Stanchev, 2010. Published in *2010 Fifth International Conference on Digital Information Management (ICDIM)*, 384-391.

On Index Merging for Semantic Web Data

Lubomir Stanchev
Indiana University - Purdue University Fort Wayne
Fort Wayne, IN, USA
stanchel@ipfw.edu

Abstract

In this paper we explore the properties of description indices that store concept descriptions rather than plain data. Although these novel data structures are beneficial for efficiently answering semantic web queries, expressed in a language such as nRQL or SPARQL-DL, they take extra storage and their maintenance can become a performance bottleneck. In order to alleviate these shortcomings, we introduce a procedure for merging description indices. Experimental results over the LUBM benchmark show that this technique can result in economy of storage space, while the performance is slightly affected for a static workload and is improved for a dynamic workload.

1. Introduction

The Internet is a wonderful invention because it allows us to share knowledge from around the globe. However, searching for specific piece of information is often compared to searching for a needle in a haystack. Search engines index the Internet based on keywords. Unfortunately, this fails to capture the semantics of the data. The long awaited advent of the semantic web has been postponed due to the innate complexity of creating and efficiently accessing a general-purpose semantic knowledgebase (i.e., it is not limited to, for example, only the medical domain - see the the SNOMED ontology [1]). In this work we extend previous research ([17]) that explains how to index a knowledgebase that is expressed in OWL-DL. In particular, we show how *description indices*, which store concept descriptions rather than plain data, can be merged in order to save space. Since our index-merging technique reduces redundant knowledge among indices, our approach can also lead to improved performance for update-intensive workloads because the size of the index structures that need to be maintained will be smaller.

Description indices make the problem of efficient query answering over a knowledgebase scalable. The major downside of description indices is that they take extra storage space and that they introduce maintenance overhead. In this paper we address these shortcomings by presenting a procedure for merging description indices. This is an important research area because index merging can reduce the performance and space penalty for creating description indices and can therefore make them a more viable option for integration in existing knowledgebase query engines, such as Pellet ([16]) and Racer ([6]).

Efficient query answering over semantic web data is a challenging problem because the information inside a knowledgebase can be imprecise and additional knowledge that is stored in a terminology may be needed to interpret it. The research in [17] shows how to create description indices that are specific to the input queries. However, merging description indices is a difficult problem because, except for the most trivial cases, the duplicate eliminating union of the data of several description indices cannot be stored in a description index and still efficiently retrieved (this is also the case for traditional indices - [5, 4]). We overcome this barrier by adding *marker bits* to description indices. Marker bits are beneficial because they allow for efficient search that is restricted to the elements of a description index that are subsumed by a predefined concept description and, as a consequence, our index-merging technique can be applied in more scenarios.

Existing techniques for optimizing queries over knowledgebases include realization (see for example the Racer approach [7]) and simplification (see for example [3]). The first technique precomputes the individuals that are associated with every concept. Although this technique improves performance, it does not take into account the type of input queries and therefore it does not create query-specific indices. The second technique simplifies the problem by considering only an RDF knowledgebase rather than a full-blown OWL-DL knowledgebase. The problem of merging indices for knowledgebases has not been addressed before because the research in the area of creating indices for OWL-DL knowledgebases is relatively new.

An OWL-DL knowledgebase consists of *individuals* that

are described by concept descriptions. Unlike the relational database case, these descriptions do not need to be descriptive. For example, an individual can be described as being a department in the Art School without specifying the actual name of the department. A description index is implemented as a balanced search tree or a hash table that store the concept descriptions of individuals and potentially pointers to other data structure instances of the same type. Throughout the paper, we will refer to the individuals in the knowledgebase and their concept descriptions interchangeably.

In order to demonstrate our description index creation and merging technique, consider the following query that is expressed in the SQWRL language ([10]).

Professor($?x_1$) ^ *rank*($?x_1$,r) ^ Course($?x_2$) ^ Department($?x_3$) ^ *name*($?x_3$,$?n$) ^ *member of*($?x_1$,$?x_3$) ^ *belongs to*($?x_2$,$?x_3$) → sqwrl:select($?x_1$,$?x_2$,$?x_3$,$?n$,$?r$) ^ sqwrl:orderBy($?n$,$?r$)

The query asks for all departments and the courses and professors inside the department, where the departments are ordered relative to *name* and the professors inside a department are ordered relative to *rank*. We assume that every department has a unique name.

Consider a description index that is implemented by a search tree that contains the concept descriptions of the individuals that can be deduced to be departments ordered by the data property *name*. For each such individual, the index also stores the concept descriptions associated with the professor individuals in that department ordered by rank and the course individuals in the department. This description index can be used to answer the example query, where the level of efficiency is proportional to the descriptiveness of the knowledgebase. In contrast, such efficient access is not possible with existing knowledgebase query engine implementations, such as Pellet ([16]) and Racer ([6]). The reasons are that: (1) existing engines do not create query specific indices and (2) they do not explore novel index types that are required for supporting knowledgebase queries and index merging.

Next, consider a second description index that stores the concept descriptions for the the individuals that can be deduced to be organizations sorted again by the data property *name*. In addition, the description index stores for each such individual the list of secretary individuals in the organization. The list of individuals that can be deduced to be departments will be stored in both description indices because departments are organizations. In order to avoid this redundancy, the two indices can be merged into a single description index that stores a search tree of organization individuals. The index will also store the associated course and professor individuals for departments and the associated secretary individuals for organizations. In addition, we will mark a node in the organization search tree exactly when the node or one of its descendants contains an individual that can be inferred to be a department. The marker bits will allow us to efficiently search for all the department individuals in the search tree and therefore the merged index will yield similar performance characteristics for searches as the initial two description indices.

The major contributions of the paper are the introduction of marker bits for description indices in Chapter 3, the index-merging technique that is present in Chapter 4, and our experimental results that are described in Chapter 5. Related research is presented in Chapter 2, while Chapter 6 summarizes the paper.

2 Related Research

The Web Ontology Language - Description Logics (OWL-DL) [2] was introduced by the World Wide Web Consortium (W3C) as a standard for representing knowledgebases. While most existing research focuses on consistency checking using a tableau-based approach (Pellet [16], Fact++ [18], and Racer [6] are three popular implementations), there have been two notable proposal for SQL-like query languages over OWL-DL knowledgebases.

The first proposal is SPARQL-DL ([15]). The language is very expressive and the free variables can be not only individuals, but also concepts or relationships (i.e., the languages has capabilities that cannot be expressed in first-order logics). Notably, [14] describes how cost-best query optimization can be used to make SPARQL-DL query execution decisions, such as join-order selection. Currently, SPARQL-DL is supported by Pellet. The second language is nRQL and was created by the developers of Racer [8].

Papers, such as [7], explain how to use realization to precompute the elements of each concept in the terminology. Although this approach reduces the number of expensive calls to the reasoner, it can still lead to linear-time complexity for answering certain queries. The reason is the lack of appropriate indices.

Ground breaking work in the area includes the papers [12, 13, 17]. They describe how to create an index that stores concept descriptions rather than individuals. The concept descriptions in the index are ordered relative to inferred values for some of the data properties. While the first two papers describe how to create description indices that can answer queries that ask for the individuals of a single concept description, the third paper describes how to create description indices that can be used to efficiently answer queries that have free variables ranging over several concept description. In this paper we extend the definition of description indices with marker bits and explain how description indices can be merged.

Two notable papers that address the problem of index merging in the relational database case are [5, 4]. Our approach differs significantly from theirs. Unlike the two papers, our index merging algorithm produces indices that have similar performance to the input indices.

3 Description Indices

In this section we review and extend the definition of the terms: *description tree index* and *description hash table*, which are the building blocks of a description index (the two terms were initially introduced in [17]). The novelty is that marker bits are added to description tree indices. The two artifacts correspond to a search tree index and a hash table, respectively, in the relational database sense.

A description tree index stores concept descriptions in a predefined order and allows for efficient range search queries and for retrieving the result relative to the index order. This index type also allows for the efficient retrieval of the individuals that are subsumed by a predefined concept description using marker bits. A description hash table stores concept descriptions that are clustered relative to inferred values of one or more properties and supports the efficient answering of partial-match queries. Our running example will use OWL-DL concept descriptions that are expressed using the Manchester OWL syntax([9]). For completeness, we next present example expressions in this language, where the reader should refer to [2, 9] for a complete overview.

3.1 The Manchester OWL Syntax

Our running example is based on the terminology of the LUBM benchmark ([11]). Part of the terminology, expressed using the Manchester OWL syntax, is shown in Table 1, where bold is used for language keywords, while properties (both data and object) are in italic.

The first subsumption states that if someone is a full professor, then they are a professor and they work for at least one organization. The second rule states that if someone is a graduate student, then they are a student and all their advisors are professors. The third rule states that undergraduate students must take between two and four courses. The last rule states that if someone is a student, then they must be either an undergraduate or graduate student.

Note that the first column of the table contains primitive concepts, while the second column contains *concept descriptions* (a primite concept is a special case of a concept description). A concept description describes all individuals that satisfy certain rules. For example, the concept description in the right column of the first row in Table 1 describes all professors that work for an organization. The relationship between the two columns of the table is *subsumption* (denoted as **SubClassOf**), that is, if an individual belongs to the primitive concept shown in the left column, then it must also belong to the set of individuals described by the concept description in the right column.

A set of subsumption rules form a *terminology* (i.e., the additional knowledge that is used when answering queries over a knowledgebase). The Manchester OWL Syntax is also used to describe the *individuals* in the knowledgebase. For example, a particular student may be described as follows.

is taking Calculus 101 **and** *is taking* **value** Stats 101 **and** *member of* Computer Science Department **and** *age* **value** $[>= 20]$

This student takes Calculus 101 and Statistics 101, they are member of the Computer Science Department, and they are at least twenty years of age.

3.2 Description Tree Index

A formal definition of a description tree index follows.

Definition 1 (description tree index) *A description tree index has the syntax $\langle\{C_1, \ldots, C_c\}, C, O\rangle$, where C and $\{C_i\}_{i=1}^{c}$ are concept descriptions and O is an ordering description. The index stores all individuals e that have concept description D for which it can be inferred from the terminology $\mathcal{T}$ that C subsumes D (i.e. $\mathcal{T}$ implies D **SubClassOf** C). The ordering description O defines the order of the elements in the search tree. The i^{th} bit of a node in the search tree will be marked ($1 \leq i \leq c$) exactly when the node or one of its descendants contains an individual e with concept description D for which it can be inferred from the terminology $\mathcal{T}$ that C_i subsumes D (i.e. $\mathcal{T}$ implies D **SubClassOf** C_i).*

Note that the marker bit part will be omitted from the syntax of a description tree index when empty. The definition relies on a way for defining order among concept descriptions, which is presented next.

Definition 2 (ordering description) *An ordering description Od has the following syntax.*

$$Od ::= \langle\rangle \mid \langle f \texttt{ dir} : Od\rangle \mid \langle D_1 : Od_1, \ldots, D_m : Od_m\rangle$$

We have used $\langle\rangle$ to denote an empty ordering, `dir` *to denote* `asc` *or* `desc`*, D to denote a concept description, and f to denote a data property. In order for this ordering description to be valid relative to a terminology, it must be the case that "D_i* **and** *D_j* **SubClassOf EMPTY***" for $1 \leq i \neq j \leq m$ and "D* **SubClassOf** *D_1* **or** *$\ldots$* **or** *D_m*"

(*concept*)	(*subsumed by*)
Full Professor	Professor **and** (*works for* **some** Organization)
Graduate Student	Student **and** (*advisor* **only** Professor)
Undergraduate Student	Student **and** (*takes course* **min** 2) **and** (*takes course* **max** 4)
Student	Undergraduate Student **or** Graduate Student

Table 1. Part of the LUBM Schema Expressed Using the Manchester OWL Syntax

*are both implied by the terminology, where D is a concept description that describes the set of individuals on which the order is defined (***EMPTY** *is used to denote the empty concept - i.e., bottom in description logics). We will say that the concept description D is before E relative to an ordering description Od and terminology $\mathcal{T}$ and write $D \prec_{Od,\mathcal{T}} E$ in the following cases (x^* is used to denote the renaming of x to x^*):*

- *when Od is $\langle f$ dir : $Od_1\rangle$:*
  - *$\mathcal{T} \cup \mathcal{T}^*$ implies (D **and** E^*) **SubClassOf** ($f < f^*$) when* dir=asc *and (D **and** E^*) **SubClassOf** ($f > f^*$) when* dir=desc *or*
  - *$\mathcal{T} \cup \mathcal{T}^*$ implies ((D **and** E^*) **SubClassOf** ($f = f^*$)) **and** $D \prec_{Od_1,\mathcal{T}} E$*
- *when Od is $\langle D_1 : Od_1, \ldots, D_m : Od_m\rangle$:*
  - *$\mathcal{T}$ implies (D **SubClassOf** D_i) **and** (E **SubClassOf** D_j) for some $i < j$ or*
  - *$\mathcal{T}$ implies (D **or** E) **SubClassOf** D_i for some i and $D \prec_{Od_i,\mathcal{T}} E$.*

Note that we will omit empty ordering descriptions from an ordering description. Similarly, when dir is equal to asc, it may be omitted. Note that if f and g are data properties, then $f = g$ ($f > g$) represents the set of individuals that have the same value for the two properties (for which the value for the f property is bigger than the value for the g property).

An example of a description tree index is: $\langle\{$Full Time$\}$, Professor, $\langle$*worksFor.name*: $\langle$Full Professor : age, Associate Professor : *salary*, Assistant Professor: *publications*$\rangle\rangle\rangle$. It denotes a search tree that contains all individuals that can be inferred to be professors. The ordering will be first relative to the name of the place of employment, which we assume to be unique. In the same workplace, full professors will come first, followed by associate and assistant professors. Full professors in the same workplace will be ordered relative to age, associate professors - relative to salary, and assistant professors - relative to the number of publications. If a node in the search tree or one of its descendants contains a concept description that can be deduced to be subsumed by the concept "Full Time", then the node will be marked. The marker bits will allow one to efficiently access the full-time professors in the index.

Consider a query that asks for all individuals that represent full time associate professors that work for a department with a given name (we will denote this name as $:P$) ordered by salary. In order to answer the query, we can perform a search in the description tree index, where for simplicity we assume that the search tree is binary. For every node with concept description D, we need to compare the concept descriptions D and "Associate Professor **and** *worksFor.name*= $:P$" relative to the ordering description of the description tree index. If D comes first, then we need to continue the search in the right subtree if its root node is marked. If D comes second, then we need to continue the search in the left subtree if its root node is marked. If the two concept descriptions are not comparable relative to the ordering description, then we need to check if the terminology implies "D **SubClassOf** (Associate Professor **and** *worksFor.name*= $:P$)". If this is the case, then we have found a concept description that belongs to the query result. If the two concept descriptions are not comparable and the terminology does not imply this subsumption, then we need to check the subtrees (potentially both left and right) that have marked root nodes. The reason is that, in general, an ordering description defines a partial order.

The search will prune out subtrees that have a root node that is not marked. The search will be efficient (i.e., will return each element of the query result in logarithmic time), when the following *descriptive sufficiency* property ([12]) holds for all concept descriptions in the description tree index. Note that the descriptive sufficiency property holds for the data that is generated from the LUBM benchmark.

Definition 3 (descriptive sufficiency) *A concept description D is sufficiently descriptive relative to an ordering description Od and terminology $\mathcal{T}$, written as $DS_{Od,\mathcal{T}}(D)$, if one of the following holds.*

- $Od = \langle\rangle$
- *$Od = \langle f$ dir : $Od_1\rangle$, $DS_{Od_1,\mathcal{T}}(D)$, and $\mathcal{T}$ implies D **SubClassOf** $f = k$ for some constant k*
- *$Od = \langle D_1 : Od_1, \ldots, D_m : Od_m\rangle$, $DS_{Od_i,\mathcal{T}}(D)$, and $\mathcal{T}$ implies "D **SubClassOf** D_i" for some i*

3.3 Description Hash Table

The general definition of a description hash table follows.

Definition 4 (description hash table) *A description hash table H has the syntax $\langle C, \{p_i\}_{i=1}^{k} \rangle$, where C is a concept description and $\{p\}_{i=1}^{k}$ are properties. It partitions all concepts descriptions that are subsumed by C using a hash function on the different values of $\{p\}_{i=1}^{k}$.*

Note that the second element of the syntax of a description hash table may be empty, in which case we will store all concept descriptions in a single bucket, that is, the description hash table will degenerate into a *description list*. When the second element is empty, we will omit it from the syntax.

An example description hash table is: $\langle$Graduate Student **or** Undergraduate Student, $\{advisor\}\rangle$. It partitions the graduate and undergraduate students relative to their advisor. If the advisor of a student is not known, than this student could be placed in several hash buckets. For example, if the advisor of a student that is in the Computer Science Department is not known, but we know that students in the Computer Science department are supervised by professor from the Computer Science Department, then we only need to store the student in the hash buckets for Computer Science professors.

Searching in a description hash table is similar to searching in a regular hash table in the sense that the clustering allows us to prune out buckets that do not contribute the query result. Since the same individual can appear in multiple hash buckets, search efficiency will depend on the level of descriptiveness of the concept descriptions that are associated with the individuals in the hash table.

3.4 Description Index

The definition of a description index follows.

Definition 5 (description index) *A description index can be represented as a rooted tree, which we will call the* definition tree *of the description index. The nodes in the tree can be either* hash nodes *or* tree nodes*. The label of a hash node is that of a description hash table, while the label of a tree node is that of a description tree index. The edges in the definition tree have labels that are object properties.*

A description index represents a set of description hash tables and description tree indices, where the connection between the data structures and their content is determined by the shape and labels of the definition tree.

Specifically, for a root node in the definition tree that is a hash node or a tree node, the corresponding description hash table or description tree index, respectively, is built. If the node n_1 is the child of the node n_2 and the edge between the two nodes has the label p, then a data structure is built for each concept description D stored in the data structure for n_2. Specifically, if n_1 is the tree node $\langle \{C_i\}_{i=1}^{c}, C, O \rangle$, then the description tree index $\langle \{C_i\}_{i=1}^{c}, C$ **and** *$(p$* **some** *$D), O\rangle$ is created. Alternatively, if n_1 is the hash node $\langle C, \{p_i\}_{i=1}^{k} \rangle$, then the description hash table $\langle C$* `and` *$(p$* `some` *$D), \{p_i\}_{i=1}^{k}\rangle$ is created.*

The definition trees of the two example description indices that were merged in Section 1 and the description tree for the resulting description index are shown in Figure 1.

4 Description Index Merging

The algorithm for creating description indices is presented in [17]. Algorithm 1 shows our description index merging algorithm.

Line 2 of the algorithm finds a clustering of the indices, where the indices in every cluster have a common prefix. Two description indices have a common prefix when their root nodes share a common prefix. Two tree nodes share a common prefix if their ordering descriptions start with the same property. A hash node shares a common prefix with a tree node if it contains the first property of the ordering description of the tree node in its set of properties. Two hash nodes share a common prefix it they share a common property in their set of properties. The reasoning behind this definition is that the order of the properties is meaningful for an ordering description, but not for the properties of a description hash table. Description indices with root nodes that are index nodes with empty ordering description and description indices with root nodes that are description lists can also be clustered together. In this case the common prefix will be the empty set. Note that there are different ways to do the partitioning and the chosen algorithm depends on the optimization function. In our experimental evaluation we chose the partitioning that minimizes the size of the produced description indices after the merging.

Next, Lines 3-31 of the pseudo-code process every cluster of indices. The common prefix property is identified in Line 6. Line 12 computes the concept description for the root node of the description index that is constructed. It will be the union of the concept descriptions of the root nodes of the input concept descriptions. Lines 13-22 compute whether the root node of the resulting index will be a hash node or a tree node and calculate what marker bits should be added. The method CREATEINDEX, which is called in Lines 24 and 26, constructs a description index. The root node of the description index is formed from all but the last parameter. The last parameter describes the child nodes. Line 24 creates a hash nodes, while Line 26 creates a tree node.

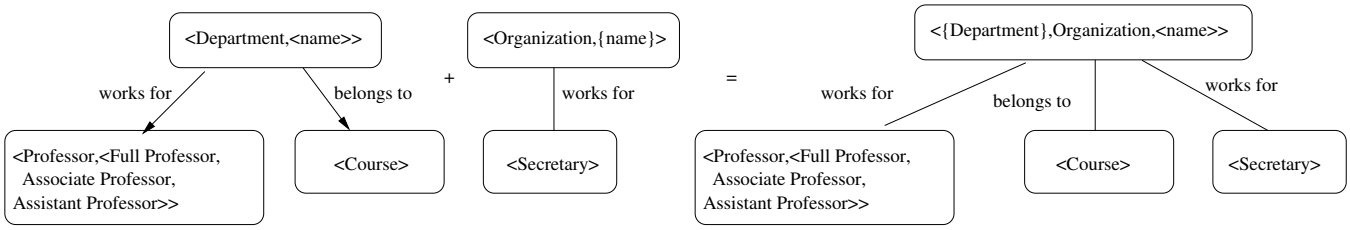

Figure 1. Merging Example.

Algorithm 1 MERGE(indices $\bar{X}$)

1: *result*$\leftarrow \emptyset$
2: $\{\bar{X_i}'\}_{i=1}^{a} \leftarrow$ FINDCLUSTERING($\bar{X}$)
3: **for** $i \leftarrow 1$ **to** a **do**
4: $\quad \{X'_j\}_{j=1}^{k} \leftarrow \bar{X_i}'$
5: $\quad$ **if** $k > 1$ **then**
6: $\quad\quad A \leftarrow$ FINDPREFIX($\{X'_j\}_{j=1}^{k}$)
7: $\quad\quad \{n_j\}_{j=1}^{m} \leftarrow$ root nodes of $\{X'_j\}_{j=1}^{k}$
8: $\quad\quad \{C_j\}_{j=1}^{m} \leftarrow$ concept descriptions in $\{n_j\}_{j=1}^{m}$
9: $\quad\quad$ *isHash*$\leftarrow$ **true**
10: $\quad\quad \bar{C} \leftarrow \emptyset$
11: $\quad\quad \bar{X}' \leftarrow \emptyset$
12: $\quad\quad D \leftarrow C_1$ `or` $\ldots$ `or` C_m
13: $\quad\quad$ **for** $j \leftarrow 1$ **to** m **do**
14: $\quad\quad\quad$ **if** type(n_j) $\neq$ hash **then**
15: $\quad\quad\quad\quad$ *isHash* $\leftarrow$ **false**
16: $\quad\quad\quad$ **end if**
17: $\quad\quad\quad$ **if not** (D **SubClassOf** C_j) **then**
18: $\quad\quad\quad\quad \bar{C} \leftarrow \bar{C} \cup \{C_j\}$
19: $\quad\quad\quad\quad$ *isHash* $\leftarrow$ **false**
20: $\quad\quad\quad$ **end if**
21: $\quad\quad\quad \bar{X}' \leftarrow \bar{X}' \cup$ *remove*(X'_j,A)
22: $\quad\quad$ **end for**
23: $\quad\quad$ **if** *isHash* **then**
24: $\quad\quad\quad$ *result* $\leftarrow$ *result* $\cup$ CREATEINDEX (D, $\{A\}$,MERGE($\bar{X}'$))
25: $\quad\quad$ **else**
26: $\quad\quad\quad$ *result* $\leftarrow$ *result* $\cup$ CREATEINDEX ($\bar{C}$,D,$\{A\}$, MERGE($\bar{X}'$))
27: $\quad\quad$ **end if**
28: $\quad$ **else**
29: $\quad\quad$ *result* $\leftarrow$ *result* $\cup \bar{X_i}'$
30: $\quad$ **end if**
31: **end for**
32: **return** *result*

The presented description index merging algorithm is correct in the sense that the created index has the same query answering capabilities as the initial indices. The algorithm is not complete and improving the quality of the result it produces (e.g., exploring more description index merging opportunities) is an area for future research.

5 Experimental Results

Our experimental results are based on the LUBM benchmark ([11]). We created a database that consists of twenty universities. The benchmark contains a program that generated data about each university (e.g., departments, students, professors, courses, etc.). The benchmark consists of fourteen queries. We executed each of the queries in three different modes. One way is using the description index structures that are described in [17]. This approach creates description indices that do not support marker bits and no index merging is applied. The second way is using merged indices, where indices were manually merged using the algorithm that is described in this paper. The third way is by rewriting the queries in SQARQL-DL and executing them through the Jena interface using the Pellet reasoner. In all cases, the heap was set to 1.5GB to avoid an out-of-space error. All tests were executed on Sony VGN-NW150J laptop with 4GB of main memory and 2.10GHz CPU. The results of the experiments, averaged over one thousand runs, are shown in Table 2.

We measured the time to compute and print the result, where the time to initially load the data was not measured. For example, Query 6 runs slowly in all three scenarios because it asks for all the students and printing all students is time consuming. Conversely, Query 2 asks for all graduate students that are students in the university from which they got their undergraduate degree. Our approach answers the query by performing an index join, which is significantly faster than the nested-loop join that is performed by Pellet.

Note that Query 14 asks for the undergraduate students, while Query 6 asks for all students. The single description index $\langle$\{Undergraduate Student\}, Student,\{\}$\rangle$ will be created to answer both queries. Since the index for all students will be bigger than the index for undergraduate students, it will take more time to use the new index to enumerate all undergraduate students, which is the reason for the increased time for executing Query 14 when indices are

(*query*)	(*description index*)	(*merging*)	(*Pellet*)
Q1	1	1	1076
Q2	47	47	3167
Q3	1	1	1263
Q4	1	1	1373
Q5	1	1	796
Q6	671	671	3619
Q7	1	1	1373
Q8	35	35	1966
Q9	51	51	4712
Q10	520	520	1404
Q11	31	31	1186
Q12	1	1	3213
Q13	35	35	858
Q14	468	482	2293

Table 2. Execution time for the LUBM queries in milliseconds

(*query*)	(*index size*)
Q1	5
Q2	4
Q3	3
Q4	3
Q5	14
Q6	13
Q7	14
Q8	21
Q9	15
Q10	14
Q11	1
Q12	1
Q13	14
Q14	10

Table 3. Size of indices for each query in MBs

merged.

We realize that there is a price to pay for creating indices. The obvious cost is the cost of storage. The size of the initial knowledgebase of twenty universities is 110MB. The amount of storage needed to store the indices for each query is shown in Table 3. As expected, indices bring slight storage overhead. If index merging is used, then this total size of the description indices is reduced from 132MB to 95MB. The reason is that we can eliminate storing redundant copies of university concept descriptions, department concept descriptions, professor concept descriptions, and student concept descriptions as a result of the merge.

Another benefit of index merging is reduced update cost. We introduced to the LUBM benchmark operations that add a student, remove a student, move a professor between departments, and change the professor that teaches a course. The update to be performed was chosen randomly and two thousand updates were performed. The results were averaged over one thousand runs. When no merging was performed on the indices that were created for the fourteen queries, it took 63 milliseconds on average to perform the 2000 updates. However, when index merging was introduced, the update time was reduced to 47 milliseconds.

6 Conclusion

Description indices are beneficial for efficient access to knowledgebases. In the paper we presented a merging technique that reduces the storage and update cost of description indices. Although the presented merging technique is not complete, our experimental results show that it can lead to increased performance and reduced storage cost. An area for future research is exploring more opportunities for description index merging.

References

[1] International Health Terminology Standards Development Organization. *http://www.ihtsdo.org/*.

[2] OWL Web Ontology Language Guide. *http://www.w3.org/TR/owl-guide/*.

[3] RDF-3X. *http://www.w3.org/RDF*.

[4] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. *SIGMOD 2005*, pages 227–238, 2005.

[5] Chaudhuri and Narasayya. Index Merging. *ICDE*, pages 296–303, 1999.

[6] V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. *Second International Workshop on Evaluation of Ontology-based Tools*, 2003.

[7] V. Haarslev and R. Möller. Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results (2004). *In Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 2004.

[8] V. Haarslev, R. Möller, and M. Wessel. Querying the semantic web with Racer + nRQL. *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics*, 2004.

[9] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The Manchester OWL syntax. *OWL: Experiences and Directions*, 2006.

[10] M. J. O'Connor and A. K. Das. Sqwrl: A query language for owl. *OWL: Experiences and Directions (OWLED), Fifth International Workshop*, 529, 2008.

[11] Z. Pan, Y. Guo, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[12] J. Pound, L. Stanchev, D. Toman, and G. E. Weddell. On ordering descriptions in a description logic. *International Workshop on Description Logics*, 2007.

[13] J. Pound, L. Stanchev, D. Toman, and G. E. Weddell. On Ordering and Indexing Metadata for the Semantic Web. *International Workshop on Description Logics*, 2008.

[14] E. Sirin and B. Parsia. Optimizations for Answering Conjunctive ABox Queries: First Results. *In Proceeding of the International Description Logics Workshop*, 2006.

[15] E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. *3rd OWL: Experiences and Directions Workshop (OWLED)*, 2007.

[16] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2), 2007.

[17] L. Stanchev and G. Weddell. On Building an Index Advisor for Semantic Web Queries. *Sixth International Conference on Formal Ontology in Information Systems*, 2010.

[18] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. *Lecture Notes in Computer Science*, 4130:292–297, 2006.