Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results

Volker Haarslev[†] and Ralf Möller[‡]

[†]Concordia University, Montreal
[‡]Technical University Hamburg-Harburg

Abstract

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research even in distributed environments. In particular, the often-discussed semantic web initiative is based on description logics (DLs) and defines important challenges for current system implementations. Recently, several standards for representation languages have been proposed (RDF, OWL). By introducing optimization techniques for inference algorithms we demonstrate that sound and complete query engines for semantic web representation languages can be built for practically significant query classes. The paper introduces and evaluates optimization techniques for tableau-based instance retrieval algorithms for the description logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$, which covers large parts of OWL. The paper discusses practical experiments with the description logic system RACER.

1 Introduction

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research. In particular, the semantic web initiative [6] is based on description logics (DLs) and defines important challenges for current system implementations. Recently, one of the main standards for the semantic web has been proposed: the Web Ontology Language (OWL) [24]. OWL is based on two other standards: Resource Description Format (RDF [19]) and its corresponding "vocabulary language" RDF Schema (RDFS) [9]. In recent research efforts, these languages are mainly considered as ontology representation languages (see e.g. [1] for an overview). The languages are used for defining classes of so-called abstract objects. Now, many applications start to use the RDF part of OWL for representing information about specific abstract objects of a certain domain. Graphical editors such as OilEd [4] or Protégé [23] support this way of using OWL quite well.

All information about specific objects (or entities) refers to an ontology (expressed in OWL). Thus, in contrast to, for instance, simple relational databases, queries for retrieving abstract objects described in RDF documents have to be answered w.r.t. to a conceptual domain model (the ontology). The paper introduces and evaluates optimization techniques for tableau-based instance retrieval algorithms for the logical basis of OWL DL, the description logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$ [17, 13], and discusses practical experiments with the description logic system RACER. By introducing optimization techniques for tableau-based inference algorithms we demonstrate that sound and complete query engines for semantic web representation languages can be built for practically significant query classes. The paper is aimed at semantic web systems developers interested in applying and implementing sound and complete knowledge representation and reasoning technologies. Note that we consider soundness and completeness as very important because incompleteness of instance retrieval can even result in unsoundness if the results are used for higher-level purposes in an unreflected way.

The paper presupposes only basic knowledge about description logics, which can be easily acquired from introductory textbooks (see [2]). A few words about the relationship of description logics, semantic web representation languages, and systems such as RACER are appropriate, however.

RACER reads OWL ontology documents from web servers and represents ontology information as a so-called T-box. T-boxes contain so-called generalized concept inclusions (GCIs). For details about description logic syntax and semantics see, e.g., [2]. RACER accepts the so-called OWL DL subset [24] with some minimal restrictions such as approximated reasoning for nominals, no full number restrictions for datatype properties, and unique name assumption (see [12] for details). DAML+OIL documents are

interpreted with the same restrictions as manifested in OWL DL [24] (the sets of classes and instances are disjoint, no reified statements, no treatment of class metaobjects etc.). For the results presented in the paper, these restrictions are of no importance.

Descriptions in RDF documents (with OWL DL restrictions) are represented as A-boxes managed by the RACER System (for details see the RACER User's Guide [12]). Basically, the instance retrieval problem for a query concept C_q and an A-box A can be implemented as a sequence of instance tests for all individuals that are mentioned in an A-box. An instance test verifies that an individual i is in the extension of a certain concept in all models of a given T-box and A-box. Retrieving resources described in OWL/RDF documents can be implemented using the A-box instance retrieval inference service [15]. In this paper we discuss a restricted version of conjunctive queries [18]. An introduction to an XML-based query syntax is given in [5].

The contribution of the paper is twofold. By introducing and analyzing practical algorithms tested in one of the mature, sound, and complete description logic systems, which is used in many research projects all over the world, the development of even more powerful semantic web query engines is directly supported. We also characterize the research frontier in order to also stimulate theoretical research for providing the basis for upcoming future system implementations. All example knowledge-bases we discuss in this paper can be downloaded for verification and comparison purposes (see the RACER download page).

2 Research Approach, Test Data, and Benchmarks

For implementing sound and complete inference algorithms, tableau-based algorithms are known to provide a powerful basis. Nowadays, almost all practical systems for $\mathcal{SHIQ}(\mathcal{D}_n)^-$ employ highly optimized versions of tableau-based algorithms. It should be emphasized that the research approach behind RACER is oriented towards applications. Thus, we start with optimization techniques for application-specific knowledge bases in order to evaluate optimization techniques in the context of instance retrieval problems. In particular, we consider applications for which the full power of A-box reasoning is actually required. We are aware of other approaches in which the use of logical reasoning for computing queries for other formalisms seems to be more appropriate (see [7], [8], [20]).

2.1 GameKB – An Application-Specific Knowledge Base

For instance, in [10, 11] a case-study with the application of DL inference services in a natural language (NL) interpretation system is presented. In particular, the instance retrieval service is investigated for various application-specific subtasks (e.g., resolution

of referring expressions, content determination, and content realization). In this application, many A-boxes are generated on the fly (see [10, 11] for details) and for each A-box a specific instance retrieval query is computed. In order to achieve good performance in the NL application, the performance of the instance retrieval procedure provided by the DL system is crucial. Furthermore, since A-boxes change quite frequently, standard techniques for optimizing instance retrieval using indexing techniques (see below for an explanation) can hardly be employed in order to improve performance because of the overhead of computing index structures in beforehand.

The T-box consists of 165 possibly cyclic GCIs for concepts as well as domain and range restrictions for 18 roles. In the T-box, many sufficient conditions for concept names are given (with appropriate GCIs, see also declarations with sameConceptAs in OWL). In the A-box around 250 individuals are mentioned in concept and role assertions. The DL used in the knowledge base is a subset of OWL DL (actually, $\mathcal{ALC}$ with inverse roles [2]).

2.2 Synthetic Knowledge Bases for Testing Behavior on Mass Data

For evaluating specific aspects of DL inference engines, a set of benchmarks containing synthetically generated KBs was developed [21]. In this paper we consider some of these tests, which are generated automatically due to different strategies. The tests consist of a so-called symmetric concept tree of depth d and branching factor b. For each concept n instances are declared. The instance retrieval query refers to a concept name at the first layer (one of the children of *top*). The second kind of test is similar to the first one but also declares relations between the individuals. An individual is set into relation to a previously generated one via a so-called role assertion [2]. Only one role is used.

The following discussion about optimization techniques starts with insight gained from application-knowledge bases. Later on we use some synthetically generated benchmarks to shed additional light on the behavior of the techniques proposed.

3 Optimization Techniques and their Evaluation

For applications, which generate A-boxes on the fly as part of their problem-solving processes and ask a few queries w.r.t. each A-box, computing index-structures (with a process called "realization", see below) is not worth the effort. In this section we discuss answering strategies for this kind of application scenario. As we will see, the techniques can also be exploited if index structures are to be computed (possibly off-line).

If an A-box contains individuals that are not "connected" by role assertions (or by constraints involving concrete domains), RACER computes so-called subset A-boxes representing these "islands", applies the algorithms described below to each subset,

and combines the results. We do not mention this kind of processing explicitly in the following subsections.

3.1 Optimized Linear Instance Retrieval

One possible alternative is to consider one individual at a time. Hence, the procedure $instance\_retrieval(C_q, A)$ can be implemented by using the following procedure call: $linear\_instance\_retrieval(C_q, contract(i, A), individuals(A))$ where $individuals(A)$ returns the set of individuals mentioned in the A-box A and the function *contract* computes a transformation of an A-box w.r.t. an individual. The idea is to transform tree-like role assertions "starting" from the individual i into equisatisfiable concept assertions with existential restrictions (see [14] for details). The reason is that in RACER, caching (see also [14]) is more effective for concepts rather than for A-box role assertions. Contracting an A-box is part of the processes for building internal data structures for A-box reasoning algorithms.

We assume that $ASAT$ is the standard A-box satisfiability test implemented as an optimized tableau calculus [17, 13]. The function $linear\_instance\_retrieval$ is then implemented as follows.

Algorithm 1 $linear\_instance\_retrieval(C, A, candidates)$:

```
result := {}
for all ind ∈ candidates do
  if instance?(ind, C, A) then
    result := result ∪ {ind}
return result
```

The function call $instance?(i, C, A)$ could be implemented as $\neg ASAT(A \cup \{i : \neg C\})$. However, although this implementation of *instance*? is sound and complete, it is quite inefficient. A faster variant uses sound but incomplete initial tests for detecting "obvious" non-instances: the individual model merging test (see [16]) and a subsumption test involving the negation of the query concept (see Algorithm 2).

Algorithm 2 $obvious\_non\_instance?(i, C, A)$:

```
return individual_model_merging_possible?(i, A, negated_concept(C))
     ∨ subsumes?(negated_concept(C), individual_concept(i))
```

The main idea of the individual model merging is to extract a (pseudo) model for an individual i from a completion of the A-box A. If the individual model of i and the (pseudo) model of $\neg C$ do not "interact" [16], i can easily be shown not to be an instance of C. If one of the "guards" returns *true*, the result of *instance*? is $false$.

Otherwise, an "expensive" instance test using the tableau algorithm is performed. The function *negated_concept* returns the negation of its input concept whereas the function *individual_concept* returns the conjunction of the concepts in all A-box concept assertions for an individual. For role assertions found in an A-box, we assume additional concept assertions. Role assertions for a role R with i on the lefthand side are represented by at-least terms and, depending on the number of different role assertions for i, corresponding conjuncts $(\geq n\, R)$ are generated by *individual_concept*. With these auxiliaries, the function *instance*? can be optimized for the average case but is still sound and complete.

Algorithm 3 $instance?(i, C, A)$:

```
if obvious_non_instance?(i, C, A) then
  return false
else
  return ¬ASAT(A ∪ {i : ¬C})
```

Although this variant of *instance*? is significantly faster (mainly due to the individual model merging guard), in the Game application discussed above, query answering times in the range of 20 seconds were still unacceptable. Although for many queries the result consists of a set of only very few individuals (compared to 250 individuals mentioned in the A-box) around a hundred individuals still cause the "expensive" $ASAT$ test to be invoked, regardless of the "guards" in Algorithm 3. Thus, although each $ASAT$ test is quite fast (200 milliseconds), its number should be further reduced in order to provide adequate performance.

3.2 Binary Instance Retrieval

How can A-box satisfiability tests be avoided at all? The observation is that only very few additions to A of the kind $\{i : \neg C\}$ lead to an inconsistency in the function *instance*? (i.e., in very few situations i is indeed an instance of C). Therefore, in many realistic scenarios the following procedure was found to be advantageous.

Algorithm 4 $binary\_instance\_retrieval(C, A, candidates)$:

```
if candidates = ∅ then
  return ∅
else
  (partition1, partition2) := partition(candidates)
  return partition_instance_retrieval(C, A, partition1, partition2)
```

We assume now that $instance\_retrieval(C_q, A)$ is implemented by calling the procedure $binary\_instance\_retrieval(C_q, contract(i, A), individuals(A))$. The function *partition*

is defined in Algorithm 5, it divides a set into two partitions. Given the partitions, *binary_instance_retrieval* calls the function *partition_instance_retrieval*. The idea of *partition_instance_retrieval* (see Algorithm 7) is to first check whether *none* of the individuals in a partition is an instance of the query concept C. This is done with the function *non_instances*? (see Algorithm 6).

Algorithm 5 *partition*(s): /* $s[i]$ refers to the i^{th} element of the set s */

if $|s| \leq 1$ **then**
  return $(s, \emptyset)$
else
  return $(\{s[1], \ldots, s[\lfloor n/2 \rfloor]\}, \{s[\lfloor n/2 \rfloor + 1], \ldots, s[n]\})$

Algorithm 6 *non_instances*?($cands, C, A$):

return $ASAT(A \cup \{i : \neg C \mid i \in cands \wedge \neg obvious\_non\_instance?(i, C, A)\})$

The evaluation we conducted with the natural language application indicates that for instance retrieval queries which return only very few individuals a performance gain of up to a factor of 5-10 can be achieved with binary search (compared to linear instance retrieval). The reason is that the *non_instances*? test is successful in many cases. Hence, with one "expensive" A-box test a large set of candidates can be eliminated. The underlying assumption is that, in general, the computational costs of checking whether an A-box $(A \cup \{i : \neg C, j : \neg C, \ldots\})$ is consistent is largely dominated by A alone. Hence, it is assumed that the size of the set of constraints added to A has only a limited influence on the runtime. For knowledge bases with, for instance, cyclic GCIs, this may not be the case, however.

Algorithm 7 *partition_instance_retrieval*($C, A, partition1, partition2$):

if $|partition1| = 1$ **then**
  $\{i\} = partition1$
  if *instance*?(i, C, A) **then**
    return $\{i\} \cup binary\_instance\_retrieval(C, A, partition2)$
  else
    return $binary\_instance\_retrieval(C, A, partition2)$
else if *non_instances*?$(partition1, C, A)$ **then**
  return $binary\_instance\_retrieval(C, A, partition2)$
else if *non_instances*?$(partition2, C, A)$ **then**
  return $binary\_instance\_retrieval(C, A, partition1)$
else
  return $binary\_instance\_retrieval(C, A, partition1)$
    $\cup\ binary\_instance\_retrieval(C, A, partition2)$

3.3 Dependency-based Instance Retrieval

Although *binary_instance_retrieval* is found to be faster in the average case, one can do better. If the function *non_instances*? returns $false$, one can analyze the dependencies of the tableau structures ("constraints") involved into all clashes of the tableau branches. If all clashes are due to an added constraint $i : \neg C$, then, as a by-product of the test, the individual i is known to be an instance of the query concept C. The individual can be eliminated from the set of candidates to be investigated, and it is definitely part of the solution set. Eliminating candidate individuals detected by dependency analysis prevents the reasoner from detecting the same clash over and over again until a partition of cardinality 1 is tested. In the example application, runtimes are reduced by another factor of 3 (compared to binary instance retrieval). If the solution set is large compared to the set of individuals in an A-box, there is some overhead compared to linear instance retrieval because only one individual is removed from the set of candidates at a time as well with the additional cost of collecting dependency information during the tableau proofs. In our investigations, dependency-based instance retrieval was always faster than binary instance retrieval.

3.4 Static Index-based Instance Retrieval

The techniques introduced in the previous section can also be exploited if indexing techniques are used for instance retrieval (see, e.g., [22, p. 108f.]). Basically, the idea is to reduce the set of candidates that have to be tested by computing the direct types of every individual. The direct types of an individual i are defined to be the most specific concept names (mentioned in a T-box) of which i is an instance. An index is constructed by deriving a function *associated_inds* defined for each concept name C mentioned in the T-box such that $i \in associated\_inds(C)$ iff $C \in direct\_types(i, A)$. Computing the direct types for each individual and the corresponding index *associated_inds* is also called *A-box realization*. The optimizations used in the RACER implementation are inspired by the marking and propagation techniques described in [3] for exploiting explicitly given information as much as possible.

The standard way to compute the index is to compute the direct types for each individual mentioned in the A-box separately (one-individual-at-a-time approach). In order to compute the direct types of individuals w.r.t. a T-box and an A-box, the T-box must be classified, i.e., for each concept name mentioned in the T-box (and the A-box) the most-specific subsumers (function *parents*) and least-specific subsumees (function *children*) are precomputed. Thus, *parents* and *children* are not really queries but just functions accessing results stored in data structures. Another view is that the children (or parents) relation defines a lattice whose nodes are concept names. The root node

is called *top*, the bottom node is called *bottom*. This lattice is also referred to as the "taxonomy".

In the following we assume that CN is the set of all concept names mentioned in the T-box (including the name *top*). Furthermore, it is assumed that the function $parents(C)$ returns the most specific subsumers of C whereas $descendants(C)$ ($ancestors(C)$) returns all subsumees (subsumers) of C including C. Subsumers and subsumees of a concept C are concept names from CN. The function $synonyms(C)$ returns all concept names from CN which are equivalent to C. Static index-based instance retrieval is implemented as follows (see [22, p. 108f.]).

Algorithm 8 $static\_index\_based\_instance\_retrieval(C, A)$:

if $\exists N \in CN : N \in synonyms(C)$ **then**
  return $\bigcup_{D \in descendants(C)} associated\_inds(D)$
else
  $known\_results := \bigcup_{D \in descendants(C)} associated\_inds(D)$
  $candidates := \bigcup_{P \in parents(C)} (\bigcup_{D \in descendants(P)} associated\_inds(D))$ (*)
  return $known\_results \cup instance\_retrieval(C, A, candidates \setminus known\_results)$

It is obvious that *instance_retrieval* can be implemented by any of the techniques introduced above.

Computing the index structures (i.e., the function *associated_inds*) is known to be time-consuming. Our findings indicate that for many applications this takes several minutes, i.e. index computation is only possible in a setup phase. Since for many applications this is not tolerable, new techniques had to be developed. The main problem is that for computing the index structure *associated_inds* the direct types are computed for every individual in isolation. Rather than asking for the direct types of every individual in a separate query, we investigated the idea of using *sets* of individuals which are "sieved" into the taxonomy, The idea is to use the procedure *non_instances*? to check whether all individuals from a set of candidates are obviously no instance of a given concept C (w.r.t. an A-box). If *non_instances*? returns *true*, many single A-box tests can be avoided. We call the approach the sets-of-individuals-at-a-time approach (see Algorithm 9 and Algorithm 10).

Algorithm 9 $traverse(inds, C, A, has\_member)$:

if $inds \neq \emptyset$ **then**
  for all $D \in children(C)$ **do**
    if $has\_member(D) = unknown$ **then**
      $instances\_of\_D := instance\_retrieval(D, inds, A)$
      $has\_member(D) := instances\_of\_D$; $traverse(instances\_of\_D, D, A, has\_member)$

In the natural language application we investigated, answering a specific query with realization-based instance retrieval and the set-of-individuals-at-a-time approach requires about 30 seconds using dependency-based instance retrieval (and 80 seconds using binary instance retrieval). Thus, for this specific application the performance gain for realization is a factor of three. But it still holds that, if A-boxes are not static, i.e., A-boxes are computed on the fly and only very few queries are posed w.r.t. the A-boxes, the direct implementation of instance retrieval as search without exploiting indexes is much faster (and it is possible even without T-box classification).

Algorithm 10 *compute_index_sets_of_individuals_at_a_time*(A):

for all $C \in CN$ **do**
  $has\_member(C) := unknown$; $associated\_inds(C) := \emptyset$
$traverse(individuals(A), top, A, has\_member)$; $has\_member(top) := individuals(A)$
for all $C \in CN$ **do**
  if $has\_member(C) \neq unknown$ **then**
    for all $ind \in has\_member(C)$ **do**
      if $\neg\exists D \in children(C) : ind \in has\_member(D)$ **then**
        $associated\_inds(C) := associated\_inds(C) \cup \{ind\}$

3.5 Dynamic Index-based Instance Retrieval

Computing a complete index (realization) as described in the previous subsection is possible if many queries are posed w.r.t. a "fixed" A-box (and T-box). However, sometimes realization is too time-consuming. Therefore, we devised a new strategy that exploits (i) explicitly given information (e.g. from A-box assertions of the form $i : CN$ where CN is a concept name) and (ii) the results of previous instance retrieval queries.

The idea can be explained as follows. The function *associated_inds* associates a set $Inds$ of individuals with each concept name C such that for each $i \in Inds$ it holds that i is an instance of C, for each $D \in descendants(C)$ the individual $i \notin associated\_inds(D)$, and for each $D \in ancestors(C)$ the individual $i \notin associated\_inds(D)$.

The function *associated_inds* is updated due to the results of queries. Let us assume $i \in associated\_inds(C)$ and $C \in ancestors(E)$. If it turns out that i is an instance of E, the function *associated_inds* is changed accordingly. Thus, the index evolves as instance retrieval queries are answered. Therefore, we call this strategy dynamic index-based instance retrieval.

In this new approach, the function $associated\_inds(C)$ returns an individual i even if C is not "most specific", i.e. even if there might exist a subconcept D of C such that i is also an instance of D. The consequence is that Algorithm 8 is no longer complete. The idea of only considering the parents of the query concept (see the line marked with an asterisk in Algorithm 8) must be dropped. Before we give a complete algorithm for dynamic index-based instance retrieval, further optimization techniques are introduced.

Let us assume concept D is a subsumer of C. If it is known for an individual $i \in associated\_inds(D)$ that $D \in direct\_types(i)$, then i can be removed from the set of candidates for the query concept C. With each concept name we also associate a set of non-instances. The non-instances are found by queries for the direct types of an individual (the non-instances are associated with the children of each direct type) or by exploiting previous calls to the function *instance_retrieval*. If an individual i is found not to be an instance of a query concept D, this is recorded appropriately by including i in $associated\_non\_instance(D)$ if there is no $E \in ancestors(D)$ such that $i \in associated\_non\_instance(E)$ (non-redundant caching). The non-instances of a query concept can then be discarded from the set of candidates. The new algorithm for instance retrieval is shown in Algorithm 11.

Algorithm 11 *dynamic_index_based_instance_retrieval_1*(C, A):

$known\_results := \bigcup_{D \in descendants(C)} associated\_inds(D)$
$possible\_candidates := \bigcup_{D \in (ancestors(C) \setminus \{C\})} associated\_inds(D)$
$candidates := possible\_candidates \setminus \bigcup_{D \in ancestors(C)} associated\_non\_instances(D)$
return $known\_results \cup instance\_retrieval(C, A, candidates \setminus known\_results)$

In order to evaluate the proposed algorithm, we first use a very simple T-box $\{Article \sqsubseteq Document, Book \sqsubseteq Document, CS\_Book \sqsubseteq Book\}$ and consider an A-box with the following assertions (for n we use different settings):

$doc\_1 : Article, doc\_2 : Article, \ldots doc\_n : Article,$
$doc\_n + 1 : Book, doc\_n + 2 : Book, \ldots doc\_n + n : Book,$
$doc\_n + n + 1 : CS\_Book, doc\_n + n + 2 : CS\_Book, \ldots doc\_n + n + n : CS\_Book$

In order to evaluate Algorithm 11, queries for *Book* and for *CS_Book* are executed. Queries can be ordered with respect to subsumption. Given the partial order induced by subsumption, an optimal execution sequence for answering multiple queries can be generated with a topological sorting algorithm. The more general queries are processed first, yielding a (possibly reduced) set of candidates for more specific queries as a by-product. This is demonstrated by considering the query set $\{retrieve(Book), retrieve(CS\_Book)\}$. There are two strategies, either all instances of *CS_Book* are retrieved first (Strategy 1) or all instances of *Book* are retrieved first (Strategy 2). The runtimes of the query sets under different strategies are indicated in Table 1.

In the first column the number n is specified (note that the A-box contains three times as many individuals), in the second column the time to generate the problem (i.e., the time to "fill" the A-box) is specified, in the third column the time for the initial A-box

n	Gen. Time	ASAT	Strategy 1	Strategy 2
10000	1	6	7	5
20000	3	10	29	19
30000	22	15	79	42
40000	34	23	164	115
50000	54	34	320	200
60000	80	42	904	552

Table 1: Runtimes (in secs) of instance retrieval query sets with different strategies.

consistency test is displayed, and in the last two columns the runtimes for the different strategies are indicated. All tests were performed on a 1GHz Powerbook running Mac OS X. Memory requirements are neglectable for all experiments ($\leq$ 100MB). Table 1 reveals that for larger values of n, Strategy 2, i.e., to first retrieve all instances of the superconcept *Book*, is approximately twice as fast as Strategy 1. The reason is that with Strategy 2 the set of candidates for the second instance retrieval query can be considerably reduced due to dynamic index-based instance retrieval.

In order to compare static index-based instance retrieval (one-by-one and set-based realization) with dynamic index-based instance retrieval, we used the synthetically generated A-box benchmarks described in Section 2.2).

Name	d	b	n	L	B	ASAT	static (1)	static (2)	dynamic
SCT	3	5	20	0.4	0.5	1.3	6.1	2.7	1.4
SCT	3	5	30	0.5	0.8	2.4	9.9	4.5	2.9
SCT	4	5	10	1.1	1.6	5.4	36.0	7.3	6.2
SCT	4	5	30	3.7	5.1	15.9	330.5	40.7	17.6
SCT	5	5	10	10.9	16.4	18.3	1528.0	70.6	31.7
SCT	5	5	30	62.821	54.8	76.8	timed out	184.7	160.3
SCT rel	3	5	10	0.5	0.8	1.4	2.8	3.6	2.6
SCT rel	4	5	10	3.3	7.5	10.2	40.3	17.4	17.7
SCT rel	5	5	10	22.0	120.0	144.6	1751.0	190.6	159.143

Table 2: Runtimes (in secs) for processing retrieval queries with static and dynamic index-based instance retrieval techniques.

The test characteristics are specified in the first four columns (SCT stands for symmetric concept tree). In column 'L' the time to load the problem from a file is given, and in column 'B' the time to build the index structures required by consistency checking and instance retrieval is indicated. The column 'ASAT' contains the time for the initial A-box consistency test (including the index building time from column B). The column 'static (1)' indicates the time for instance retrieval using the sets-of-individuals-at-a-time

realization approach whereas 'static (2)' indicates the time using the one-individual-at-a-time approach. The last column contains the runtime for dynamic index-based instance retrieval. The results obtained from analyzing the experiments can be summarized as follows.

One-individual-at-a-time realization is much faster for these tests than using sets-of-individuals-at-a-time realization. In these synthetic benchmarks, there exist n instances for each of the b^d concept names. The assumption that the result set contains only few individuals is not met in these benchmarks (the result set contains $b^{(d-1)}$ elements). Furthermore, it can be seen that dynamic index-based instance retrieval causes almost no overhead for these synthetic benchmarks (this may be due to the fact that the retrieval concept is located close to the root of the taxonomy). In addition it becomes apparent that the runtime for instance retrieval is mostly dominated by the initial A-box satisfiability test (which cannot be easily eliminated). In particular, building index structures is an expensive process (see column B) and cannot be neglected. Faster query evaluation results for RACER can be achieved by optimizing this process.

4 Conclusion

In this paper we demonstrated optimization techniques that make A-box inferences based on tableau-based DL systems suitable for many non-naive applications. We motivated the techniques described in this paper with the semantic web scenario and its representation language OWL/RDF. In this context, reasoning over individuals (e.g., instance retrieval) cannot be easily reduced to database lookups. The examples we gave here do not cover the full expressivity of OWL, however, they already demonstrate the need for more advanced optimization techniques.

For very restricted sublanguages of OWL (i.e., no existential restrictions at all), initial experiments indicate that datalog-based approaches could become an alternative to tableau-based approaches [21]. However, research in this area has just started and no stable implementations are available at the time of this writing. We have shown that tableau-based algorithms provide a sound basis for applications, provided the implementation technique proposed in this paper are implemented in practical systems. Nevertheless, the experiments also show some limitations of current DL technology. Only up to 30,000 individuals can be appropriately handled by current system implementations given non-naive T-boxes (ontologies) and A-boxes. Note that this holds for all data stored in main memory. Further research is necessary (in particular for contexts such as the semantic web) to provide for appropriate internal data structures in order to avoid unnecessary overhead. This holds for instance retrieval as well as for other inference services provided by current DL systems.

References

[1] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In D. Hutter and W. Stephan, editors, *Festschrift in honor of Jörg Siekmann*. LNAI. Springer-Verlag, 2003.

[2] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[4] S. Bechhofer, I. Horrocks, and C. Goble. OilEd: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna*. LNAI Vol. 2174, Springer-Verlag, 2001.

[5] S. Bechhofer, R. Möller, and P. Crowther. The DIG description interface. In *Proc. International Workshop on Description Logics – DL'03*, 2003.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[7] A. Borgida and R. Brachman. Loading data into description reasoners. *ACM SIGMOD Record*, 22(2):217–226, 1993.

[8] Paolo Bresciani. Querying databases from description logics. In *Knowledge Representation Meets Databases*, 1995.

[9] D. Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF Schema, http://www.w3.org/tr/2002/wd-rdf-schema-20020430/, 2002.

[10] Malte Gabsdil, Alexander Koller, and Kristina Striegnitz. Building a text adventure on description logic. In *International Workshop on Applications of Description Logics, Vienna, September 18*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-44/`, 2001.

[11] Malte Gabsdil, Alexander Koller, and Kristina Striegnitz. Playing with description logic. In *Proceedings Second Workshop on Methods for Modalities M4M-02*. `http://turing.wins.uva.nl/~m4m/M4M2/program.html`, November 2001.

[12] V. Haarslev and R. Möller. The Racer user's guide and reference manual, 2003.

[13] V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy*, Lecture Notes in Computer Science, pages 29–44. Springer-Verlag, June 2001.

[14] Volker Haarslev and Ralf Möller. Consistency testing: The RACE experience. In *Proceedings International Conference Tableaux'2000*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 57–61. Springer-Verlag, 2000.

[15] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, 2001.

[16] Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. Exploiting pseudo models for tbox and abox reasoning in expressive description logics. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[17] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Computer Science, Germany, 2000. Springer Verlag.

[18] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 177–191. Springer-Verlag, 2002.

[19] O. Lassila and R.R. Swick. Resource description framework (RDF) model and syntax specification. recommendation, W3C, february 1999. http://www.w3.org/tr/1999/rec-rdf-syntax-19990222, 1999.

[20] Lei Li and Ian Horrocks. Matchmaking using an instance store: Some preliminary results. In *Proceedings of the 2003 International Workshop on Description Logics (DL'2003)*, 2003.

[21] B. Motik, R. Volz, and A. Maedche. Optimizing query answering in description logics using disjunctive deductive databases. In *Proceedings of the 10th International Workshop on Knowledge Representation Meets Databases (KRDB-2003)*, pages 39–50, 2002.

[22] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1990.

[23] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[24] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference, http://www.w3.org/tr/owl-guide/, 2003.