Benchmarking Reasoners for Multi-Ontology Applications

Ameet N Chitnis, Abir Qasem and Jeff Heflin

Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015
{anc306, abq2, heflin}@cse.lehigh.edu

Abstract. We describe an approach to create a synthetic workload for large scale extensional query answering experiments. The workload comprises multiple interrelated domain ontologies, data sources which commit to these ontologies, synthetic queries and map ontologies that specify a graph over the domain ontologies. Some of the important parameters of the system are the average number of classes and properties of the source ontology which are mapped with the terms of target ontology and the number of data sources per ontology. The ontology graph is described by various parameters like its diameter, number of ontologies and average out-degree of node ontology. These parameters give a significant degree of control over the graph topology. This graph of ontologies is the central component of our synthetic workload that effectively represents a web of data.

1 Introduction

One of the primary goals of the Semantic Web is to be able to integrate data from diverse sources irrespective of the ontology to which it commits to. Unfortunately it is difficult to measure progress against this goal. Although there are a large number of ontologies, few have data associated with them, thereby making it difficult to execute large scale integration experiments. The aim of this paper is to provide a benchmark for a synthetic workload that can be easily scaled to the desired configuration for executing large scale extensional query answering experiments.

The benchmark described here was originally developed for evaluating our OBII system [1]. However our approach could be applied to evaluate other Semantic Web systems in general. In this paper we present the various workload components that are of general interest. We also discuss wherever applicable how they can be further generalized. Specifically we make the following two technical contributions in this paper.

1. We design and implement an algorithm to generate a graph of ontologies defined by parameters like diameter, average out-degree of node ontology, number of paths having a diameter length, number of terminal ontologies, number of maps etc. Thereafter we generate mapping ontology axioms that conform to a subset of OWL DL.

2. We use these in conjunction with an approach to generate synthetic domain ontologies, synthetic data sources and synthetic queries in order to provide a complete Semantic Web workload.

The rest of the paper is organized as follows: In section 2 we provide a background about the related work. In Section 3 we define the various steps of data generation process like generation of domain ontologies, data sources, queries and the graph of ontologies to create mapping axioms. We introduce a mapping language for describing maps. In Section 4, we describe the methodology for carrying out an experiment and the performance metrics that can be used for evaluation. In Section 5, we conclude and discuss future work.

2 Background

The LUBM [2] is an example of a benchmark for Semantic Web knowledge base systems with respect to use in large OWL applications. It makes use of a university domain workload for evaluating systems with different reasoning capabilities and storage mechanisms. Li Ma et. al [3] extend the LUBM so that it can support both OWL Lite and OWL DL (except Tbox with cyclic definition and Abox with inequality definition). However LUBM and extended LUBM use a single domain/ontology namely the university domain comprising students, courses, faculty etc. We need workloads comprising multiple interrelated ontologies.

Tempich and Volz [4] perform statistical analysis of the available Semantic Web ontologies and derive important parameters which could be used to generate synthetic ontologies. T D. Wang et. al [5] have conducted a more recent survey on OWL ontologies and RDFS schemas to perform analysis over the statistical data and report some important trends. The latter are used to determine if there are interesting trends in modeling practices, OWL construct usages and OWL species utilization. These works can be used in determining reasonable parameters for Semantic Web benchmarks but do not present benchmarks in themselves.

There has been some prior work on benchmarking DL systems. Horrocks and Patel-Schneider [6] use a benchmark suite comprising four kinds of tests: concept satisfiability tests, artificial Tbox classification tests, realistic Tbox classification tests and synthetic Abox tests. The TBox refers to the intentional knowledge of the domain (similar to an ontology) and the ABox contains extensional knowledge. Elhaik et. al. [7] provide the foundations for generating random Tboxes and Aboxes. The satisfiability tests compute the coherence of large concept expressions without reference to a Tbox. However, these approaches neither create OWL ontologies nor SPARQL queries and only focus on a single ontology at a time.

Garcia-Castro and Gomez-Perez [8] provide a benchmark suite for primarily evaluating the performance of the methods provided by the WebODE ontology management API. Although their work is very useful in evaluating ontology based tools it provides less information on benchmarking knowledge base systems.

J. Winick and S. Jamin [9], present an Internet topology generator which creates topologies with more accurate degree distributions and minimum vertex covers as compared to Internet topologies. Connectivity is one of the fundamental characteris-

tics of these topologies. On the other hand while considering a Semantic Web of ontologies there could be some ontologies not mapping to any other ontology thereby remaining disconnected from the graph.

3 Data Generation

We now describe the process of generating several types of synthetic workloads to represent a wide variety of situations. While generating the data set the user is given the freedom to modify the independent parameters while the rest essentially serve as controls whose values are dependent on the nature of applications, like information integration etc. The characteristics of a domain ontology and a map ontology are clearly demarcated in that the former does not have any import statements and a map inherits the axioms of the two ontologies being mapped. This approach is equivalent to having a set of ontologies some of which inherit the axioms of the others. But our approach is very useful in creating the graph of ontologies.

3.1 Generation of Domain Ontologies

We implemented a workload generator that allows us to control several characteristics of our dataset. In generating the synthetic domain ontologies we decided to have on the average 20 classes and 20 properties (influenced by the dominance of small ontologies in the current Semantic Web).

Due to restrictions placed on our OBII system our existing implementation only generates domain ontologies comprising subClassOf and subPropertyOf axioms in order to support taxonomic reasoning. Also, following the statistical analysis of the DAML ontology library [4] we maintain more subClassOf axioms than subPropertyOf axioms. We designate these ontologies as simple ontologies. But however we can easily enhance the degree of expressivity to OWL DL or OWL Lite by including complex axioms like unionOf, intersectionOf, inverseOf etc; because the classes/properties used in our ontology are synthetic without possessing any intuitive semantics. Also, there has been some related work like the Artificial Tbox Classification tests of Horrocks and Patel-Schneider [6] for benchmarking DL systems.

To create a domain ontology, we randomly establish subClassOf and subPropertyOf relationships across classes and properties respectively. The class and property taxonomy have an average branching factor of 4 and an average depth of 3.

3.2 Generation of the graph of interlinked ontologies

We consider a directed graph of interlinked ontologies, where every edge is a map from the source ontology to the target ontology. This map ontology comprises a set of mapping axioms. We describe the following terms for discussing such a graph -

- Diameter: The length of the longest path in the graph

- Whether the node is a terminal node i.e. has a zero out-degree. Before the map is created, we determine the number of terminal nodes and randomly mark those many domain ontologies as terminal. The algorithm is so designed, that it prevents a non-terminal node from attaining a zero out-degree. Also, there could be some terminal nodes with a zero in-degree, thereby disconnecting them from the graph.
- Out-path length: The length of the longest outgoing path of a node
- In-path length: The length of the longest incoming path to a node

The inputs to the graph creation algorithm are the number of ontologies, average out-degree of the nodes and diameter of the graph. There is a parameter – *longPaths* which indicates the number of paths having a diameter length. This parameter has been hard coded to 1 because we need to have at least one path of diameter length. The algorithm usually creates additional paths having a diameter length.

Another important parameter is the total number of maps. We show how this can be calculated from other parameters.

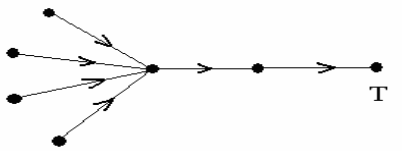
Let

maps – total number of maps
out – average out-degree
onts – total number of ontologies
term – total number of terminal ontologies

Parameters like *maps*, *out* and *term* are interrelated in that *maps* is approximately equal to the product of non terminal ontologies and *out*. Hence we have -

$$(onts - term) * out \cong maps \quad (1)$$

However we do not provide *term* as an input parameter. We show how a reasonable value can be computed from other parameters. We can express *maps* as the product of *term* and *diameter*. The number of maps is at least equal to this product. This is because the in-path length of a terminal node is equal to the diameter. There could be more maps, in situations where more than one diameter length path leads to a terminal node as explained below –



As shown above, the terminal node (marked 'T') has 4 paths of diameter length (diameter is 3) leading to it, effectively yielding more maps. Hence the equation below is desirable but not a requirement. Given that we prefer graphs that will branch out we will use -

$$term \cong maps / diameter \quad (2)$$

Substituting (2) in (1) we get

$$(onts - maps / diameter) * out \cong maps$$

$$onts * out - (maps * out) / diameter \cong maps$$
$$onts * out \cong maps + (maps * out) / diameter$$
$$onts * out * diameter \cong maps * diameter + maps * out$$
$$onts * out * diameter \cong maps * (diameter + out)$$

$$maps \cong (onts * out * diameter) / (diameter + out) \quad \textbf{(3)}$$

Also, by substituting (3) in (2)

$$term \cong (onts * out) / (diameter + out) \quad (\textbf{4})$$

Steps of the Algorithm

1. At the outset determine the number of terminal nodes using the equation- (4) above. Then randomly mark those many domain ontologies as terminal.
2. Thereafter create a path of diameter length. This ensures that there is at least one path of length equal to that of diameter.
3. For every non-terminal ontology, randomly select its out-degree which falls within some range of the specified average out-degree. This range extends by one half of the specified average out-degree on its either side. We choose a uniform distribution for generating a random number. Thereafter randomly select as many target ontologies as the chosen out-degree. The target ontology could be either terminal or non terminal. The sources and the target ontologies will eventually be used for creating mapping axioms.
4. While creating a map ontology between a source and a target certain constraints have to be satisfied which are as follows
    i. The in-path length of the source should be less than the diameter in order to prevent the creation of a path of length greater than the diameter.
    ii. The target should be different from the source
    iii. There shouldn't already exist a direct mapping between the source and the target
    iv. The target should not be among those ontologies from which the source could be visited. This prevents the creation of any cycles in the graph. This is a requirement for OBII, which could be relaxed for other systems.
    v. With the given source and the selected target a transitive path of length greater than the diameter shouldn't be created. This means that the in-path length of the source + the out-path length of the target + 1 should not be greater than the diameter.
    vi. If the target is a non-terminal node and by virtue of creating a map between the source and the target, the latter or any of its non-terminal descendants could become a terminal node then it should be avoided. This happens when the in-path length of the source is one less than the diameter.

vii. There sometimes arises a situation, where none of the existing nodes can satisfy the above constraints. This can happen in cases of large diameters and large out-degrees or when the diameter is equal to the number of ontologies. When such a situation arises a new ontology is created to serve as a target. Such ontologies which are dynamically created are termed as fresh ontologies. So the total number of ontologies at the end may be greater than the number of ontologies with which the algorithm began.

5. Once a map ontology is created the attributes of the source and the target have to updated as follows
   i. The source and the set of ontologies from which it can be reached must be added to the set of ontologies from which the target and its descendants can be reached
   ii. The out-degree of the source has to be updated.
   iii. The source must be made the parent of the target
   iv. The target should be made the child of the source
   v. The out-path length of the source and all its ancestors has to updated if applicable
   vi. The in-path length of the target and all its descendants has to be updated if applicable

3.3 Generation of mapping axioms

Once the source and the target ontologies have been identified mapping axioms need to be established. A specific number of terms (classes and properties) from the source ontology are mapped to terms in the target ontology. Since the domain ontologies are randomly chosen while creating a map ontology we expect the latter to reflect a partial overlap between the two. Hence this value has been hard coded to 20% of the total number of classes and properties in the source ontology.

OBII uses the language OWLII [1] which is a subset of OWL DL. This language has been defined as follows.

Definition **OWLII**

i. Let L_{ac} be a DL language where A is an atomic class, and if C and D are classes and R is a property, then $C \sqcap D$ and $\exists R.C$ are also classes.

ii. Let L_a include all classes in L_{ac}. Also, if C and D are classes then $C \sqcup D$ is also a L_a class.

iii. Let L_c includes all classes in L_{ac}. Also, if C and D are classes then $\forall R.C$ is also an L_c class.

iv. OWLII axioms have the form $C \sqsubseteq D$, $A \equiv B$, $P \sqsubseteq Q$, $P \equiv Q$, $P \equiv Q^{-}$, where C is an L_a class, D is an L_c class, A, B are L_{ac} classes and P, Q are properties.

At present we generate mapping axioms that fall strictly within OWLII. The limited expressivity of OWLII prevents generating inconsistent axioms, but when extended to

more expressive axioms we can incorporate a consistency check to the ontology generation process.

In what follows we first describe how this is implemented in our current system and how it can be easily extended to OWL-DL. We create each mapping axiom by essentially generating an OWL parse tree with the root node being a subclass operator. Then based on a user supplied frequency table of various OWL constructors the tree is expanded by using named classes or owl constructors. The frequency table allows the users to specify a ratio of various owl constructors which they expect to have in their mapping axioms.

Our algorithm recursively builds the parse tree based on the above and terminates by choosing named classes for all the remaining operands when the maximum length of a mapping axiom is reached. If there doesn't exist a mapping between a pair of ontologies, it simply means that the latter are not related and represent different domains. Such a landscape truly reflects the nature of semantic web comprising groups of interrelated ontologies as well as lone ontologies. Thus answering a query demands being selective about particular data sources instead of scanning the entire data set. Our OBII system [1] uses the concept of "rel-files" in order to select only those data sources which contain relevant information.

Note: In the above approach we essentially restrict the axiom generation to remain within OWLII by using certain constructors in either the subject or the object position of an axiom. This is done because our current implementation is geared towards data for OBII system. However, if we lift these restrictions and allow for any constructors to be on either side of the tree, we can generate axioms that are OWL-DL.

3.4 Generation of data sources

A specified number of data sources are generated for every domain ontology. Every data source comprises ABox assertions with named classes/properties. For every source a particular number of classes and properties are used for creating triples. These triples are added to the source ontology being created. The number of classes and properties to be used for creating triples can be controlled by specifying the relevant parameters. With our current configuration the average data source has 75 triples. Considering the sparse landscape of the number of classes/properties from an ontology which are actually instantiated [10] and also due to the lack of knowledge about the prospective manifestation of the actual semantic web we have currently chosen to instantiate 50% of the classes and 50% of the properties of the domain ontology. But however this can be easily modified to suit the nature of application.

3.5 Generation of Queries

Our query generation process generates SPARQL queries from a given set of ontologies. Currently we support single ontology queries i.e. queries that have predicates from a single namespace. This approach can be extended to multi ontology queries quite easily. In our current approach we randomly choose an ontology from a set of

ontologies to be the query ontology. These queries are conjunctive in nature as in the conjunctive query language of Horrocks and Tessaris [11]. We then randomly generate a set of query predicates. The number of predicates for each query is determined by a user specified parameter. We generate the queries based on the following policies:

1. We choose the first predicate from the classes of the query ontology.
2. We bias the next predicate to have a 75% (modifiable) chance of being one of the properties of the query ontology in order to achieve some degree of control over query selectivity.
3. In order to generate interesting queries that require some joins between query predicates, we need to have variables that are shared by at least two predicates of a given query. In order to guarantee this shared variable, when generating a new predicate we can use one variable from the previous predicate that has been generated. If the new predicate is unary we use the variable from the previous predicate and if it is binary in addition to the "used" variable we also create a fresh one. Furthermore in choosing the position of the "used" variable in a new binary predicate that is being created, on the average we choose to put it in the subject position 50% of the time and in the object position 50% of the time. This ensures that the former is equally likely to be in the subject as well as object position of connected triples.
4. If the query we generate is a single predicate query we make all the variables distinguished. For any other queries we make on the average 2/3rd of the variables distinguished and the rest non-distinguished.
5. We bias the introduction of a constant in a query predicate with a chance of 10%.

The above policy reflects our desire to have a simplistic query generation approach that can generate queries that are useful in measuring a system's performance. It allows us to generate queries with a decent mix of classes, properties and individuals.

Note: Every conjunct/constant added to the query makes it more selective. With a diverse data set and randomly generated queries we obtain a wide range in the degree of query selectivity.

4 Experimental Methodology

We present here our methodology of setting up an experiment for OBII and also the performance metrics that could be used for evaluation.

We feel that the most significant parameters that should be investigated are the number of ontologies, data sources, out-degree and diameter. A configuration is denoted as: nO-nD-nS where nO is number of ontologies, nD is diameter and nS is number of sources that commit to an ontology.

Metrics like Load Time, Repository Size, Query Response Time, Query Completeness and Soundness could serve as good candidates for performance evaluation [2].

Load Time: This could be calculated as the time taken to load the Semantic Web space: domain and map ontologies and the selected data sources.
Repository Size: This refers to the resulting size of the repository after loading the benchmark data into the system. Size is only measured for systems with persistent storage and is calculated as the total size of all files that constitute the repository. Instead of specifying the occupied disk space we could express it in terms of the configuration size.
Query Response Time: We recommend this to be based on the process used in database benchmarks where every query is consecutively executed on the repository for 10 times and then the average response time is calculated.
Query Completeness and Soundness: With respect to queries we say a system is complete if it generates all answers which are entailed by the knowledge base. However on the Semantic Web partial answers will also be acceptable and hence we measure the degree of completeness of each query as a percentage of the entailed answers that are returned by the system. On similar lines we measure the degree of soundness of each query as the percentage of the answers returned by the system that are actually entailed. On small data configurations, the reference set for query answers can be calculated by using state of the art DL reasoners like Racer and FaCT. For large configurations we can use partitioning techniques such as those of Guo and Heflin [12].

5 Conclusion and Future Work

Initial inspection has shown that our approach creates reasonable ontology graphs in that they are consistent with our input parameters and also have a good path length distribution from 0 to diameter. The graph topologies for some of the configurations are as follows. The nodes in the graph represent the ontologies and the links represent the mappings. A configuration is denoted by the following triple: "No. of ontologies – Outdegree – Diameter".

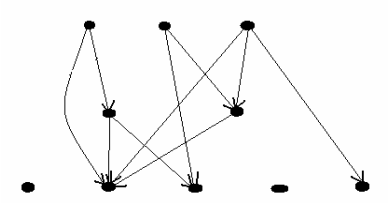


Fig. 1a. 10 -2- 2

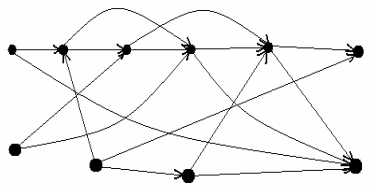


Fig. 1b. 10 – 2 - 5

There are some nodes in Fig. 1a which are disconnected from the graph. These are terminal nodes with zero in-degree. In the actual semantic web there could be such ontologies which do not map to any ontology and remain isolated.

In this paper we have discussed our approach for developing a benchmark for a complete synthetic workload. In any kind of benchmark there is some tradeoff between realism and in being simple and sufficient. Our approach is simple but could be easily generalized to support more expressive domain ontologies.

We have also introduced a new methodology for creating a graph of multiple interrelated ontologies that could be used by distributed query systems like OBII. The graph can be controlled effectively by parameters like diameter and average out-degree of the nodes. We could incorporate additional variables to represent in-degree and out-degree distributions where a few ontologies serve as "hubs" with very high out-degree and in other cases as "authorities" with a very high in-degree.

A single workload is incapable of evaluating different knowledge base systems. But our workload can be easily scaled to various configurations for the purpose of evaluation. This might encourage the development of more scalable reasoners in the near future.

It would be useful to allow the user to specify the distribution of RDFS, OWL Lite, OWL DL ontologies. Furthermore, we intend to conduct an initial experiment for comparing OWL reasoners such as Sesame, KAON2, Minerva and OWLIM.

References

1. A. Qasem, D. A. Dimitrov, J. Heflin. Efficient Selection and Integration of Data Sources for Answering Semantic Web Queries. In *New Forms of Reasoning Workshop, ISWC 2007.*
2. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
3. L. Ma, Y. Yang, Z. Qiu, G, Xie and Y. Pan. Towards A Complete OWL Ontology Benchmark. In *Proc. of the third European Semantic Web Conference.(ESWC 2006)*, 2006
4. Tempich, C. and Volz, R. Towards a benchmark for Semantic Web reasoners – an analysis of the DAML library. In *Workshop on Evaluation on Ontology Based Tools, ISWC2003*.
5. T D. Wang, B. Parsia and J. Hendler. A Survey of the Web Ontology Landscape. In *Proc. of the 5th International Semantic Web Conference. (ISWC 2006)*, 2006
6. I. Horrocks and P. Patel-Schneider. DL Systems Comparison. In Proc. Of the 1998 Description Logic Workshop (DL' 98), 1998.
7. Q. Elhaik, M-C Rousset and B. Ycart. Generating Random Benchmarks for Description Logics. In Proc. of the 1998 Description Logic Workshop (DL' 98), 1998.
8. R. Garcia-Castro and A. Gomez-Perez. A Benchmark Suite for Evaluating the Performance of the WebODE Ontology Engineering Platform. In Proc. of the 3rd International Workshop on Evaluation of Ontology-based Tools, 2004.
9. J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. In University of Michigan Technical Report CSE-TR-456-02.
10. Z. Pan, A. Qasem, J. Heflin. An Investigation into the Feasibility of the Semantic Web. In Proc. of the Twenty First National Conference on Artificial Intelligence (AAAI 2006), Boston, USA, 2006. pp. 1394-1399.
11. I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI/IAAI*, pages 399–404, 2000.
12. Guo Y. and Heflin J. Document-Centric Query Answering for the Semantic Web. 2007 IEEE/WIC/ACM International Conference on Web Intelligence (WI' 07), 2007.
13. B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of WWW2003*, Budapest, Hungary, May 2003. World Wide Web Consortium.