SQ_7^{btc}	SQ_8^{btc}
RCQ	$\mathcal{K}_{btc}$	491	91	44,591	1,933	100,829	1,093	491	11,649
	$\mathcal{S}_{btc}$	**133**	**28**	**4,126**	**239**	**577**	**197**	**166**	**1,317**
Time	$\mathcal{K}_{btc}$	224.98	114.87	–	12.87	4.45	4.11	55.01	–
	$\mathcal{S}_{btc}$	**57.35**	**11.07**	**492.49**	**3.27**	**2.21**	**2.44**	**26.49**	**191.53**
Ans	$\mathcal{K}_{btc}$	13,586,245	98	–	15	3	72	162	–
	$\mathcal{S}_{btc}$	13,586,245	98	96	15	3	72	162	26

an example. Without KB partitioning, evaluating SQ_7^{dbp} over $\mathcal{K}_{dbp}$ needs to answer 18,289 conjunctive queries over the ABox of $\mathcal{K}_{dbp}$, whereas in the situation of KB partitioning, the largest number of conjunctive queries needed to be answered becomes 377. This is similar for SQ_1^{dbp}, SQ_3^{dbp}, SQ_1^{btc}, SQ_4^{btc}, SQ_5^{btc}, SQ_6^{btc} and SQ_8^{btc}. Secondly, smaller ABoxes enable the finally rewritten queries to be evaluated much faster. Take SQ_6^{dbp} for example. Although answering SQ_6^{dbp} over $\mathcal{K}_{dbp}$ solely needs to answer 1 conjunctive query, due to the massive size of the ABox of $\mathcal{K}_{dbp}$, this query cannot be answered within 30 minutes. In the situation of KB partitioning, however, evaluating SQ_6^{dbp} can be completed over the ABox of each sub-KB in less than 13 minutes. This is similar for SQ_2^{dbp}, SQ_4^{dbp}, SQ_6^{dbp}, SQ_8^{dbp}, SQ_1^{btc}, SQ_2^{btc} and SQ_7^{btc}. Overall, the simple-query answering results indicate that KB partitioning equipped with parallelization can significantly improve the performance of simple-query answering over large-scale DL-Lite$_{\mathcal{A}}$ KBs.

Next, we show the effect of query partitioning combined with KB partitioning as well as sub-query value transfer on the performance of query answering. The results of evaluating the tested non-simple queries and the simple-queries SQ_3^{dbp}, SQ_6^{dbp}, SQ_3^{btc} and SQ_8^{btc} over the corresponding KBs and KB partitions are shown in Table 7. The motivation of testing the simple-queries SQ_3^{dbp}, SQ_6^{dbp}, SQ_3^{btc} and SQ_8^{btc} again is to demonstrate how query partitioning can function when KB partitioning fails to largely improve the performance of simple-query answering. The evaluated partitions of the corresponding tested queries are presented in the Appendix of this paper.

Generally speaking, by comparing the values of Time in different situations listed in Table 7(a) and (b), we can conclude that when a conjunctive query cannot be evaluated over the original KB efficiently, query partitioning combined with KB partitioning and sub-query value transfer can make this query to be evaluated in a very efficient way. For example, evaluating SQ_3^{dbp} over $\mathcal{K}_{dbp}$ cannot be done in 30 minutes, whereas the combination of the partitioning and optimization has completed the task in 10.8 seconds. This is similar for SQ_6^{dbp}, NQ_1^{dbp}, NQ_4^{dbp}-NQ_6^{dbp}, SQ_3^{btc}, SQ_8^{btc}, NQ_4^{btc}-NQ_6^{btc}. Such performance improvement mainly embodies in the following two aspects. On the one hand, no matter with or without KB partitioning, query partitioning can reduce the number of the rewritten conjunctive queries significantly. This can be seen from the values of RCQ listed in Table 7(a) and (b). For example, without query partitioning, evaluating NQ_6^{btc} over $\mathcal{K}_{btc}$ needs to answer 3299,661 rewritten queries over the ABox of $\mathcal{K}_{btc}$, whereas evaluating a partition of NQ_6^{btc} over $\mathcal{K}_{btc}$ decreases the number to 656, which is further downsized to 197 in the situation of KB partitioning (the maximum number of rewritten queries over one sub-KB). Similar cases can be seen for SQ_3^{btc}, SQ_8^{btc}, NQ_3^{btc}-NQ_5^{btc}, SQ_3^{dbp}, and NQ_4^{dbp}-NQ_6^{dbp}. On the other hand, even though query partitioning cannot reduce the number of rewritten queries significantly, such as for SQ_6^{dbp} and NQ_1^{dbp}, evaluating query partitions over KB partitions with parallelization and sub-query value transfer can speed up the procedure, as exemplified by NQ_5^{dbp}. Solely using query partitioning, NQ_5^{dbp} cannot be evaluated over $\mathcal{K}_{dbp}$ successfully, whereas combined with KB partitioning and sub-query value transfer, evaluating NQ_5^{dbp} has completed in 14.78 seconds.

We analyze the results in Table 7 in more details. Firstly, comparing the 7th with the 8th rows of Table 7(a) and (b), one can see that solely with query partitioning, queries that cannot be evaluated successfully still suffer from low efficiency, as exemplified by SQ_6^{dbp}, NQ_4^{dbp}-NQ_6^{dbp}, SQ_3^{btc}, SQ_8^{btc} and NQ_3^{btc}-NQ_6^{btc}. What is worse is that query partitioning may lead to poor performance for queries that can be originally answered quickly, such as NQ_2^{dbp}, NQ_3^{dbp}, NQ_1^{btc} and NQ_2^{btc}. In addition, as shown by the 11th rows of Table 7(a) and (b), it is clear that even combined with KB partitioning, such poor performance caused by query partitioning is not improved. Fortunately, this can be changed by sub-query value transfer, especially in the situation of KB partitioning, as seen from the 9th and 12th rows of Table 7(a) and (b). Secondly, it is undeniable that sub-query value transfer will increase the number of conjunctive queries eventually needed to be answered. This can be seen from Table 8, where for example, without sub-query value transfer, evaluating SQ_6^{dbp} just needs to answer 1 conjunctive query, but sub-query value transfer has caused the number to become $1 \times 238 = 238$. Nevertheless, in the end sub-query value transfer has made evaluating SQ_6^{dbp} from impossible to possible. Similar cases can be seen in NQ_2^{dbp}-NQ_6^{dbp} and the queries designed for BTC 2012 dataset. Note that the results of evaluating NQ_1^{dbp} indicate that if all the sub-queries in a query partition can be answered efficiently, then sub-query value transfer may not further speed up the procedure. Thirdly, in the situation of query partitioning, no matter with or without sub-query value transfer, KB partitioning can accelerate the overall query evaluation procedure, as evaluating queries over smaller KBs tends to be faster. This can be seen by respectively comparing the 8th row with the 11th row and the 9th row with the 12th row of Table 7(a) and (b), respectively. Finally, we notice that for queries already efficiently answerable over the original KB, our approach may not further improve the performance, as seen in NQ_2^{dbp}-NQ_3^{dbp} and NQ_1^{btc}-NQ_3^{btc}.

Table 7
Results of evaluating the tested queries over the corresponding KBs through query partitioning, where — denotes a query cannot be evaluated in 30 minutes, % denotes inapplicability, *pp* and *npp* respectively denote query answering with and without query partitioning, and *vt* and *nvt* respectively denote evaluating query partitions with and without sub-query value transfer. The best values of RCQ and Time are marked in bold.

(a)

		SQ_3^{dbp}	SQ_6^{dbp}	NQ_1^{dbp}	NQ_2^{dbp}	NQ_3^{dbp}	NQ_4^{dbp}	NQ_5^{dbp}	NQ_6^{dbp}
RCQ	$\mathcal{K}_{dbp}^{npp}$	1,145	**1**	**1**	4	**1**	4,237	29,732	–
	$\mathcal{K}_{dbp}^{pp+nvt}$	1,146	2	2	5	2	1,065	2,302	–
	$\mathcal{K}_{dbp}^{pp+vt}$	1,146	2	2	5	2	1,065	2,302	–
	$\mathcal{S}_{dbp}^{npp}$	**378**	**1**	%	%	%	%	%	%
	$\mathcal{S}_{dbp}^{pp+nvt}$	379	2	2	**2**	2	**126**	**385**	**12,958**
	$\mathcal{S}_{dbp}^{pp+vt}$	379	2	2	**2**	2	**126**	**385**	**12,958**
Time	$\mathcal{K}_{dbp}^{npp}$	–	–	–	1.07	60.75	–	–	–
	$\mathcal{K}_{dbp}^{pp+nvt}$	75.40	–	106.02	–	–	–	–	–
	$\mathcal{K}_{dbp}^{pp+vt}$	51.99	122.81	103.79	1.16	61.21	–	–	–
	$\mathcal{S}_{dbp}^{npp}$	201.86	824.03	%	%	%	%	%	%
	$\mathcal{S}_{dbp}^{pp+nvt}$	38.80	–	**22.82**	–	–	686.11	208.89	–
	$\mathcal{S}_{dbp}^{pp+vt}$	**10.80**	**63.33**	31.61	**0.49**	**18.75**	**679.53**	**14.78**	**188.96**
Ans	$\mathcal{K}_{dbp}^{npp}$	–	–	–	20	40	–	–	–
	$\mathcal{K}_{dbp}^{pp+nvt}$	1	–	8,164	–	–	–	–	–
	$\mathcal{K}_{dbp}^{pp+vt}$	1	162	8,164	20	40	–	–	–
	$\mathcal{S}_{dbp}^{npp}$	1	162	%	%	%	%	%	%
	$\mathcal{S}_{dbp}^{pp+nvt}$	1	–	–	–	–	3	138	–
	$\mathcal{S}_{dbp}^{pp+vt}$	1	162	8,164	20	40	3	138	40

(b)

		SQ_3^{btc}	SQ_8^{btc}	NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
RCQ	$\mathcal{K}_{btc}^{npp}$	44,591	11,649	91	91	8,281	123,873	312,131	3,299,661
	$\mathcal{K}_{btc}^{pp+nvt}$	**189**	120	92	92	182	17,704	3,522	656
	$\mathcal{K}_{btc}^{pp+vt}$	**189**	120	92	92	182	17,704	3,522	656
	$\mathcal{S}_{btc}^{npp}$	4,126	1,317	%	%	%	%	%	%
	$\mathcal{S}_{btc}^{pp+nvt}$	191	**70**	**29**	**29**	**56**	**1,318**	**305**	**197**
	$\mathcal{S}_{btc}^{pp+vt}$	191	**70**	**29**	**29**	**56**	**1,318**	**305**	**197**
Time	$\mathcal{K}_{btc}^{npp}$	–	–	11.91	3.03	–	–	–	–
	$\mathcal{K}_{btc}^{pp+nvt}$	–	–	–	–	–	–	–	–
	$\mathcal{K}_{btc}^{pp+vt}$	–	91.03	4.54	1.93	37.95	–	117.63	112.86
	$\mathcal{S}_{btc}^{npp}$	492.49	191.53	%	%	%	%	%	%
	$\mathcal{S}_{btc}^{pp+nvt}$	–	–	9.88	–	–	–	–	–
	$\mathcal{S}_{btc}^{pp+vt}$	**361.65**	**12.08**	**2.38**	**0.61**	**6.03**	**213.71**	**43.23**	**39.68**
Ans	$\mathcal{K}_{btc}^{npp}$	–	–	843	36	–	–	–	–
	$\mathcal{K}_{btc}^{pp+nvt}$	–	–	–	–	–	–	–	–
	$\mathcal{K}_{btc}^{pp+vt}$	–	26	843	36	38	25	2,092	1,685
	$\mathcal{S}_{btc}^{npp}$	96	26	%	%	%	%	%	%
	$\mathcal{S}_{btc}^{pp+nvt}$	–	–	843	–	–	–	–	–
	$\mathcal{S}_{btc}^{pp+vt}$	96	26	843	36	38	25	2,092	1,685

Table 8
Details of sub-query value transfer, where B_i denotes the number of variable bindings transferred from the previous $(i-1)$ evaluated sub-queries to the ith evaluated sub-query and R_i denotes the RCQ value of the ith evaluated sub-query.

	SQ_3^{dbp}		SQ_6^{dbp}		NQ_1^{dbp}		NQ_2^{dbp}		NQ_3^{dbp}		NQ_4^{dbp}		NQ_5^{dbp}		NQ_6^{dbp}			
	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_3	R_3
$\mathcal{K}_{dbp}$	5	1,145	238	1	2,3225	1	4	1	4	1	–	–	–	–	–	–	–	–
$\mathcal{S}_{dbp}$	5	378	238	1	2,3225	1	4	1	4	1	4	1	20	7	1	12,802	3	149

	SQ_3^{btc}		SQ_6^{btc}		NQ_1^{btc}		NQ_2^{btc}		NQ_3^{btc}		NQ_4^{btc}		NQ_5^{btc}		NQ_6^{btc}			
	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_2	R_2	B_3	R_3
$\mathcal{K}_{btc}$	98	491	5	91	1	91	6	91	4	91	5	7	25	91	61	74	78	7
$\mathcal{S}_{btc}$	98	166	5	28	1	28	6	28	4	91	5	2	25	28	61	65	78	2

7.4. The comparison with related systems

In this subsection we present the results of comparing our approach respectively with one related work on satisfiability checking and two state-of-the-art RDF triple stores for query answering.

For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, the classical way of checking the satisfiability of $\mathcal{K}$ is realized by the following two steps. Firstly, compute the negative closure $\mathsf{cln}(\mathcal{T})$ of $\mathcal{T}$ which is a set of axioms in the form $\gamma \sqsubseteq \neg\zeta$ or $\mathtt{Fun}(P)$, and generate a query for each

Table 9
The results of checking the satisfiability of the tested KBs and KB partitions by the classical approach (ClaA) and [39], where ++ denotes out of memory error.

		$\mathcal{K}_{dbp}$	$\mathcal{S}_{dbp}$	$\mathcal{K}_{btc}$	$\mathcal{S}_{btc}$
Time	ClaA	6.44	**0.07**	++	**0.15**
	[56]	**2.14**	0.82	**10.57**	4.03
Result	ClaA	False	False	++	False
	[56]	False	False	False	False

axiom in $\mathsf{cl}(\mathcal{T})$. Secondly, evaluate the generated queries over $\mathcal{A}$ to check whether there exist assertions in $\mathcal{A}$ that conflict with $\mathsf{cln}(\mathcal{T})$. Alternatively, the work of [39] provides a bottom-up way of satisfiability checking where the step of precomputing negative closure is discarded. Concretely, for every two assertions sharing an individual, the approach in [39] checks whether $\mathcal{T}$ entails the corresponding negative axiom. For example, if the two assertions are $A(a)$ and $P(a, b)$ then check whether $\mathcal{T}$ entails $A \sqsubseteq \neg\exists P$; and if the two assertions are $A(a)$, check whether $\mathcal{T}$ entails $A \sqsubseteq \neg A$. If the result is true then $\mathcal{K}$ is unsatisfiable; otherwise continue checking other pairs. The results of checking the satisfiability of the KBs in $\mathcal{S}_{dbp}$ ($\mathcal{S}_{btc}$) with parallelization and the KB $\mathcal{K}_{dbp}$ ($\mathcal{K}_{btc}$) by the classical approach and the approach illustrated in [39] are shown in Table 9. Note that, for the approach in [39], due to the large size of the ABoxes of the tested KBs, we pre-compute the groups of the assertions in an ABox that share individuals and then gradually load each group into main memory.

The results in Table 9 indicate the following three aspects. Firstly, from the results of $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ in Table 9, we can get that for the KBs with large-scale TBoxes, the approach in [39] may perform better then the classical approach for satisfiability checking, since the classic approach may spend much time to compute all the queries needed to be checked over the ABoxes of KBs. Secondly, from the results of $\mathcal{S}_{dbp}$ and $\mathcal{S}_{btc}$ in Table 9, we can obtain that if the queries needed to be evaluated over the ABoxes can be computed quickly, blindly trying pairs of assertions sharing individuals may slow down the satisfiability checking procedure. Thirdly, from the results in the 2nd row of Table 9, we can conclude that even for [39], KB partitioning equipped with parallelization still has the potential to improve the performance of satisfiability checking. Note that, for the approach in [39], if $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ are satisfiable, the performance improved by KB partitioning will be much more remarkable, since checking the satisfiability of $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ need to check 99 billions and 53,242 billions of individual assertion pairs, respectively. However, for the KBs in $\mathcal{S}_{dbp}$ ($\mathcal{S}_{btc}$), the numbers of assertion pairs needed to be checked are no more than 7 (226) billions.

On the other hand, in order to capture the abundant schema knowledge described in the open datasets, we propose to use DL-Lite$_{\mathcal{A}}$ techniques to reason with and query these datasets, i.e., considering these datasets as DL-Lite$_{\mathcal{A}}$ KBs rather than RDF graphs. The advantages lie in the availability of checking the satisfiability of the datasets as well as more certain answers being identified by replacing evaluating one query with evaluating many rewritten queries. Correspondingly, the disadvantage is that more time will be used to answer a query. In order to improve the overall performance, we provide a divide-and-conquer reasoning and query answering approach. Next, we present the performance difference between our divide-and-conquer strategy and the approach of evaluating queries directly without reasoning, and show the importance of reasoning as well as the rationality of partitioning for large-scale open datasets. For comparison, we selected two state-of-the-art, centralized triple stores, Jena-TDB (version 2.10.1)[9] and Virtuoso (version 7.2.4.2).[10] They both adopt triple table model to manage RDF graphs and support lightweight RDFS reasoning. The BTC 2012 dataset and the extended DBpedia_200 dataset (after dropping duplicates and materializing owl:sameAs assertions) were first loaded into the two triple stores, which respectively took about 30.55 and 9.71 hours for Virtuoso and 57.31 and 10.42 hours for Jena-TDB. The results of evaluating the tested queries over the corresponding datasets by our approach and Virtuoso and Jena-TDB without and with RDFS reasoning are shown in Table 10. These results indicate the following three aspects.

Firstly, by comparing the 1st with the 2nd and 3rd rows of Table 10(a) and (b), one can clearly see that even with KB and query partitioning, our approach is not as efficient as the two RDF stores which evaluate queries directly without taking reasoning into consideration. For example, it took our prototype system 4.08 seconds to answer SQ_2^{dbp}, whereas in Virtuoso and Jena-TDB the query was evaluated within 0.21 seconds. This is the same case for the queries SQ_2^{btc}, SQ_3^{btc}, SQ_5^{dbp} and many more.

Secondly, by comparing the 2nd with the 4th, and the 3rd with the 5th rows of Table 10(a) and (b), respectively, one can see that when reasoning is considered, the performance of the two triple stores drops significantly, and they do not perform as better as our approach. For instance, without reasoning, NQ_5^{btc} can be evaluated by Virtuoso and Jena-TDB in 2.41 and 0.21 seconds, respectively, whereas under RDFS reasoning, the two triple stores failed to evaluate the query within 15 minutes, due to the large size of the two tested datasets. On the other hand, taking advantage of KB and query partitioning, our prototype system succeeded in completing the task in 43.23 seconds, even though what being performed is reasoning in DL-Lite$_{\mathcal{A}}$ which is more expressive than RDFS. This is the same case for the queries NQ_4^{dbp}, SQ_1^{btc}, SQ_3^{btc}, SQ_7^{btc}, and NQ_6^{btc}, as shown by comparing the 1st with the 4th and 5th rows of Table 10(a) and (b). Moreover, when (one of) the two triple stores can answer the query under reasoning within given time, for 16 out of 21 such cases, our approach spent less time, as exemplified by SQ_1^{dbp} where Virtuoso and Jena-TDB respectively took 235.97 and 281.11 while ours 29.97 seconds. This indicates that when reasoning is taken into account, KB and query partitioning has the potential to improve the performance of query answering significantly.

Thirdly, by comparing the 6th with the 7th and 8th rows of Table 10(a) and (b), one can see that query answering without considering reasoning can miss many certain answers. Take NQ_5^{btc} as an example. Under DL-Lite$_{\mathcal{A}}$ semantics, i.e., considering the tested datasets as DL-Lite$_{\mathcal{A}}$ KBs, there are a total of 2092 certain answers in the BTC 2012 dataset for NQ_5^{btc}; whereas without reasoning, none of these answers can be obtained. This is the same case for the queries NQ_5^{dbp}, NQ_6^{dbp}, NQ_6^{btc}, SQ_1^{dbp}, SQ_1^{btc}, and so on.

Note that for query answering with reasoning, an alternative strategy is materialization [40–42], i.e., to compute the entailed individual assertions and add the entailments to the original datasets so that queries can be directly evaluated over the materialized datasets without reasoning. However, such materialization is incomplete for query answering in DL-Lite$_{\mathcal{A}}$. Take the KB $\mathcal{K}$ consisting of the following axiom and assertion described in the BTC 2012 dataset as an example.

$$\begin{aligned} &\mathit{Event} \sqsubseteq \exists \mathit{factor\_of}.\mathit{Event} \\ &\mathit{Event}(a) \end{aligned}$$

Consider the following query Q asking for the events that are factors of other events:

$$\mathit{Event}(?x) \wedge \mathit{factor\_of}(?x, ?y) \wedge \mathit{Event}(?y) \rightarrow q(?x)$$

9 https://jena.apache.org/index.html.

10 http://vos.openlinksw.com/owiki/wiki/VOS/VOSMakeWindows.

Table 10
Results of answering the tested queries by Virtuoso and Jena-TDB, where `Vir` and `TDB` respectively denote query answering by Virtuoso and Jena-TDB without reasoning, `ViR` and `TDR` respectively denote query answering by Virtuoso and Jena-TDB with RDFS reasoning, `Our` denotes our divide-and-conquer query answering approach, – denotes a query cannot be evaluated within 15 minutes, and M denotes million. The best values of `Time` and `Ans` are marked in bold. When reasoning is considered, the best values of `Time` are marked in italic.

(a)

		SQ_1^{dbp}	SQ_2^{dbp}	SQ_3^{dbp}	SQ_4^{dbp}	SQ_5^{dbp}	SQ_6^{dbp}	SQ_7^{dbp}	SQ_8^{dbp}	NQ_1^{dbp}	NQ_2^{dbp}	NQ_3^{dbp}	NQ_4^{dbp}	NQ_5^{dbp}	NQ_6^{dbp}
T	Our	*29.97*	*4.08*	*10.80*	***1.40***	45.74	60.33	*30.51*	*49.44*	*22.82*	*0.49*	18.75	*679.53*	*14.78*	*188.96*
i	Vir	**7.20**	0.21	5.27	9.84	4.68	**4.67**	**2.81**	6.17	**2.52**	0.29	**0.59**	0.24	0.89	**0.45**
m	TDB	103.78	0.21	**0.30**	1.54	**0.21**	5.51	3.64	**0.65**	148.25	**0.25**	36.23	**0.20**	**0.31**	23.49
e	ViR	235.97	25.02	–	16.15	*18.88*	*19.88*	447.19	62.17	36.64	15.83	*15.61*	–	510.72	251.48
	TDR	281.11	4.05	–	4.38	589.46	33.86	569.66	155.97	199.56	4.17	49.96	–	105.19	–
	Our	**846,391**	5	1	728	25	162	70	7	8,164	20	40	3	**138**	**40**
A	Vir	463,820	5	1	728	25	162	70	7	8,164	20	40	3	0	0
n	TDB	463,820	5	1	728	25	162	70	7	8,164	20	40	3	0	0
s	ViR	**846,391**	5	–	728	25	162	70	7	8,164	20	40	–	**138**	**40**
	TDR	**846,391**	5	–	728	25	162	70	7	8,164	20	40	–	**138**	–

(b)

		SQ_1^{btc}	SQ_2^{btc}	SQ_3^{btc}	SQ_4^{btc}	SQ_5^{btc}	SQ_6^{btc}	SQ_7^{btc}	SQ_8^{btc}	NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
T	Our	***57.35***	*11.07*	*361.65*	*3.27*	*2.21*	*2.44*	*26.49*	*12.08*	*2.38*	*0.61*	*6.03*	213.71	*43.23*	*39.68*
i	Vir	127.41	1.27	1.46	**2.65**	**0.22**	**1.35**	**0.42**	**1.46**	**0.21**	**0.28**	**0.25**	**0.41**	2.41	**2.33**
m	TDB	305.74	**0.37**	**0.37**	4.63	0.30	5.54	44.32	3.82	–	13.01	0.42	1.50	**0.21**	5.90
e	ViR	–	30.46	–	28.87	29.99	30.77	–	45.29	29.87	29.44	29.81	*72.99*	–	–
	TDR	–	25.91	–	19.52	21.42	24.28	–	65.80	–	105.31	18.60	157.94	–	–
	Our	**13M**	**98**	**96**	**15**	3	**72**	**162**	26	**843**	**36**	**38**	25	**2,092**	**1,685**
A	Vir	12M	7	5	0	3	51	138	26	1	4	34	25	0	1,187
n	TDB	12M	7	5	0	3	51	138	26	1	4	34	25	0	1,187
s	ViR	–	**98**	–	**15**	3	**72**	–	26	**843**	**36**	**38**	25	–	–
	TDR	–	**98**	–	**15**	3	**72**	–	26	–	**36**	**38**	25	–	–

Evaluating Q over $\mathcal{K}$ by materialization obtains an empty answer set, as none extra assertions are added to $\mathcal{K}$. On the other hand, by our approach which adopts the semantics of conjunctive queries and DL-Lite$_\mathcal{A}$ KBs, asking Q over $\mathcal{K}$ obtains a certain answer (a).

8. Related work

Our study in this paper aims at providing powerful and efficient approaches to reason and query Web-scale and complex open datasets described by Semantic Web standards based on DL-Lite$_\mathcal{A}$ techniques. Related work is presented as follows in four aspects, query answering over Linked Open Data, KB modularity and partitioning, query rewriting optimization, and fast satisfiability checking.

8.1. Linked open data query answering

The significant growth of the Linked Open Data has motivated a considerable amount of works on Linked Open Data query answering. We divide them into centralized, distributed, and hybrid methods for comparison and discussion.

Centralized methods. Centralized methods replicate the content of remote data resources into a local store over which SPARQL queries can be evaluated locally, as for instance [12,43]. Merits of centralization include high reliability and efficiency, as demonstrated by Google and its alikes in dealing with text-based Web documents. Drawbacks, on the other hand, include inflexibility especially when the underlying data resources exhibit high update rates.

Distributed methods. Distributed methods, focusing on the distributed feature of Linked Data, mainly consist of federated and link-based query processing. The core idea of federated query processing, as in [13,44–49], is to execute queries over a federation of endpoints independently through these three steps: (1) split the input query into sub-queries; (2) send the relevant sub-queries to individual endpoints in situ; and (3) subsequently merge and process the results. Link-based query processing, as in [14,50–54], aims at dynamically selecting, retrieving and building a dataset for each SPARQL query at runtime. Moreover, works like [15,54–56] combine federated and link-based query processing. Compared with centralized ways, distributed methods adapt to the updates of data, whereas spend more time in detecting relevant resources and following links when evaluating queries.

Hybrid methods. Hybrid methods focus on combining centralized query processing with distributed query processing. An interesting issue is how to take advantage of the local and remote querying techniques, respectively, while complementing each other on both theoretical and engineering level. A number of works have tackled this issue, including [55–58].

In terms of Linked Open Data query answering, our approach falls in the *centralized* category. Most of the works on querying Linked Open Data mentioned above, no matter centralized or distributed, concentrate on efficiency and scalability whereas reasoning is often ignored or only lightweight RDFS or OWL 2 RL/RDF reasoning is performed [14]. Our approach pursues both expressivity and scalability, capturing more schema knowledge and handling more expressive queries with the support of DL-Lite$_\mathcal{A}$ query answering techniques. Moreover, for scalability, we provide a way of partitioning DL-Lite$_\mathcal{A}$ KBs into independent smaller chunks so that the massive knowledge in Linked Open Data can be managed by parallel techniques. Expressive reasoning under distributed situations is harder than centralized ones, and shall be explored in our future work.

8.2. KB Modularity and partitioning

Modularization of DL KBs has been addressed by [59–61], where the theoretical modularity model has been generated. Roughly speaking, for a DL KB $\mathcal{K}$ and a name set Σ, the motivation of modularity is to extract a subset $\mathcal{K}'$ of $\mathcal{K}$ such that $\mathcal{K} \models \alpha$ iff $\mathcal{K}' \models \alpha$ for each axiom (assertion) α solely containing the names in Σ. In the following, we discuss this kind of works in terms of TBox modularity and ABox modularity.

TBox modularity. Many works in this branch focus on TBox modularity [62–66]. For the DL-Lite family, work [67] discusses conservation extension and work [68] presents minimal module extraction. In Definition 5 of this paper, we provide a way of computing a sub-TBox $\mathcal{T}'$ for a given sub-ABox $\mathcal{A}'$ of the ABox of a DL-Lite$_\mathcal{A}$

KB $(\mathcal{T}, \mathcal{A})$. We share with the TBox modularity works in extracting subsets from TBox, while the condition and goal are different. Our extraction is under a given sub-ABox $\mathcal{A}'$, whereas the modularity works is based on a given name set Σ. For the goal, we concentrate on satisfiability checking and conjunctive query answering rather then axiom or assertion entailment checking. More precisely, we require that $(\mathcal{T}', \mathcal{A})$ and $(\mathcal{T}, \mathcal{A})$ coincide in terms of the considered reasoning tasks, whereas modularity works enforce that same axioms or assertions composed of names in Σ can be inferred by both the module and its original KB.

ABox modularity. The work of [69] provides a formal definition of ABox modularity over $\mathcal{SHIQ}$ and gives concrete ways of extracting ABox modules for a given name set. Concretely, for a $\mathcal{SHIQ}$ [70] KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and individual set S, $\mathsf{M} \subseteq \mathcal{A}$ is called a module of $\mathcal{A}$ iff for any individual assertion α that contains individuals in M, $(\mathcal{T}, \mathsf{M}) \models \alpha$ iff $\mathcal{K} \models \alpha$. Unlike [69], works in [71], [72] and [73] study ABox modularity in terms of ABox partitioning over $\mathcal{SHIF}$, $\mathcal{SHIQ}$, and $\mathcal{SHI}$, respectively. Concretely, for a corresponding KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a module M of $\mathcal{A}$ is a set of ABoxes such that $\mathcal{K} \models \alpha$ iff there exists $\mathcal{A}' \in \mathsf{M}$ and $(\mathcal{T}, \mathcal{A}') \models \alpha$ for every individual assertion α. The difference is that work [71] requires that the ABoxes in M are subsets of $\mathcal{A}$, work [72] requires that the ABoxes in M not only are subsets of $\mathcal{A}$ but also are pairwise disjoint, and work [73] does not require the ABoxes to be subsets of $\mathcal{A}$.

Compared with these works, we study a tractable language DL-Lite$_{\mathcal{A}}$, and concentrate on satisfiability checking and conjunctive query answering rather than entailment checking of individual assertions. Moreover, we provide a way of ABox partitioning with linear data complexity where reasoning is not necessary.

Another direction of research related to us is RDF graph partitioning with the motivation of processing the grand scale of RDF data on the Web. The state-of-the-art approaches in this topic include hash partitioning [33–35,74–77] and graph partitioning [78–82]. In these works, only the graph structure of RDF data is considered whereas the semantics of RDFS and OWL vocabulary terms, i.e., axioms in the data, are usually ignored. Conversely, we consider RDF graphs as DL-Lite$_{\mathcal{A}}$ KBs such that constraints in these graphs can be captured. Moreover, for high efficiency and scalability, motivated by hash partitioning of RDF graphs, we provide a divide-and-conquer reasoning and query answering approach for DL-Lite$_{\mathcal{A}}$.

8.3. Query rewriting optimization

First-order rewritability (FOR) of conjunctive queries is a highly desirable property, as it allows to take advantage of the optimized database management systems to store and query the ABoxes of KBs. However, the bottleneck of FOR is that for KBs with a large number of axioms, the rewriting may result in an expensive evaluation of an exponential size of queries. This constitutes a serious limitation for practical applications. Due to this, many works have been proposed on optimizing the procedure of conjunctive query rewriting [83–90]. Most of these works focus on reducing the rewritten queries as many as possible. Concretely, the work in [83] presents an optimized query rewriting algorithm for DL-Lite family by rewriting a user query into a Datalog program whose rules encode only necessary steps to prevent the generation of queries. Work [84] presents a resolution-based rewriting algorithm for DL-Lite$_{\mathcal{R}}$ (a sub-language of DL-Lite$_{\mathcal{A}}$), where the issue of the useless factorizations is addressed by directly handling existential quantification through proper functional terms. The work of [85] presents a more efficient algorithm for DL-Lite$_{\mathcal{R}}$ based on the selective and stratified application of resolution rules. It makes use of the query structure and applies a restricted sequence of resolutions that may lead to useful and redundant-free rewritings. An alternative query rewriting technique for DL-Lite$_{\mathcal{R}}$ is presented in [86], where in most practical cases the rewritings are correct and of polynomial size, although in general the obtained rewritings can be of exponential size. In [87], the problem of computing query rewritings for DL-Lite$_{\mathcal{R}}$ in an incremental way is investigated. Also a technique that computes an extended query by extending a previously computed rewriting of the initial query is proposed. The works [88–93] discuss query rewriting optimization in the Datalog$^{\pm}$ category [94]. From a different point of view, the work of [95] concentrates on query rewriting optimization in DL-Lite$_{\mathcal{R}}$ by searching within a set of alternative equivalent first-order logic queries and choosing one with the minimal evaluation cost over a relational database management system.

Compared with these works, our approach features query rewriting optimization in DL-Lite$_{\mathcal{A}}$ from the perspective of query partitioning, and the experiments on real-world datasets show that the number of rewritten queries can be significantly reduced.

8.4. Fast satisfiability checking

To improve the performance of satisfiability checking in OWL ontology has been a research focus of DL community. Among them, in [96] the authors propose to use the summary of ontology to reduce satisfiability checking to a small subset of Abox, and proves that the technique is sound and complete in $\mathcal{SHIN}$. The work of [97] presents a fast approximate ABox consistency checking approach based on machine learning, and applies the provided approach to the problem of consistency checking of a large number of ABoxes adhering to the same TBox. The work in [39] proposes an extension of incomplete reasoning methods for checking the consistency of a large number of ABoxes against a given TBox based on DL-Lite$_{\mathcal{A}}$ techniques, which is sound and incomplete for the ontologies exceeding the expressivity of DL-Lite$_{\mathcal{A}}$.

Different with these works, in this paper we propose to improve the performance of satisfiability checking in DL-Lite$_{\mathcal{A}}$ from the perspective of KB partitioning, i.e., by checking the satisfiability of smaller sub-KBs to obtain the satisfiability of the original KB. Experiments on real-world, Web-scale datasets demonstrate the efficiency of our approach. To the best of our knowledge, this is the first study on optimizing the procedure of satisfiability checking in DL-Lite$_{\mathcal{A}}$.

9. Conclusions

With the motivation of consuming Web-scale and complex open datasets described by Semantic Web standards efficiently, in this paper, we propose a divide-and-conquer reasoning and query answering approach for DL-Lite$_{\mathcal{A}}$ to cater for the real-world scalability. The basic idea is to partition both KBs and queries into smaller chunks and decompose the original considered reasoning tasks into a group of independent subtasks. Motivated by hash partitioning of RDF graphs, our divide-and-conquer approach is developed by subsequently discussing the following three problems from both theoretical and practical perspectives: (1) partition DL-Lite$_{\mathcal{A}}$ KBs into small chunks with the local feature of both satisfiability checking and simple-query answering; (2) based on simple-queries, partition conjunctive queries into smaller sub-queries and evaluate them over KB partitions with the desired local feature; and (3) optimize the procedure of evaluating query partitions over KB partitions. Experiments on two Web-scale open datasets, i.e., DBpedia and BTC 2012 dataset, demonstrate the rationality and efficiency of the overall proposed divide-and-conquer approach.

Future work will mainly contain the following aspects. Firstly, in terms of KB partitioning, we intend to design a mathematical model to analyze the impact of the number and size of the sub-KBs in a SCSQA-local partition of a DL-Lite$_{\mathcal{A}}$ KB on the performance of satisfiability checking and query answering. Obviously, a DL-Lite$_{\mathcal{A}}$

Table 11
Mappings for all the prefixes used.

dbo:	http://dbpedia.org/ontology/	skos:	http://www.w3.org/2004/02/skos/core#
dbp:	http://dbpedia.org/property/	foaf:	http://xmlns.com/foaf/0.1/
dbr:	http://dbpedia.org/resource/	rdfs:	http://www.w3.org/2000/01/rdf-schema#
swrc:	http://swrc.ontoware.org/ontology#	fb:	http://rdf.freebase.com/ns/m/
odp:	http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#	sc:	http://data.semanticweb.org/person/

Table 12
Evaluated query partitions of the tested queries. Note that in the scenario of query partitioning equipped with sub-query value transfer, the order of evaluating the sub-queries in a query partition is $q_1, \ldots, q_n$.

SQ_3^{dbp}	$\{ q_1 : rdfs : label(?x, 'Alan\ Turing@en') \rightarrow q(?x),\ q_2 : dbo : Person(?x) \rightarrow q(?x) \}$	SQ_3^{btc}	$\{ q_1 : rdfs : label(?x, 'Tim\ Berners-Lee') \rightarrow q(?x),\ q_2 : foaf : Person(?x) \rightarrow q(?x) \}$
SQ_6^{dbp}	$\{ q_1 : skos : subject(?x, dbr : Category : Computer\_pioneers) \rightarrow q(?x),\ q_2 : foaf : name(?x, ?y) \wedge rdfs : comment(?x, ?z) \rightarrow q(?x, ?y, ?z) \}$	SQ_8^{btc}	$\{ q_1 : dbo : Film(?x) \wedge dbo : writer(?x, dbr : Robert\_Thoeren) \wedge dbo : director(?x, ?y) \wedge foaf : primaryTopic(?d, ?x) \rightarrow q(?x, ?y, ?z, ?d),\ q_2 : rdfs : label(?x, ?z) \rightarrow q(?x, ?z) \}$
NQ_1^{dbp}	$\{ q_1 : skos : subject(dbr : Alan\_Turing, ?x) \wedge skos : subject(?y, ?x) \rightarrow q(?x, ?y),\ q_2 : dbp : knownFor(?y, ?z) \wedge dbp : prizes(?y, ?d) \rightarrow q(?y, ?z, ?d) \}$	NQ_1^{btc}	$\{ q_1 : dbp : prizes(14 : 01w7np, ?x) \rightarrow q(?x),\ q_2 : dbp : prizes(?y, ?x) \wedge rdfs : label(?y, ?z) \rightarrow q(?x, ?y, ?z) \}$
NQ_2^{dbp}	$\{ q_1 : dbo : writer(?x, dbr : Robert\_Thoeren) \wedge dbo : director(?x, ?y) \rightarrow q(?x, ?y),\ q_2 : rdfs : label(?y, ?z) \rightarrow q(?y, ?z) \}$	NQ_2^{btc}	$\{ q_1 : dbo : director(?x, dbr : George\_Lucas) \wedge dbo : writer(?x, ?y) \rightarrow q(?x, ?y),\ q_2 : rdfs : label(?y, ?z) \rightarrow q(?y, ?z) \}$
NQ_3^{dbp}	$\{ q_1 : rdfs : label(?x, 'Alan\ Turing@en') \wedge dbo : influencedBy(?y, ?x) \rightarrow q(?x, ?y),\ q_2 : rdfs : label(?y, ?z) \rightarrow q(?y, ?z) \}$	NQ_3^{btc}	$\{ q_1 : rdfs : label(?x, 'Alan\ Turing@en') \wedge dbo : influencedBy(?y, ?x) \rightarrow q(?x, ?y),\ q_2 : rdfs : label(?y, ?z) \rightarrow q(?y, ?z) \}$
NQ_4^{dbp}	$\{ q_1 : dbo : Work(?x) \wedge dbo : writer(?x, dbr : Robert\_Thoeren) \wedge dbo : director(?x, ?y) \wedge foaf : primaryTopic(?z, ?x) \rightarrow q(?x, ?y, ?z),\ q_2 : dbo : birthDate(?y, ?d) \rightarrow q(?y, ?d) \}$	NQ_4^{btc}	$\{ q_1 : dbo : Work(?x) \wedge dbo : writer(?x, dbr : Robert\_Thoeren) \wedge dbo : director(?x, ?y) \wedge rdfs : comment(?x, ?z) \wedge foaf : primaryTopic(?d, ?x) \rightarrow q(?x, ?y, ?z, ?d),\ q_2 : rdfs : comment(?y, ?e) \rightarrow q(?y, ?e) \}$
NQ_5^{dbp}	$\{ q_1 : dbo : Person(?x) \wedge skos : subject(?x, dbr : Category : Computer\_pioneers) \wedge rdfs : comment(?x, ?y) \wedge dbo : doctoralAdvisor(?x, ?z) \rightarrow q(?x, ?y, ?z),\ q_2 : dbo : Scientist(?z) \rightarrow q(?z) \}$	NQ_5^{btc}	$\{ q_1 : foaf : Person(?x) \wedge dbp : prizes(?x, fb : 0dt39) \wedge rdfs : comment(?x, ?y) \wedge dbo : doctoralAdvisor(?x, ?z) \rightarrow q(?x, ?y, ?z),\ q_2 : rdfs : label(?z, ?e) \rightarrow q(?z, ?e) \}$
NQ_6^{dbp}	$\{ q_1 : dbo : Scientist(?y) \wedge dbp : knownFor(?y, dbr : Turing\_Award) \rightarrow q(?y),\ q_2 : dbo : Organisation(?x) \wedge dbo : employer(?y, ?x) \wedge \wedge odp : hasLocation(?x, ?z),\ q_3 : dbo : PopulatedPlace(?z) \rightarrow q(?z) \}$	NQ_6^{btc}	$\{ q_1 : foaf : Person(?y) \wedge prizes(?y, fb : 0dt39) \wedge dbo : field(?y, ?z) \wedge dbo : knownFor(?y, ?d) \rightarrow q(?y, ?z, ?d),\ q_2 : dbo : EducationalInstitution(?x) \wedge dbo : almaMater(?y, ?x),\ q_3 : rdfs : label(?d, ?e) \rightarrow q(?d, ?e) \}$

KB may have many SCSQA-partitions and each performs differently for the considered reasoning tasks. To identify KB partitions with the optimal performance is crucial for real-life applications. Secondly, in terms of query partitioning, if a conjunctive query is not simple-query reducible then for completeness multiple query partitions have to be answered, thus strategies shall be developed to reduce the number of query partitions needed to be evaluated. Moreover, when two query partitions have the same certain answer set over a partition of a DL-Lite$_{\mathcal{A}}$ KB, deciding which one can be evaluated faster is also important for improving the overall query answering performance. Thirdly, we plan to discuss partitioning DL-Lite$_{\mathcal{A}}$ KBs and conjunctive queries based on other types of queries rather than simple-queries. Lastly, applying the main idea of our divide-and-conquer approach to other DL languages to improve the performance of satisfiability checking and conjunctive query answering is definitely worth exploring.

Acknowledgments

We thank the anonymous reviewers of this article for their invaluable and constructive comments and suggestions. This work has been supported by the National Key Research and Development Program of China under grants 2017YFC1700300, 2017YFB1002300 and 2016YFB1000902 and the Natural Science Foundation of China grants 61232015 and 61621003.

Appendix

See Tables 11 and 12.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Sci. Am. 284 (5) (2001) 34–43.
[2] R. Cyganiak, D. Wood, M. Lanthaler, RDF 11 Concepts and Abstract Syntax - W3C Recommendation 25 February 2014, https://www.w3org/TR/rdf11-concepts/.
[3] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer, second ed., 2012, W3C Recommendation 11, https://www.w3org/TR/owl2-primer/.
[4] C. Bizer, T. Heath, T. Berners-Lee, Linked data - the story so far, Int. J. Semant. Web Inf. Syst. 5 (3) (2009) 1–22.
[5] T. Heath, C. Bizer, Linked data: Evolving the web into a global data space, in: Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool, 2011.
[6] S. O'Riain, E. Curry, A. Harth, XBRL And open data for global financial ecosystems: A linked data approach, Int. J. Account. Inf. Syst. 13 (2) (2012) 141–162.
[7] A. Polleres, S. Steyskal, Semantic web standards for publishing and integrating open data, in: Standards and Standardization: Concepts, Methodologies, Tools, and Applications, 2015.
[8] F. Bauer, M. Kaltenböck, Linked open data: The essentials, Edition mono/monochrom, Vienna, Austria, 2011.
[9] L. Yu, Linked open data, in: A Developer's Guide to the Semantic Web, 2011.
[10] C. Bizer, J. Lehmannb, G. Kobilarova, S. Auer, C. Becker, R. Cyganiakc, S. Hellmannb, Dbpedia - A crystallization point for the web of data, J. Web Semant. 7 (3) (2009) 154–165.
[11] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008.
[12] B. Bishop, A. Kiryakov, D. Ognyanov, I. Peikov, Z. Tashev, R. Velkov, Factforge: A fast track to the web of data, Semant. Web 2 (2) (2011) 157–166.
[13] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: Optimization techniques for federated query processing on linked data, in: Proceedings of the 10th International Semantic Web Conference, 2011.
[14] J. Umbrich, A. Hogan, A. Polleres, S. Decker, Link traversal querying for a diverse web of data, Semant. Web. 6 (6) (2012) 585–624.

[15] S.J. Lynden, I. Kojima, A. Matono, A. Nakamura, M. Yui, A hybrid approach to linked data query processing with time constraints, in: Proceedings of the WWW 2013 Workshop on Linked Data on the Web, 2013.
[16] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003.
[17] I. Horrocks, O. Kutz, U. Sattler, The Irresistible SRIQ, in: Proceedings of the OWLED 2005 Workshop on OWL: Experiences and Direction, 2005.
[18] Y. Kazakov, B. Motik, A resolution-based decision procedure for SHOIQ, J. Autom. Reason. 40 (2–3) (2008) 89–116.
[19] C. Lutz, D. Toman, F. Wolter, Conjunctive query answering in the description logic EL using a relational database system, in: Proceedings of the 21th International Joint Conference on Artificial, 2009.
[20] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible SROIQ, in: Proceedings of the 21th National Conference on Artificial Intelligence, 2006.
[21] B. Motik, P.F. Patel-Schneider, B.C. Grau, OWL 2 Web Ontology Language Direct Semantics, second ed., 2012, W3C Recommendation 11, https://www.w3org/TR/owl2-direct-semantics/.
[22] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2002.
[23] A. Rector, Description Logics in Medical Informatics, 2003.
[24] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The DL-Lite family, J. Autom. Reason. 39 (3) (2007) 385–429.
[25] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The DL-Lite family and relations, J. Artificial Intelligence Res. 36 (2009) 1–69.
[26] A. Poggi, L. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semant. 10 (2008) 133–173.
[27] A. Poggi, Structured and semi-structured data integration (Ph.D. thesis), Dipartimento di Informatica e Sistemistica, Università di Roma La Sapienza, 2006.
[28] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, Ontology-based database access, in: Proceedings of the 15th Italian Symposium on Advanced Database Systems, 2007.
[29] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to ontology-based data access, in: Proceedings of the 22th International Joint Conference on Artificial Intelligence, 2011.
[30] M. Rodriguez-Muro, R. Kontchakov, M. Zakharyaschev, Ontology-based data access: Ontop of databases, in: Proceedings of the 12th International Semantic Web Conference, 2013.
[31] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, M. Zakharyaschev, Ontology-based data access: A survey, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018.
[32] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, Ontologies and databases: The DL-Lite approach, in: S. Tessaris, et al., Reasoning Web. Semantic Technologies for Information Systems. Reasoning Web 2009, in: Lecture Notes in Computer Science, vol. 5689, Springer, Berlin, Heidelberg.
[33] K. Lee, L. Liu, Scaling queries over big RDF graphs with semantic hash partitioning, Proc. VLDB Endow. 6 (14) (2013) 1894–1905.
[34] J.M. Giménez-García, J.D. Fernández, M.A. Martínez-Prieto, Mapreduce-based solutions for scalable SPARQL querying, Open J. Semant. Web 1 (1) (2014) 1–18.
[35] R. Harbi, I. Abdelaziz, P. Kalnis, N. Mamoulis, Evaluating SPARQL queries on massive RDF datasets, Proc. VLDB Endow. 8 (12) (2015) 1848–1851.
[36] Z.G. Ives, N.E. Tayor, Sideways information passing for push-style query processing, in: Proceedings of the IEEE International Conference on Data Engineering, 2008.
[37] T. Neumann, G. Weikum, Scalable join processing on very large RDF graphs, in: Proceedings of the ACM SIGMOD International Conference on Management of data, 2009.
[38] O. Hartig, C. Buil-Aranda, Bindings-restricted triple pattern fragments, in: Proceedings of the OTM Confederated International Conferences On the Move to Meaningful Internet Systems, 2016.
[39] C. Meilicke, D. Ruffinelli, A. Nolle, H. Paulheim, H. Stuckenschmidt, Fast ABox consistency checking using incomplete reasoning and caching, in: Proceedings of the International Joint Conference on Rules and Reasoning, 2017.
[40] B. Motik, Y. Nenov, R. Piro, I. Horrocks, D. Olteanu, Parallel materialisation of datalog programs in centralised, main-memory RDF systems, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.
[41] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, J. Banerjee, RDFox: A highly-scalable RDF store, in: Proceedings of the International Semantic Web Conference, 2015.
[42] J. Weaver, J.A. Hendler, Parallel materialization of the finite RDFS closure for hundreds of millions of triples, in: Proceedings of the International Semantic Web Conference, 2009.
[43] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, G. Tummarello, Sindice.com: a document-oriented lookup index for open linked data, J. Metadata Semant. Ontol. 3 (1) (2008) 37–52.
[44] M. Acosta, M.E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus, ANAPSID: An adaptive query processing engine for SPARQL endpoints, in: Proceedings of the 10th International Semantic Web Conference, 2011.
[45] C. Buil-Aranda, M. Arenas, O. Corcho, Semantics and optimization of the SPARQL 1.1 federation extension, in: Proceedings of the 8th Extended Semantic Web Conference, 2011.
[46] B. Quilitz, U. Leser, Querying distributed RDF data sources with SPARQL, in: Proceedings of the 5th Extended Semantic Web Conference, 2008.
[47] O. Görlitz, S. Staab, Federated data management and query optimization for linked open data, New Directions in Web Data Management 1, 2011.
[48] A. Harth, J. Umbrich, A. Hogan, S. Decker, YARS2: A federated repository for querying graph structured data from the web, in: Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, 2007.
[49] G. Montoya, H. Skaf-Molli, P. Molli, M.E. Vidal, Federated SPARQL queries processing with replicated fragments, in: Proceedings of the 14th International Semantic Web Conference, 2015.
[50] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.U. Sattler, J. Umbrich, Data summaries for on-demand queries over linked data, in: Proceedings of the 19th international conference on world wide web, 2010.
[51] O. Hartig, J.C. Freytag, Foundations of traversal based query execution over Linked Data, in: Proceedings of the 23rd ACM Conference on Hypertext and Social Media, 2012.
[52] O. Hartig, Zero-knowledge query planning for an iterator implementation of link traversal based query execution, in: Proceedings of the 8th Extended Semantic Web Conference, 2011.
[53] O. Hartig, C. Bizer, J.C. Freytag, Executing SPARQL queries over the web of linked data, in: Proceedings of the 8th International Semantic Web Conference, 2009.
[54] G. Ladwig, T. Tran, Linked data query processing strategies, in: Proceedings of the 9th International Semantic Web Conference, 2010.
[55] O. Hartig, A. Langegger, A database perspective on consuming linked data on the web, Datenbank-Spektrum 10 (2) (2010) 57–66.
[56] J. Umbrich, M. Karnstedt, A. Hogan, J.X. Parreira, Freshening up while staying fast: Towards hybrid SPARQL queries, in: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management, 2012.
[57] G. Ladwig, T. Tran, SIHJoin: Querying remote and local Linked Data, in: Proceedings of the 8th Extended Semantic Web Conference, 2011.
[58] M.M. Sabri, A hybrid framework for online execution of linked data queries, in: Proceedings of the 24th International Conference on World Wide Web Companion, 2015.
[59] B.C. Grau, I. Horrocks, Y. Kazakov, U. Sattler, Just the right amount: Extracting modules from ontologies, in: Proceedings of the 16th International Conference on World Wide Web, 2007.
[60] B.C. Grau, I. Horrocks, Y. Kazakov, U. Sattler, A logical framework for modularity of ontologies, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007.
[61] B.C. Grau, I. Horrocks, Y. Kazakov, U. Sattler, Modular reuse of ontologies: theory and practice, J. Artificial Intelligence Res. 31 (2008) 273–318.
[62] C. Del Vescovo, B. Parsia, U. Sattler, T. Schneider, The modular structure of an ontology: an empirical study, in: Proceedings of the 23th International Workshop on Description Logics, 2010.
[63] B.C. Grau, B. Parsia, E. Sirin, A. Kalyanpur, Modularity and web ontologies, in: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning, 2006.
[64] F. Martín-Recuerda, D. Walther, Fast modularisation and atomic decomposition of ontologies using axiom dependency hypergraphs, in: Proceedings of the 13th International Semantic Web Conference, 2014.
[65] B. Konev, C. Lutz, D. Walther, F. Wolter, Model-theoretic inseparability and modularity of description logic ontologies, Artificial Intelligence 203 (2013) 66–103.
[66] E. Botoeva, B. Konev, C. Lutz, V. Ryzhikov, F. Wolter, M. Zakharyaschev, Inseparability and conservative extensions of description logic ontologies: A survey, in: Reasoning Web: Logical Foundation of Knowledge Graph Construction and Query Answering, 2016.
[67] R. Kontchakov, F. Wolter, M. Zakharyaschev, Modularity in DL-Lite, in: Proceedings of the 20th International Workshop on Description Logics, 2007.
[68] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, M. Zakharyaschev, Minimal module extraction from DL-Lite ontologies using QBF solvers, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, 2009.
[69] J. Xu, P. Shironoshita, U. Visser, N. John, M. Kabuka, Module extraction for efficient object queries over ontologies with large aboxes, Artif Intell Appl. 2 (1) (2015) 8–31.
[70] I. Horrocks, U. Sattler, S. Tobies, Reasoning with individuals for the description logic SHIQ, in: Proceedings of the 17th International Conference on Automated Deduction, 2000.
[71] Y. Guo, J. Heflin, A scalable approach for partitioning OWL knowledge bases, in: Proceedings of the 2th International Workshop on Scalable Semantic Web Knowledge Base Systems, 2006.

[72] J. Du, Y.D. Shen, Partitioning Aboxes based on converting DL to plain datalog, in: Proceedings of the 20th International Workshop on Description Logics, 2007.
[73] S. Wandelt, R. Möller, Towards abox modularization of semi-expressive description logics, Applied Ontology 7 (2) (2012) 133–167.
[74] K. Zeng, J. Yang, H. Wang, B. Shao, Z. Wang, A distributed graph engine for web scale RDF data, Proc. VLDB Endow. 6 (4) (2013) 265–276.
[75] O. Erling, I. Mikhailov, Virtuoso: RDF support in a native RDBMS, in: Semantic Web Information Management, 2010.
[76] A. Harth, J. Umbrich, A. Hogan, S. Decker, Yars2: A federated repository for querying graph structured data from the web, in: Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference, 2007.
[77] R. Harbi, I. Abdelaziz, P. Kalnis, N. Mamoulis, Evaluating SPARQL queries on massive RDF datasets, Proc. VLDB Endow. 8 (12) (2015) 1848–1851.
[78] A. Potter, B. Motik, I. Horrocks, Querying distributed RDF graphs: The effects of partitioning, in: Proceedings of the 10th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with the 13th International Semantic Web Conference, 2014.
[79] K. Hose, R. Schenkel, WARP: Workload-aware replication and partitioning for RDF, in: Proceedings of the 4th International Workshop on Data Engineering Meets the Semantic Web, 2013.
[80] J. Huang, D.J. Abadi, K. Ren, Scalable SPARQL querying of large RDF graphs, Proc. VLDB Endow. 4 (11) (2011) 1123–1134.
[81] S. Yang, X. Yan, B. Zong, A. Khan, Towards effective partition management for large graphs, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012.
[82] K. Lee, L. Liu, Efficient data partitioning model for heterogeneous graphs in the cloud, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013.
[83] R. Rosati, A. Almatelli, Improving query answering over DL-Lite ontologies, in: Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning, 2010.
[84] H. Perez-Urbina, B. Motik, I. Horrocks, Tractable query answering and rewriting under description logic constraints, J. Appl. Logic. 8 (2) (2010) 186–209.
[85] A. Chortaras, D. Trivela, G. Stamou, Optimized query rewriting for OWL 2 QL, in: Proceedings of the 23rd International Conference on Automated Deduction, 2011.
[86] S. Kikot, R. Kontchakov, M. Zakharyaschev, Conjunctive query answering with OWL 2 QL, in: Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, 2012.
[87] T. Venetis, G. Stoilos, G.B. Stamou, Query extensions and incremental query rewriting for OWL 2 QL ontologies, J. Data Semantics 3 (1) (2014) 1–23.
[88] M. König, M. Leclère, M.L. Mugnier, M. Thomazo, A sound and complete backward chaining algorithm for existential rules, in: Proceedings of the 6th International Conference on Web Reasoning and Rule Systems, 2012.
[89] M. König, M. Leclère, M.L. Mugnier, M. Thomazo, On the exploration of the query rewriting space with existential rules, in: Proceedings of the 7th International Conference on Web Reasoning and Rule Systems, 2013.
[90] G. Gottlob, G. Orsi, A. Pieris, Query rewriting and optimization for ontological databases, ACM Trans. Database Syst. 39 (3) (2014) 25:1–25:46.
[91] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, M. Ruzzi, D. Fabio Savo, MASTRO: A reasoner for effective ontology-based data access, in: Proceedings of the 1st International Workshop on OWL Reasoner Evaluation, 2012.
[92] M. Thomazo, Compact rewritings for existential rules, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, 2013.
[93] G. Gottlob, G. Orsi, A. Pieris, Ontological queries: Rewriting and optimization, in: Proceedings of the 27th International Conference on Data Engineering, 2011.
[94] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, J. Web Semant. 14 (2012) 57–83.
[95] D. Bursztyn, F. Goasdoué, I. Manolescu, Teaching an RDBMS about ontological constraints, Proc. VLDB Endow. 9 (12) (2016) 1161–1172.
[96] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, K. Srinivas, The summary Abox: Cutting ontologies down to size, in: Proceedings of the International Semantic Web Conference, 2006.
[97] H. Paulheim, H. Stuckenschmidt, Fast approximate A-Box consistency checking using machine learning, in: Proceedings of the Extended Semantic Web Conference, 2016.