Evaluating Reasoners Under Realistic Semantic Web Conditions

Yingjie Li, Yang Yu and Jeff Heflin

Department of Computer Science and Engineering, Lehigh University
19 Memorial Dr. West, Bethlehem, PA 18015, U.S.A.
{yil308, yay208, heflin}@cse.lehigh.edu

Abstract. Evaluating the performance of OWL Reasoners on ontologies is an ongoing challenge. LUBM and UOBM are benchmarks to evaluate Reasoners by using a single ontology. They cannot effectively evaluate systems intended for multi-ontology applications with ontology mappings, nor can they evaluate OWL 2 applications and generate data approximating realistic Semantic Web conditions. In this paper we extend our ongoing work on creating a benchmark that can generate user-customized ontologies together with related mappings and data sources. In particular, we have collected statistics from real world ontologies and used these to parameterize the benchmark to produce more realistic synthetic ontologies under controlled conditions. The benchmark supports both OWL and OWL 2 and applies a data-driven query generation algorithm that can generate diverse queries with at least one answer. We present the results of initial experiments using Pellet, HermiT, OWLIM and DLDB3. Then, we show the approximation of our synthetic data set to real semantic data.

Keywords: Benchmark, Ontology generation, Query generation

1 Introduction

Various semantic applications based on ontologies have been developed in recent years. They differ in the ontology expressivity such as RDF, OWL, OWL 2 or some fragment of these languages or the number of ontologies such as single-ontology systems or multi-ontology federated systems. One of the major obstacles for these system developers is that they cannot easily find a real world experimental data set to evaluate their systems in terms of the ontology expressivity, the number of ontologies and data sources, the ontology mappings and so on. In order to bridge this gap, LUBM [5] and UOBM [9] were developed to evaluate Semantic Web knowledge base systems (KBSs) by using a single domain ontology. But they cannot effectively evaluate systems intended for multi-ontology applications with ontology mappings, nor can they evaluate OWL 2 applications and generate data approximating realistic Semantic Web conditions. To solve these problems, we extend our early work [8] on creating a benchmark that can generate user-customized ontologies together with related mappings and data sources. In particular, we have collected statistics from real world ontologies and data sources and used these to parameterize the benchmark to produce realistic synthetic ontologies under controlled conditions. Unlike our previous work, this benchmark allows us to speculate about different visions of the future Semantic Web and examine how current systems will perform in these contrasting scenarios. Although Linking Open Data and Billion Triple Challenge data is frequently used to test scalable systems on real data, these sources typically have weak ontologies and little ontology integration (i.e., few OWL and OWL 2 axioms). The benchmark can be also used to speculate about similar sized (or larger) scenarios where there are more expressive ontologies and richer mappings.

Based on our previous work [8], in this paper, we first extend the two-level customization model including the web profile (to customize the distribution of different types of desired ontologies) and the ontology profile (to customize the relative frequency of various ontology constructors in each desired ontology) to support any sublanguage of both OWL and OWL 2 by taking OWL 2 constructors as our constructor seeds instead of OWL constructors. Then, we demonstrate that our benchmark can be used to evaluate OWL/OWL 2 reasoners such as Pellet [12], HermiT [10], OWLIM [7] and DLDB3 [11]. Finally, we show how well our synthetic data approximates real semantic data, specifically using the Semantic Web Dog Food corpus.

The remainder of the paper is organized as follows: Section 2 reviews the related work. Section 3 describes the benchmark algorithms. Section 4 presents the experiments. Finally, in Section 5, we conclude and discuss future work.

2 Related Work

The LUBM [5] is an example of a benchmark for Semantic Web knowledge base systems with respect to large OWL applications. It makes use of a university domain workload for evaluating systems with different reasoning capabilities and storage mechanisms. L. Ma et al. [9] extended the LUBM to make another benchmark - UOBM so that OWL Lite and OWL DL can be supported. However, both of them use a single domain/ontology and did not consider the ontology mapping requirements that are used to integrate distributed domain ontologies in the real Semantic Web. In addition, they do not allow users to customize requirements for their individual evaluation purposes. S. Bail et al. proposed a framework for OWL benchmarking called JustBench [1], which presents an approach to analyzing the behavior of reasoners by focusing on justifications of entailments through selecting minimal entailing subsets of an ontology. However, JustBench only focuses on the ontology TBoxes and does not consider the ABoxes (data sources) and the TBox mappings. C. Bizer et al. proposed a Berlin SPARQL Benchmark (BSBM) for comparing the SPARQL query performance of native RDF stores with the performance of SPARQL-to-SQL rewriters [2]. This benchmark aims to assist application developers in choosing the right architecture and the right storage system for their requirements. However, the BSBM can only output benchmark data in an RDF representation and a purely relational representation and does not support users' customizations on OWL and OWL 2 for different applications. I. Horrocks and P. Schneider [6] proposed a benchmark suite comprising four kinds of tests: concept satisfiability tests, artificial TBox classification tests, realistic TBox classification tests and synthetic ABox tests. However, this approach neither creates OWL ontologies and SPARQL queries nor ontology mappings, and only focuses on a single ontology at a time. Also, it did not consider users' customizability requirements.

3 The Benchmark Algorithms

In this section, we first introduce our extended two-level customization model consisting of a web profile and several ontology profiles for users to customize ontologies, then briefly describe the axiom construction and data source generation, and finally give an introduction to the data-driven query generation algorithm.

3.1 The Extended Two-level Customization Model

In order to support both OWL and OWL 2, our extended two-level customization model chooses the set of OWL 2 DL constructors as our constructor seeds and is designed to be flexible in expressivity by allowing users to customize these constructors in range of OWL 2 DL. Similar to our previous work [8], we still use ontology profiles to allow users to customize the relative frequency of various ontology constructors in the generated ontologies and web profile to allow users to customize the distribution of different types of ontologies. However, in this paper, besides user customized ontology profile, our extended benchmark can also automatically collect statistics of Table 1 listed types of axioms from real world ontologies and use them to parmeterize our ontology profile in order to generate realistic synthetic ontologies.

Compared to OWL, OWL 2 offers new constructors for expressing additional restrictions on properties such as property self restriction, new characteristics of properties such as data property cardinality, property chains such as property chain inclusions and keys such as property key. OWL 2 also provides new data type capabilities such as data intersection, data union, etc. In order to support these new additions in OWL 2, we categorized all OWL 2 DL constructors into five groups: axiom types, class constructors, object property constructors, data type property constructors and data type constructors. Since the data type property constructors contain one and only one constructor ($DatatypeProperty$), in each ontology profile, we let users fill in four tables with their individual configurations: the axiom type (AT) table, the class table (CT), the object property constructor table (OPT) and the datatype constructor table (DTT). The new constructor table is shown in Table 1. Compared to the old one in [8], the new table extends AT with constructors of $disjointUnionOf$, $ReflexiveProperty$, etc. and CT with constructors of $dataAllValuesFromRestriction$, $dataSomeValues$

FromRestriction, etc. The new *DTT* contains data constructors such as *dataComplementOf*, *dataUnionOf*, etc. As a result, Table 1 contains eleven types of operands in total: class type (*C*), named class type (*NC*), object property type (*OP*), datatype property type (*DP*), instance type (*I*), named object property (*NOP*), named datatype property (*NDP*), which is not listed in the table because it is only for *DatatypeProperty*, facet type (*F*), data type (*D*), a literal (*L*) and an integer number (*INT*). The *C* means the operand is either an atomic named class or a complex sub-tree that has a class constructor as its root. The *NC* means the operand is a named class. The *OP*, *DP* means the operand can be one of constructors listed in the table of object property and a datatype property, respectively. The *NOP*, *NDP* means the operand is not a complex constructor but a named object property or a named datatype property respectively. The *I* means the operand can be a single instance. The *F* is the facet type borrowed from XML Schema Datatypes. The *D* is the data type. The *L* is a literal. The *INT* stands for an integer number for the cardinality restriction. In these types, *NC*, *NOP*, *NDP*, *F*, *I*, *L* and *INT* are leaf node types.

In Table 1, {x} stands for a set of instances, whose cardinality is set by a uniform distribution. For cardinality constructors such as *minCardinality*, *maxCardinality*, *Cardinality*, *minQualifiedCardinality*, *maxQualifiedCardinality*, *qualifiedCardinality*, since the involved integer value should be positive and 1 is the most common value in the real world, we apply the Gaussian distribution with mean being 1, standard deviation being 0.5 (based on our experiences) and each generated value required to be greater than or equal to 1.

Table 1. Axiom type constructors, class constructors and property constructors.

Axiom Type Constructor					Class Constructor				
Constructors	**DL Syntax**	**Op1**	**Op2**	**Op3**	**Constructors**	**DL Syntax**	**Op1**	**Op2**	**Op3**
rdfs:subClassOf	$C1 \sqsubseteq C2$	C	C		allValuesFrom	$\forall P.C$	OP	C	
rdfs:subPropertyOf	$P1 \sqsubseteq P2$	OP	OP		someValuesFrom	$\exists P.C$	OP	C	
equivalentClass	$C1 \equiv C2$	C	C		intersectionOf	$C1 \sqcap C2$	C	C	
equivalentProperty	$P1 \equiv P2$	OP	OP		one of	$\{x1,\ldots,x2\}$	{I}		
disjointWith	$C1 \sqsubseteq \neg C2$	C	C		unionOf	$C1 \sqcup C2$	C	C	
TransitiveProperty	$P^{+} \sqsubseteq P$	NOP			complementOf	$\neg C$	C		
SymmetricProperty	$P \equiv (P^{-})$	NOP			minCardinality	$\geq nP$	OP	INT	
FunctionalProperty	$T \sqsubseteq \leq 1P^{+}$	NOP			maxCardinality	$\leq nP$	OP	INT	
InverseFunctionalP.	$T \sqsubseteq \leq 1P$	NOP			Cardinality	$= nP$	OP	INT	
rdfs:domain	$\geq 1P \sqsubseteq C$	NOP,DP	NC		hasValue	$\exists P.\{x\}$	OP	I	
rdfs:range	$T \sqsubseteq \forall U.D$	NOP,DP	NC,D		namedClass				
disjointUnionOf		C	{C}		dataAllValuesFromR.		DP	D	
ReflexiveProperty		NOP			dataSomeValuesFromR.		DP	D	
IrreflexiveProperty		NOP			minQualifiedCardinality		OP,DP	INT	C,D
AsymmetricProperty		NOP			maxQualifiedCardinality		OP,DP	INT	C,D
propertyDisjointWith		OP,DP	OP,DP		qualifiedCardinality		OP,DP	INT	C,D
propertyChainAxiom		NOP	NOP	NOP	dataHasValue		DP	L	
hasKey		C	{C}		hasSelf		OP	TRUE	
Object Property Constructor					Datatype Constructor				
inverseOf	P^{-}	OP			dataComplementOf		D		
namedProperty					dataUnionOf		D	D	
					xsdDatatype				
					namedDatatype		F		
					dataIntersectionOf		D	D	
					dataOneOf		L		

A sample input of the new ontology profiles and web profile is shown in Fig.1. In this sample input, the web profile contains eight ontology profiles: RDFS, OWL Lite, OWL DL, Description Horn Logic (DHL), OWL 2 EL, OWL

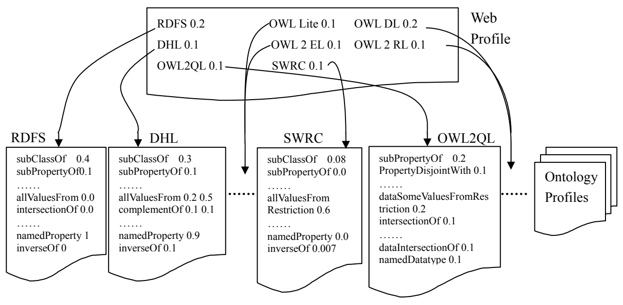

Fig. 1. Two-level customization model.

2 RL, OWL 2 QL and SWRC (Semantic Web Dog Food). Their distribution probabilities are set to be 0.2, 0.1, 0.2, 0.1, 0.1, 0.1, 0.1 and 0.1 respectively. This configuration means that in our final generated ontologies, 20% ontologies use RDFS, 10% ontologies use OWL Lite, 20% ontologies use OWL DL, 10% use DHL, 10% use each of the three OWL 2 profiles and 10% use SWRC ontology. For each ontology profile, the distributions of different ontology constructors used in the generated ontology are displayed. Each cell in the input tables is a number between 0 and 1 inclusive, which means the percentage of this constructor appearing in a generated ontology. Note, the SWRC profile is learned from the statistics of the real SWRC ontology instead of user customization. Also, in ontology languages such as DHL, an axiom has different restrictions on its left hand side (LHS) and right hand side (RHS). To support this, users can specify two probabilities for a constructor, as shown in the DHL profile of Fig. 1.

3.2 Axiom Construction and Data Source Generation

Since each ontological axiom can be seen as a parse tree with the elements in the axiom as tree nodes, the table AT actually contains those elements that can be used as the root. The tables CT, OPT and DTT provide constructors that can be used to create class, object property and datatype property expressions respectively as non-root nodes. From this perspective, the axiom construction can be seen a parse tree construction. During this process, the web profile is first used to select the configured ontology profile(s). Second, we use the distribution in the selected ontology profile to randomly select one constructor from AT table as the root node. Then, according to operand type of the selected root constructor, we use the CT, OPT or DTT tables to randomly select one class, one object property or one data type constructor to generate our ontological axioms. This process is repeated until either each branch of the tree ends with a leaf node type or the depth of the parse tree exceeds the given depth threshold. For ontology mappings, since each mapping is essentially an axiom, we apply the same procedure. Thus, the mapping ontologies have the same expressivity as the domain ontologies. Besides, we also need to consider the linking strategy of different ontologies, which is described in detail in our previous work [3].

For every domain ontology, we generate a specified number of data sources. In our configuration, this number can be set by the benchmark users according to their individual needs. For every source, a particular number of classes and properties are used for creating triples. They can be also controlled by specifying the relevant parameters in our configuration. To determine how many triples each source should have, we collected statistics from 200 randomly selected real-world Semantic Web documents. Since we found that the average number of triples in each result document is around 54.0 with a standard deviation of 163.9, we set the average number of triples in a generated source to be 50 by using a Gaussian distribution with mean 50 and standard deviation 165. In addition, based on our statistics of the ratio between the number of different URIs and the number of data sources in the Hawkeye knowledge base [4], we set the total number of different URIs in the synthetic data set to equal to the number of data sources times a factor around 2 in order to avoid the instance saturation during the source generation. In order to make the synthetic data set much closer to real world data, we ensure that each source is a connected graph, which more accurately reflects most real-world RDF files. To achieve this point, in our implementation, those instances that have already been used in the current source are chosen to generate new triples with higher priority.

In our benchmark, we also generate some *owl:sameAs* triples. Based on our Sindice statistics of randomly issuing one term query and took the top 1000 returned sources as samples, we found 27.1% of them contain *owl:sameAs* statements. Thus, our benchmark generates *owl:sameAs* triples in a ratio of 27.1% of the total number of triples. Furthermore, for each instance involved into the *owl:sameAs* triple, according to our experiences, we take a probability of 0.1 to select it from the set of all generated instances in the whole data set and a probability of 0.9 to select it from the set of all generated instances in the current source. As a result, all of *owl:sameAs* triples in our data set are categorized into different equivalence classes. Each equivalence class is defined to be a set of instances that are equivalent to each other (explicitly or implicitly connected by *owl:sameAs*). The average cardinality of the equivalence class is around 3.7.

3.3 Data-driven Query Generation Algorithm

It is well-known that the RDF data format is by its very nature a graph. Therefore, a given semantic web knowledge base (KB) can be essentially modeled as one big possibly disconnected graph. On the other hand, each SPARQL query is basically a subgraph and in order to guarantee each query has at least one answer, our SPARQL queries can be generated from the subgraphs over the big KB graph. Therefore, we proposed a data-driven query generation algorithm in our work [8]. Here, we only summarize this algorithm in order to make the paper complete.

According to the algorithm, we first identify a subgraph meeting the initial query configuration from the big KB graph. Within the identified subgraph, we randomly select one node as the starting node to construct a query pattern graph (QPG). Begin with the starting node, we randomly select one edge that is starting with the starting node and not contained in QPG and then add this edge into QPG. If the selected edge is already in QPG, we need to select another edge that is not selected before. Then, we replace the ending node of the newly added edge with a new variable in the probability P. Currently, the default value of P is set 0.5. This process is iterated until the QPG qualifies the initial query configuration. By this step, we have successfully constructed one QPG. Then, we need to check if each edge in QPG contains at least one variable. If not, we randomly replace one node of the edge without variable nodes with a new variable. Based on the variable-assigned QPG, a SPARQL query can be generated and returned. Note, if the junction node of QPG is replaced by a query variable, this variable is counted as a join variable. For more details, please read our paper [8].

4 Evaluation

In order to demonstrate how our benchmark can be used to evaluate very different semantic reasoners, in this section, we describe two group of experiments. The first is to use our benchmark to evaluate four representative semantic reasoners: Pellet [12], HermiT [10], OWLIM [7] and DLDB3 [11]. The second is to show the approximation of our synthetic data set to real semantic data under the control of collected statistics from real world ontologies. For each experiment in each group, we issued 100 random queries, which are grouped by the number of QTPs that ranges from one to ten. Each group has ten queries. Each query has at most twenty query variables. Each QTP of each query satisfies the join condition with at least one sibling QTP. We denote an experimental configuration as follows: (*nO*-*nD*-*Ont*), where nO is the number of ontologies, nD is the number of data sources and Ont is the ontology profile type. In order to eliminate the outlier results, we applied the probabilistic statement of Chebyshev's inequality: $Pr(|X-\mu| \geq k\sigma) \leq \frac{1}{k^2}$, where X stands for a random variable, μ stands for the mean, σ stands for the standard deviation and $k = 3$, which counts at most 10% of each group of metric values as outliers. We applied this inequation to all metrics and for each system, any value that did not satisfy the inequation would be thrown out. The reason is that these outliers will greatly distort our experimental statistics. All our experiments are done on a workstation with a Xeon 2.93G CPU and 6G memory running UNIX.

4.1 Reasoner Evaluation

In this experiment, we want to evaluate our benchmark in various OWL and OWL 2 reasoners. In selecting the reasoners, first we decided to consider only noncommercial systems or free versions of those commercial ones. Moreover, we did not intend to carry out a comprehensive evaluation of all existing semantic reasoners and our selected ones should cover OWL and OWL 2. In addition, we also believe a practical semantic reasoner must be able to read OWL/OWL 2 files and provide programming APIs for loading data and issuing queries. As a result, we have settled on four different reasoners including Pellet 2.2.2, HermiT 1.3.4, SwiftOWLIM and DLDB3. Except DLDB3, all candidate systems are from the W3C OWL 2 implementation system website [1]. Other candidate systems such as FaCT++, CEL, ELLY, QuOnto and Quil, we rejected due to difficulties in obtaining functions executable for the Unix platform. Our experiments are grouped by the three W3C recommended OWL 2 profiles: OWL 2 EL, RL and QL because we wanted to investigate the physical (as opposed to theoretical) consequences of these profiles. We computed the query response time, the source loading time and the query completeness respectively for each test system. Since Pellet is complete for all OWL 2 profiles and able to complete all our experiments, we chose Pellet results as our completeness ground truth. The query completeness is defined to be $\frac{\#\text{ of answers returned by each test system for all tested queries}}{\#\text{ of answers returned by Pellet for all tested queries}}$. All experimental results are shown in Fig.2. Note, in Fig.2 (b), (d) and (f), the HermiT curve is hiding behind the Pellet because their performances are very close.

OWL 2 EL OWL 2 EL is intended for applications that have ontologies that contain very large numbers of properties and/or classes. It captures the expressive power used by many such ontologies. Therefore, in this experiment, we evaluate the target systems by varying the number of ontologies but keeping the data sources constant at 500. Fig.2(a) and Fig.2(b) show how each system's query response time and loading time are affected by increasing the number of ontologies in OWL 2 EL. From these results, we can see that OWLIM performs best in both query response time and loading time. DLDB3 suffers from the worst loading time because it uses a persistent database backend, while the other three systems are in-memory. Pellet has better query response time than Hermit but performs very close to HermiT in loading time. Pellet and HermiT are complete, but DLDB3 and OWLIM are incomplete with 21.97% and 40.5% completeness on average respectively. The reason is that DLDB3 is only a limited OWL reasoner and does not support OWL 2, while OWLIM is only complete for OWL 2 RL and QL and incomplete for OWL 2 EL. Of the four systems, Pellet appears to be the best choice for EL. Although OWLIM is the fastest, it is significantly lacking in completeness.

OWL 2 QL OWL 2 QL focuses on applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In this experiment, we evaluate the target systems by varying the number of data sources with the constant number of 5 ontologies. Fig.2(c) and Fig.2(d) show how each system's query response time and loading time are affected by increasing the number of data sources in OWL 2 QL. In both query response time and loading time, OWLIM performs best, while HermiT is worst. In particular, HermiT cannot scale to points of 5-5000-QL and 5-10000-QL because of an out of memory error but OWLIM, Pellet and DLDB3 can. Starting from point of 5-5000-QL, the loading time of Pellet performs worse than DLDB3 even though

[1] http://www.w3.org/2007/OWL/wiki/Implementations

DLDB3 uses secondary storage. We think the reason is that Pellet is in-memory and when the number of loaded data sources increases to some number (5000 in our experiment), it requires more memory to do the consistency checking during its loading period than the memory we have provided (6GB). OWLIM, Pellet and HermiT (in the first three points) are complete. DLDB3 is incomplete with 68.81% completeness on average, but it is better than it did on OWL 2 EL. In summary, of the four systems, OWLIM appears to be the best choice for OWL 2 QL. DLDB3 is an alternative for large scales where some incompleteness is tolerable.

OWL 2 RL OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency. Therefore, in this experiment, we evaluate the target systems by varying both the number of ontologies and the number of data sources. Fig.2(e) and Fig.2(f) show how each system's query response time and loading time are affected by increasing the number of ontologies and the number of data sources. As shown by the results, OWLIM still performs best in the query response time and loading time. HermiT suffers from the worst performance and cannot scale to points of 10-2000-*RL* and 15-3000-*RL*. As was the case to OWL 2 QL, Pellet starts to have worse loading time than DLDB3 from the point of 10-2000-*RL* because it requires more memory. OWLIM, Pellet and HermiT are complete, but DLDB3 is still incomplete with 44.96% completeness on average. In summary, OWLIM appears to be the clear winner for OWL 2 RL.

4.2 Approximation Evaluation

Constructors	**Percentage**	**Constructors**	**Percentage**
rdfs:subClassOf	8.13%	complementOf	8.98%
TransitiveProperty	0.17%	intersectionOf	8.98%
rdfs:domain	1.93%	unionOf	8.98%
rdfs:range	1.85%	namedClass	1.51%
oneOf	1.09%	allValuesFromRestriction	57.71%
inverseOf	0.67%		

Table 2. SWRC ontology constructor statistics.

In this experiment, we evaluate how approximate our synthetic data set is to real semantic data. We have chosen Semantic Web Dog Food (SWRC) corpus [2] as our real semantic data set. Since the downloaded SWRC data is in

[2] http://data.semanticweb.org/dumps/

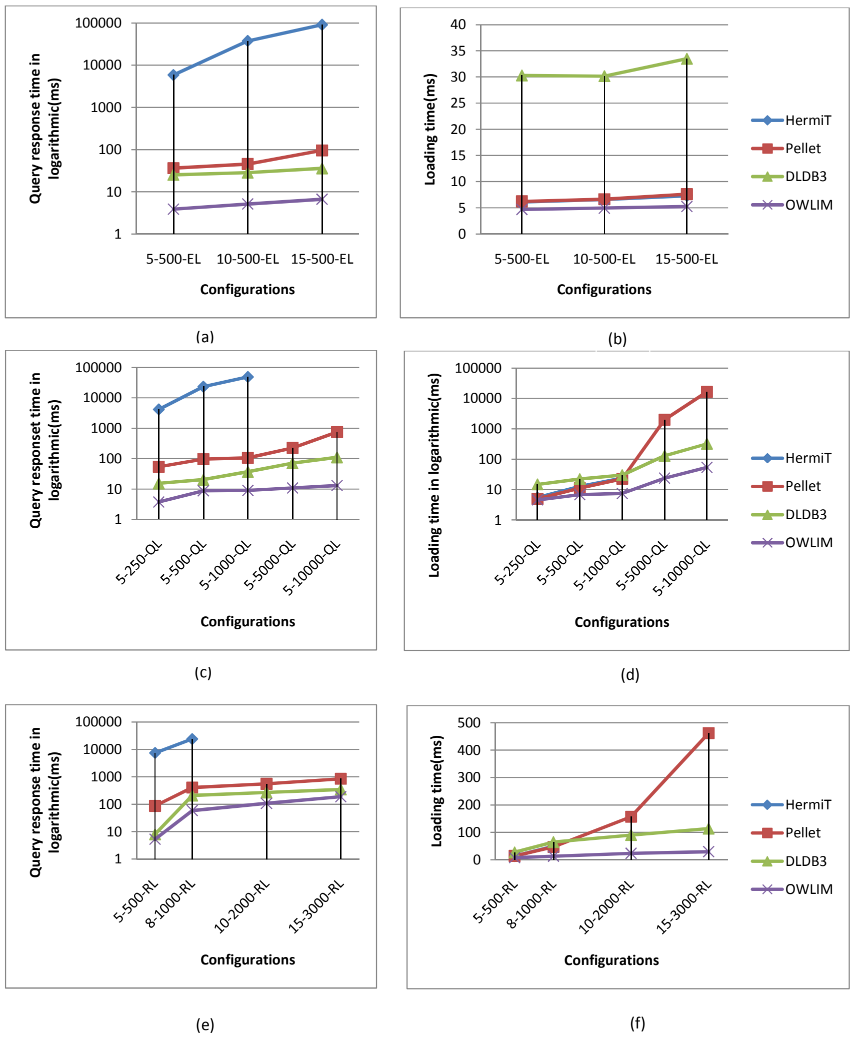

Fig. 2. Query response time and loading time of OWL 2 EL, QL and RL.

dump, in order to make it meet our experimental setup, we partitioned it into different subsets using the number of triples of each test configuration ($50 \times nD$ in each configuration). In addition, we collected the statistics of the number of each type of ontological axiom shown in Table 1 in the SWRC ontology and used them to parameterize the benchmark to produce realistic synthetic ontologies. For each test configuration, we use the learned SWRC ontology profile to generate five SWRC-like domain ontologies together with their corresponding ontology mappings. The SWRC ontology constructor statistics is shown in Table 2. We evaluate Pellet and OWLIM because both systems are memory-based and two representatives of applying two different well-known reasoning algorithms: tableau and rule-based. We compute the average query response time for each configuration. As shown in Fig. 3, although Pellet is slightly faster on benchmark data than on real data, and OWLIM is slightly slower on the benchmark data, each maintains the same trend on the benchmark that it had with the original data. Although more experiments are needed to draw significant conclusions, this suggests that our benchmark may be able to generate synthetic datasets that are representative enough for users to evaluate their systems or reasoners instead of using the real semantic data, which cannot be easily customized and has little ontology integration.

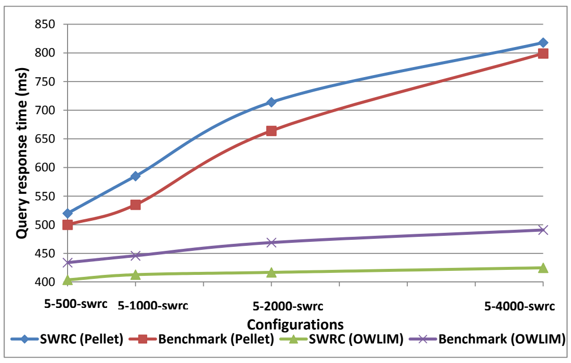

Fig. 3. Query response time of Benchmark and SWRC.

5 Conclusions and Future Work

We extend our early work [8] on creating a benchmark that can generate user-customized ontologies together with related mappings and data sources. It supports both OWL and OWL 2. It can also generate diverse conjunctive queries with at least one answer. Our experiments have demonstrated that this benchmark can effectively evaluate semantic reasoners by generating realistic synthetic semantic data. In particular, we have collected statistics from real world ontologies and used these to parameterize the benchmark to produce more realistic synthetic ontologies under controlled conditions. According to our evaluation, OWLIM has the best performance and is complete for OWL 2 QL, RL and SWRC. Pellet has better performance than HermiT in all experimental settings. HermiT has the worst scalability in OWL 2 QL and RL because of high memory consumption. DLDB3 is incomplete in all OWL 2 profiles because it is designed for an OWL fragment reasoner, but it shows better performance in query response time than Pellet and HermiT.

However, there is still significant room for improvement. First, we need to consider to collect statistics of semantic data besides the ontology schemas. One way to do so is to find the different RDF graph patterns implied by the real semantic data and use these to guide our data generation. Second, in the approximation evaluation, we need to evaluate our benchmark using more data sets with different characteristics from SWRC. It should be also pointed out that we believe that the performance of any given system will vary depend on the structure of the ontology and data used to evaluate it. Thus our benchmark does not provide the final say on each system's characteristics. However, it allows developers to automate a series of experiments that give a good picture of how system performs under the general parameters of a given scenario.

References

1. S. Bail, B. Parsia, and U. Sattler. JustBench: A framework for owl benchmarking. In *International Semantic Web Conference (1)*, pages 32–47, 2010.
2. C. Bizer and A. Schultz. The berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
3. A. Chitnis, A. Qasem, and J. Heflin. Benchmarking reasoners for multi-ontology applications. In *EON*, pages 21–30, 2007.
4. Z. P. et al. Hawkeye: A practical large scale demonstration of semantic web integration. Technical Report LU-CSE-07-006, Lehigh University, 2007.
5. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.
6. I. Horrocks and P. F. Patel-Schneider. Dl systems comparison (summary relation). In *Description Logics*, 1998.
7. A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM - a pragmatic semantic repository for owl. In *WISE Workshops*, pages 182–192, 2005.
8. Y. Li, Y. Yu, and J. Heflin. A multi-ontology synthetic benchmark for the semantic web. In *In Proc. of the 1st International Workshop on Evaluation of Semantic Technologies*, 2010.
9. L. Ma, Y. Yang, Z. Qiu, G. T. Xie, Y. Pan, and S. Liu. Towards a complete owl ontology benchmark. In *ESWC*, pages 125–139, 2006.
10. B. Motik, R. Shearer, and I. Horrocks. Optimized reasoning in description logics using hypertableaux. In *CADE*, pages 67–83, 2007.
11. Z. Pan, Y. Li, and J. Heflin. A semantic web knowledge base system that supports large scale data integration. In *In Proc. of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 125–140, 2010.
12. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.