Rapid Benchmarking for Semantic Web Knowledge Base Systems

Conference Paper · November 2005

DOI: 10.1007/11574620_54 · Source: DBLP

CITATIONS
26

READS
45

4 authors, including:

Abir Qasem
Lehigh University
19 PUBLICATIONS **187** CITATIONS

SEE PROFILE

Jeff Heflin
Lehigh University
102 PUBLICATIONS **3,423** CITATIONS

SEE PROFILE

Rapid Benchmarking for Semantic Web Knowledge Base Systems

Sui-Yu Wang, Yuanbo Guo, Abir Qasem, and Jeff Heflin

Computer Science and Engineering Department, Lehigh University, Bethlehem, PA 18015, USA
{syw2, yug2, abq2, heflin}@cse.lehigh.edu

Abstract. We present a method for rapid development of benchmarks for Semantic Web knowledge base systems. At the core, we have a synthetic data generation approach for OWL that is scalable and models the real world data. The data-generation algorithm learns from real domain documents and generates benchmark data based on the extracted properties relevant for benchmarking. We believe that this is important because relative performance of systems will vary depending on the structure of the ontology and data used. However, due to the novelty of the Semantic Web, we rarely have sufficient data for benchmarking. Our approach helps overcome the problem of having insufficient real world data for benchmarking and allows us to develop benchmarks for a variety of domains and applications in a very time efficient manner. Based on our method, we have created a new Lehigh BibTeX Benchmark and conducted an experiment on four Semantic Web knowledge base systems. We have verified our hypothesis about the need for representative data by comparing the experimental result to that of our previous Lehigh University Benchmark. The difference in both experiments has demonstrated the influence of ontology and data on the capability and performance of the systems and thus the need of using a representative benchmark for the intended application of the systems.

1 Introduction

As the Semantic Web catches on we should expect to see a large growth of Web data that has formal semantics and an increasing number of systems that process them. Now that OWL is a W3C recommendation it is quite foreseeable that a fair share of those data will be marked up in OWL and we will have a variety of tools that will process these knowledge bases (KB). We should also see a lot of companies attempting to adapt the technology but finding it very difficult to choose the right tool. Historically this has been the case for various "killer" technologies like databases and object oriented systems. Companies have invested millions of dollar in technologies that eventually had to be discarded [3]. It would be prudent for us in the Semantic Web community to have tools and techniques ready for evaluating these emergent systems, so we are not caught off guard and benefit from a proactive strategy. It is therefore critical that we explore various approaches for evaluating these KB processing tools in an objective and practical manner.

Benchmarking has been a powerful tool for evaluation and comparison of computer systems. However, benchmarking of Semantic Web KB systems is challenging

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 758 – 772, 2005.

due to a) the wide variety of types and sizes of KBs b) the difference in reasoning tasks involved and c) the breadth of the application domains. First, if the data is large, scalability and efficiency become crucial issues. Second, the system must provide sufficient reasoning capabilities to support the semantic requirements of the application. However, increased reasoning capability usually means an increase in query response time as well. An important question is how well existing systems support these conflicting requirements. Finally, different domains and their associated applications may place emphasis on different requirements. As such, any data that is used in the benchmark has to be a good representative of the domain.

In our previous work [12, 13] we have considered the first two issues. This resulted in the Lehigh University Benchmark (LUBM). LUBM is intended to evaluate the performance of OWL repositories with respect to extensional queries over a large dataset that commits to a single realistic ontology. It consists of the ontology, customizable synthetic data, a set of test queries, and several performance metrics. The main features of the benchmark include simulated data for the university domain, and a repeatable data set that can be scaled to an arbitrary size.

In this paper we address the third issue of benchmarking Semantic Web KB systems with respect to a given domain. The synthetic data generator for LUBM was bound to a particular ontology and the data was generated by rules specified a priori based on subject matter expert's knowledge of the domain. To extend the benchmark over different domains, we have to be capable of generating synthetic data of different ontologies due to insufficient quantities of real data at the current stage of the Semantic Web development. We present a probabilistic model that, given representative data of some domain, can capture the properties of the data and generate synthetic data that has similar properties. To the best of our knowledge, this is the first work to model Semantic Web knowledge bases.

In our previous LUBM experiment [13] we have evaluated four KB systems for the Semantic Web from several different aspects. We evaluated two memory-based systems (memory-based Sesame and OWLJessKB) and two systems with persistent storage (database-based Sesame and DLDB-OWL).

To evaluate our new approach, we have created the Lehigh BibTeX Benchmark (LBBM) and used this benchmark to evaluate the same systems. We are interested not only in testing the systems against the new benchmark but also in seeing the potential difference in the performances of the systems between the two benchmarks.

Finally, to evaluate the performance of the LBBM, we collected a fair amount of OWL data, randomly selected a subset from the data, and used the subset as the training file for the data-generator. The synthetic data, the original data, and two other sets of data (one generated completely randomly by only looking into the ontology and one generated based on the authors' knowledge about what the data is like) are compared by using them to benchmark the memory-based Sesame system. The results showed that although the synthetic data still performs differently from the original data in some queries, the synthetic data always outperforms the completely random dataset.

We believe our approach to data generation for OWL KB system benchmarks has three distinct advantages over a more static data generation approach. First, it reduces the benchmark development time drastically. Second, it allows the same systems to be tested against KBs that commit to different ontologies, because data sets for new

ontologies can be quickly generated. A third advantage is more representative benchmark data, because it is statistically based on actual data, as opposed to the developer's knowledge of the domain. We presented a comparison between benchmarks of synthetic data and the real data and showed that the performances are very similar.

In Section 2 of the paper we describe the new data generation approach in detail and discuss its validity. In Section 3 we describe the LBBM experiment and compare it to the LUBM. In Section 4 we present the experiment that evaluates our new data generation technique. In Section 5 we talk about related work and in Section 6 we conclude.

2 Using a Learned Probabilistic Model to Generate Data

The main idea behind our approach is to extract properties relevant for benchmarking from real world data, and then use them to guide the generation of the synthetic data. In this section we first define terminologies and notations that are used in formalizing the problem, then describe the property-learning algorithm. Following that we describe a Monte Carlo algorithm that utilizes the discovered properties to generate synthetic data.

2.1 Pattern-Extraction Problem

The property-discovering process is motivated by the desire to solve the problem of not having enough real-world data for benchmarking a semantic web KB system. In a real-world OWL dataset, there are governing rules behind the generation of the data that are not observable from the ontology itself. Some parts of the ontology may be used more frequently than others. Take the BibTeX data for example, while assertions like each journal paper must have at least one author can be defined in the ontology, the probability that a paper has three authors cannot be obtained by simply looking at the ontology. We wanted to discover these properties, and use them to synthetically generate data that is a legitimate substitute for the real data. This is especially critical since we still do not have enough real OWL data on the Web, but at the same time we need sufficient and credible data to develop effective benchmarks.

Given representative actual data of some domain, the probabilistic modeling of the data can capture the features important for benchmarking in the data. We tried only to capture the statistical features of the data concerning the classes and properties. The actual content, or values, of the triples are filled with strings with similar length to that of the original file. Such simplification largely reduces the complexity of the tool, while preserving the performance of the benchmark using the synthetic data.

We assume that each individual only belongs to one most specific class. Although this is not always true, this assumption is valid for a reasonably large portion of existing Semantic Web data. In our model, an individual has a probability of being a member of a particular class, while a member of a particular class has a probability of a particular cardinality for each property. This model differs from the LUBM in that the LUBM assumes a minimum/maximum cardinality and an uniform distribution of cardinalities, while this new model can be used to simulate more complex real world distributions, thus giving the benchmark more power in dealing with different forms

of ontologies/data. We will first introduce the terminology used in the model, then present an algorithm that can extract features from the data.

We first define the predicate type(x,y) to indicate if an individual x is an instance of the class y. Let an RDF triple Tp be represented as Tp=(s,p,o), where s, p, o are the subject, predicate, and object, respectively. An individual $Ind(S_{ID})$ is then the set of RDF triples $Ind(S_{ID}) = \{Tp_1, Tp_2, \ldots, Tp_n\}$ such that for each triple Tp_i of the form $Tp_i = (s_i, p_i, o_i)$, $s_i = S_{ID}$.

We next define the *property pattern*, which plays a key role in mining patterns in the training data. Let $\boldsymbol{C}$ be the set of classes, $\boldsymbol{P}$ be the set of properties that are defined in the ontology and $\boldsymbol{G}$ be [RDF literals $\cup$ XML Schema datatypes$\cup$ $\boldsymbol{C}$]. A *property pattern Prop* is a 4-tuple $Prop = (c, p, g, \delta_{Prop})$ where: 1) $c \in \boldsymbol{C}$, 2) $p \in \boldsymbol{P}$, 3) $g \in \boldsymbol{G}$, and 4)δ_{Prop}: a probability distribution function. Also, an RDF triple Tp is said to *match* some property pattern *Prop*, *match*(*Tp*, *Prop*), iff $\text{type}(s_i, c) \wedge (p_i = p) \wedge \text{type}(o_i, g)$. The probability δ_{Prop} distribution is then defined as

$$\delta_{\text{Prop}}(n) = P\Big(\sum_{Tp \in Ind(s_{ID})} I(match(Tp, Prop)) = n \mid type(s_{ID}, c), Ind(s_{ID}) \in KB\Big) \quad (1)$$

where I is an indicator function[1], and KB denotes the pool of individuals in the training file. $\delta_{Prop}(n)$ is then the probability that there are n triples in some individual $Ind(S_{ID})$ in the training file KB that matches $Prop$. $\delta_{Prop}(n)$ is then describing how likely for an individual $Ind(S_{ID})$ to have n triples that matches the property pattern $Prop$.

A *property pattern set* for a class c, $ppSet(c)$, is the set of property patterns $ppSet(c) = \{Prop_1, Prop_2, \ldots, Prop_m\}$, such that $c_i = c, \forall i, i = 1,\ldots,m$. The property pattern set denotes all the property patterns we discovered for the individuals of the class c, thus is the basis for generating data about class c. When generating an individual, the property pattern set determines the kind of contents/triples the individual should have.

The synthetic data generation is divided into two phases, the property-discovering phase, and the data generation phase. We now describe the procedure of discovering properties in the training data. The knowledge of the algorithm about the training file is denoted as Γ, a collection of property pattern sets. The algorithm goes through the training data only once. Initially the algorithm has no knowledge about the training file, $\Gamma = \varnothing$. As it goes through the data on the basis of individuals, it will either 1) create new property pattern set of the class c based on the information/individuals encountered so far, if there is no property pattern set of the class c in its knowledge, or 2) update its current knowledge. Then the collected information determines the following parameters: 1) h, the number of property pattern sets, 2) $\Gamma = \{ppSet(c_1), ppSet(c_2), \ldots, ppSet(c_h)\}$ such that $c_i \neq c_j, i \neq j$, 3) a set of values $\{\tau_1, \tau_2, \ldots, \tau_h\}$, where. $\sum_{k=1}^{j} \tau_k = 1$. τ_k is defined to be the proportion of individuals of the class c_k. The algorithm is shown in Fig 1.

As the algorithm goes through the data, it will continuously update its *knowledge* about the data, where the knowledge in the form of property pattern sets for different

[1] The indicator function has value 1 if the event is true, and zero otherwise.

classes, Γ, and the corresponding *Count_τ(k),* the number of individuals of class c_k in *KB*. τ_k is then obtained by normalizing *Count_τ(k)* at the end of the algorithm. In practice, we define an additional function, *Count_δProp*(*n*), which can be viewed as a histogram, with x-axis being *n*, and y axis being the number of individuals of the class *c* that matches *Prop*. δ*Prop*(*n*) is obtained by performing normalization on *Count_δProp*(*n*) after all data has been processed.

Initial Condition: Γ = ∅ (h=0)
For all individuals *Ind(SIDi), i* =1, 2, …, u, with triples $Tp_j = (s_{ij}, p_{ij}, o_{ij}) \in Ind(S_{IDi})$
If *there are triples in the individual that match property patterns in current knowledge*, if ∃*Prop=(ck,p,g,* δ*Prop)* s.t. *match*(*Tpj*, *Prop*):
Update Prop: check *n*, the number of triples in current individual *Ind(SIDi)* that matches *Prop*, then increase the value of *Count_*δ*Prop* (*n*) by one.
Else
***Initialize Prop*:** generate a new instance of property pattern *Prop*=(*ck*, *p*, *g*, δ*Prop*) s.t. type(s_{ij},c_k) ∧ type(o_{ij},*g*), with *Count_*δ*Prop* (*n*) =1 where *n* is the number of triples in current individual *Ind(SIDi)* that matches *Prop*, zero elsewhere.
If *there is property pattern set of the class ck in current knowledge,* ∃*ppSet*(ck) s.t. type(s_{ij},c_k):
$ppSet(c_k) \leftarrow ppSet(c_k) \cup Prop$
Else
$h \leftarrow h+1$, $\Gamma \leftarrow \Gamma \cup \{ppSet(c_k)\}$ where $ppSet(c_k) = \{prop\}$
$Count\_\tau(k) \leftarrow Count\_\tau(k)+1$

***Normalize*:**
For all property patterns: $\delta_{\Pr op}(i) = Count\_\delta_{\Pr op}(i) \Big/ \sum_{j=0}^{\infty} Count\_\delta_{\Pr op}(j), i = 1,...,\infty$
For all *i=1,…,h*: $\tau_i = Count\_\tau(i) \Big/ \sum_{j=1}^{h} Count\_\tau(j)$

Fig. 1. Algorithm *Extract* for extracting property patterns from training file

2.2 The Monte Carlo Data Generation Algorithm

The algorithm presented in this section is capable of generating synthetic data that has a structure similar to that of the training data. Inherited from the nature of Monte Carlo methods [20], this is a scalable algorithm that can generate files of arbitrary sizes that have similar properties to that of the training data. The algorithm will initialize the "framework" of the synthetic data first, that is, it first generates a set of individuals and assigns a class to each, then the algorithm goes on to generate the properties and corresponding vales to each individual. Define Rand(χ) to be a random number generator with seed χ that generates random number between [0, 1]. Let the function F(x) be the *cumulative distribution function* of δ:

$$F(x) = \sum_{i=0}^{x} \delta(i) \tag{2}$$

Note that in practice, the range of i such that $\delta(i) > 0$ is finite, that is, $\delta(i) = 0$ $\forall i > \varepsilon$, where ε is some threshold. Let $\Pi = \{ \pi_1, \pi_2, ..., \pi_h \}$ be the set of generated individuals, where π_i is the set of individuals generated according to the distribution of the property pattern set *ppSet(ci)*.

Initial conditions: $\Pi = \varnothing$, $i = 1,\ldots,h$.

Given the total number of desired individuals λ in the document to be generated, initialize the set of individuals according to the desired number of individuals. These individuals only have id and classes assigned, without detail of properties and values assigned yet.

For all individuals
  For all the property patterns $Prop_{ij}(c_i, p_{ij}, g_{ij})$ of the class c_i
    Find y such that $Rand(\chi_{ij}) = F_{ij}(y)$
      Generate y RDF triples according to $Prop_{ij}$, if $g_{ij} \in$ C, then randomly select y members from π_k, type(π_k, g_{ij}), as the objects. Otherwise generate the objects such that they are random values of type g_{ij}.

Fig. 2. Algorithm *Generate* for generating synthetic data

Note that we skip the details of generating the values of the properties. For benchmarking purposes, the content itself is less important as long as the generated content has similar "features" to the training data. The feature can refer to the length of the string, the range of the integer value, etc.

2.3 Probabilistic Model Versus the Power Law

Since we are trying to generate data that resembles the real world data, a key question is how representative our synthesized data is. The first claim of our tool is its scalability. One may argue that real-world data should be self-similar, that is, some complex patterns emerge when the size of the data increases. We argue the legitimacy of the scalability in the fundamental inapplicability of the power law on our approach.

The power-law specifies a relationship between two variables such that one is proportion to the power of the other [11]. This law has been shown to hold in many different kinds of network. Take the World-Wide-Web for example, the number of links to a certain node can be predicted by the rank[2] of the node. However, in a structural dataset like OWL, the distribution of the links is often constrained by the type of link it is. Take the Bibtex domain for example, although the out- link of the ObjectProperty "editor" of a certain publication may vary over a wide range, most of them have 1-3 editors. If the power-law applies, then as the size of the file increases, the maximum number of editors will also increase, without limit, which is clearly not true. No matter how big the file size is, the maximum number of authors is unlikely to exceed 5 people. Still, there could be links where the power law is valid. In the FOAF

[2] The index in the order of decreasing measurements (in-degree, out-degree, etc.).

domain, the number of acquaintances one has, as the community of FOAF increases, could also increase without boundary. In such cases, the data generation is scalable to the size of the real world data as long as the training file is a representative subset of the real world data. When we say representative subset, we mean when the sample is being drawn, we collect all the information about the sample, that is, when an individual is being collected to the subset, all the links it has are also taken into the subset.

3 LBBM: A New Benchmark

The data generation method described in previous section allows us to rapidly develop a benchmark that is specific to a domain and then conduct experiments based on it. We highlight the workflow in the following. First we choose an ontology that represents the domain in question. Then we collect sample data and create synthetic test data that commits to that ontology by utilizing the new data generation approach. We use this approach to generate a new benchmark for the Bibtex domain. At the end we compare the LBBM to our previous approach, the LUBM.

3.1 Lehigh Bibtex Benchmark (LBBM)

To test drive our approach, we have used it to create a new benchmark named Lehigh BibTeX Benchmark (LBBM). We have used the Lehigh University BibTeX ontology as our domain definition [18]. This ontology is a modified version of the Bibtex ontology 0.1 by MIT [4]. It is important to note our rationale behind choosing this particular ontology. Tempich and Volz [23] have done some preliminary work towards a benchmark for Semantic Web reasoners. Though their benchmark is still under construction, they analyze the publicly available ontologies and report them to be clustered into three categories: description logic-style, database schema-like, and terminological ontologies.

The BibTeX ontology is expressed in OWL Lite and consists of 28 classes and 80 properties, half of which are datatype properties. According to the classification of [23], the ontology is more of a database schema-like ontology. The classes and properties in the BibTeX ontology used by LBBM correspond to entries and fields in BibTeX respectively.

We designed test queries as realistic as possible against the benchmark data. Moreover, we choose the queries that cover a range of types in terms of input size, selectivity, complexity, required hierarchy information, and required OWL inference. We designed twelve test queries in LBBM. A complete list of the queries can be found in [24]. Because of the nature of the ontology and the data, most of queries are RDF style queries and only one of them assumes OWL inference.

In order to acquire a suitable set of training data, we take advantage of the fact that there are plenty of BibTeX files on the Web, and convert them into OWL format by the Java BibTeX to RDF Converter 1.0 developed by the University of Karlsruhe [15] and the perl DAML+OIL conversion script by BBN. This resulted in a 2.4 MB OWL file which was used as our training data. We then used our tool to identify patterns from the training data and generate synthetic benchmark data based on those patterns.

In order to test a system in the benchmark framework, we wrap the system with an instantiation of the predefined interface between the benchmark test module and the target system. This involves the implementation of Java APIs for operations such as loading and query execution.

Query response time is collected in the way based on the process used in database benchmarks [5, 6, 9, 22]. To account for caching, each query is executed for ten times consecutively and the average time is computed. The elapsed time is counted from when the query is issued till the result set is returned and traversed from the beginning to the end. We also measure query completeness and soundness. We do not measure them with a coarse distinction of yes or no. Instead, we measure the degree of completeness as the percentage of the entailed answers that are returned by the system, and the degree of soundness as the percentage of the answers returned by the system that are actually entailed [13].

3.2 An Experiment Using LBBM

We have conducted an experiment based on LBBM. Although newer systems with improved performance had been introduced, for the sake of comparison, we have evaluated the same systems as in our previous LUBM experiment, including DLDB-OWL (04-03-29 release), Sesame (both main memory-based and database-based, v1.0), and OWLJessKB (04-02-23 release). First we briefly introduce the reasoning features of these systems.

DLDB-OWL [21] loosely couples a relational database system (MS Access) with a description logic reasoning reasoner (FaCT). Sesame [7] is a storing and querying facility for RDF and RDF Schema (RDFS). Sesame is an incomplete reasoner for OWL Lite. We evaluate two implementations of Sesame, i.e., main memory-based and database-based. For brevity, we hereinafter refer to them as Sesame-DB and Sesame-Memory respectively. OWLJessKB [17] is a memory-based reasoning tool for OWL implemented as a set of JESS production rules.

We have tested the above systems against four datasets, the largest one consisting of 320 OWL files totaling 189MB and containing over 2,600,000 triples. The test queries are expressed in RQL [16], a KIF-like language and JESS and issued to Sesame, DLDB-OWL and OWLJessKB respectively. The experiment is conducted on a PC with following environment 1) 1.80GHz Pentium 4 CPU, 2) 256MB of RAM, 3) 80GB of hard disk, 4) Windows XP Professional OS, and 5) Java SDK 1.4.1, 512MB of max heap size.

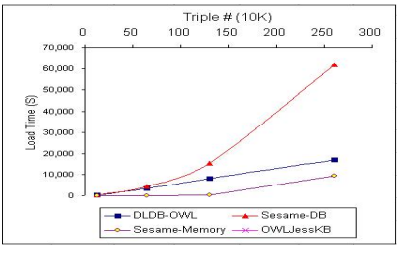

Fig. 3. Load Time

Table 1 shows the load time of each system and the consequent repository sizes of DLDB-OWL and Sesame-DB. Fig. 3 depicts how the load time grows as the data size increases. It needs to be mentioned that OWLJessKB has failed to load the smallest dataset even after we increased the max heap size to 1GB. Consequently we will not include OWLJessKB in the subsequent discussion except for query completeness and soundness.

Impressively, Sesame-Memory could load all of the datasets, despite that it nearly reached the memory limitations when loading the largest dataset. Furthermore, it is the fastest system to load every dataset. As for the two systems with secondary storage, DLDB-OWL could obviously scale better than Sesame-DB. We will return to this in next section.

Fig. 4 compares the selected query response time between DLDB-OWL and Sesame systems. A complete set of results can be found at [24]. Sesame-Memory performed the best in querying too. It was the fastest system in answering the queries upon the four of the datasets, with a few exceptions at the largest dataset when its performance went down drastically (e.g. Queries 4, 10, and 12). We believe this was caused by frequent page swapping due to main memory limitations. For the other two systems, Sesame-DB was faster than DLDB-OWL to answer almost all the queries. Furthermore, for most of the queries, Sesame-DB has showed no proportional increase in the response time as the data size grows.

Table 1. Load time and repository sizes

	Data Set	Data Size (MB)	Triple #	Load Time	Repository Size (KB)
DLDB-OWL	LBBM(1)	9.46	130,138	00:07:44	19,005
Sesame-DB				00:07:14	54,788
Sesame-Memory				00:00:12	-
OWLJessKB				failed	-
DLDB-OWL	LBBM(2)	47.3	651,392	01:00:03	86,606
Sesame-DB				01:13:18	261,390
Sesame-Memory				00:01:09	-
DLDB-OWL	LBBM(3)	94.8	1,302,952	02:14:56	172,130
Sesame-DB				04:17:57	521,110
Sesame-Memory				00:02:54	-
DLDB-OWL	LBBM(4)	189	2,606,739	4:43:18	342,458
Sesame-DB				17:09:55	1,039,991
Sesame-Memory				02:37:26	-

Next we look at the query completeness and soundness of each system. In order to get a flavor of its capability in this aspect, we have tested OWLJessKB on a single file extracted from the test dataset. It turned out that all the systems were sound in answering the twelve queries. However, they differed from each other in query completeness.

Specifically, all of them could answer Query 1 through Query 9 as well as Query 12 completely. However, while Sesame and OWLJessKB were complete, DLDB-OWL

could only find partial answers (about 98%) for Query11 since it does not make inferences about the domain of properties. Moreover, as expected, OWLJessKB was the only system that could answer Query10, which requires *owl:inverseOf* inference.

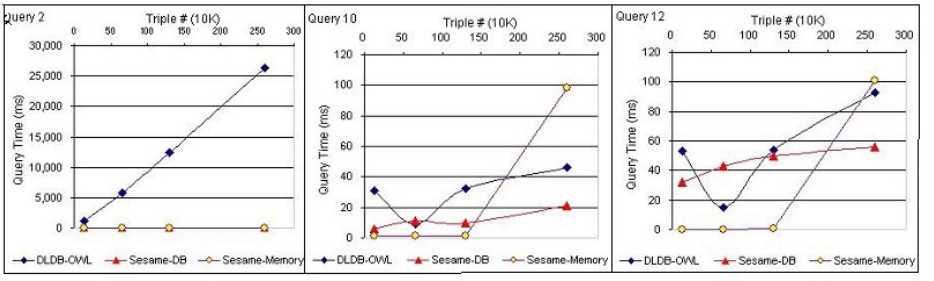

Fig. 4. Comparison between different systems for selected queries

3.3 Comparison to the LUBM Experiment

As noted, the same systems have previously been evaluated with our other benchmark: LUBM. The scales of the datasets used herein are also close to the first four datasets in the LUBM experiment respectively. The major difference between the two benchmarks lies in the ontology, the data model, and the test queries. We are interested to see how these have influenced the performance of each system.

In terms of loading, DLDB-OWL was the system showing the best scalability in both experiments. Large scales of data remain a big challenge for OWLJessKB. It could load only the smallest dataset in the LUBM experiment after spending much longer time than the others. In this experiment, it was still slow in loading and could not even handle the smallest dataset.

In contrast, Sesame systems have performed quite better in this experiment. Sesame-Memory succeeded to load a considerably larger size of data. Sesame-DB still faces the scalability problem, but it is much less prominent than in the LUBM experiment. We have considered Sesame's inability to scale in loading to relate to two reasons. One is its use of forward-chaining inference and the overhead of recording the dependency among statements. The other reason is the time cost of ID management for resources and literals during loading. (Readers are referred to [13] for a detailed discussion.). Compared to LUBM, LBBM's ontology and data are more simplistic and as a result, there are less inferred statements by Sesame during loading. In addition, there are fewer unique literals in the data. We think these could account for the significant scalability improvement of Sesame.

It turned out that the difference between both benchmarks have also influenced the query performance, especially for Sesame. In the LUBM experiment, Sesame-DB was very slow in answering some of the queries (particularly those do not contain a specific URI as the subject or object in the statements and have a complex pattern of relationships among the individuals concerned). Given the simplicity of the BibTeX ontology and the model of the test data, we have not found any appropriate queries

representing complex connections between individuals. For the present queries in LBBM, Sesame was able to answer them rather quickly.

The ontology and test data used in our LBBM, although described in OWL language, is essentially an RDFS ontology. The case is similar for the test queries. It is known that Sesame is developed as an RDF repository. We believe it is more optimized for processing RDF-style data and queries than systems like DLDB-OWL and OWLJessKB. The result has clearly showed that Sesame has become a better choice than the other two systems for the applications in a similar domain to what our LBBM represents. Overall, the experiment has verified that the ontology and instance data could make a difference in both the capability and performance of the same system. This demonstrates the need for using a benchmark that resembles the domain in which the evaluated systems are intended to be applied.

4 Evaluation of the Data Generation Technique

We claimed our tool is capable of generating a legitimate substitute to the real data. To examine this assumption, we take subsets of the original file, generate synthetic data using the subset as the training file, and compare the benchmark result of the synthetic file to that of the original file. A subset was derived by randomly selecting an individual and taking it along with all neighboring individuals two hops away from it into the subset. The definition of neighboring individuals, however, is a tricky question. Consider the data as a huge directed graph. The resulting sets of neighbors are different when considering neighbors via link direction or just via links. We take both subset selection schemes into consideration.

In additional to the synthetic data and the original data, we also generated two other sets of data for comparison. They both use a tool that can parse a given ontology, let user specify the kind of triples and the min/max cardinality of that kind of triple, let user specify min/max number of individuals of a certain class, and generate data according to the given parameters in a uniform distribution. The first set of data has parameters assigned by a domain expert (based on knowledge about the kinds of triples and approximate ratio between different classes of individuals). The second set is generated by assigning the parameters according to a series of random numbers.

The size of the original file is one million. We have subsets of sizes 125, 250, 500, 1000, 5000, 10000, 20000, and 50000 triples. For each size, we have a subset that includes directional neighbors and one that includes neighbors regardless of directions. These subsets are used as training files to generate a set of synthetic data consisting of one million triples. The two data sets generated according to domain knowledge and completely randomly also consist of one million triples.

The system for evaluation is the memory-based Sesame-DB. Eleven queries were designed so that the first half of the queries are bound to have results in almost any kind of data, while the second part is more likely to have answers for more realistic data. The set of queries can be found at [24]. The experiment is conducted on a PowerBook G4 with environment as follows: 1) 1.0GHz PowerPC G4 CPU; 2) 512MB of RAM; 60GB of hard disk; 3) Mac OS X 10.3.9; 4)Java SDK 1.4.2.

Fig 5. shows the result of the query time verses the size of the training files for selected queries. In the legends, “orig” refers to the result from the original Bibtex data,

"man" refers to the data generated with domain knowledge, "rand" refers to the set of data generated completely randomly, "dir" refers to the set of results with training files subsetted with direction of the links considered, while "non-dir" refers to those that does not. Other queries have either close-to-zero query time or zero results thus are less valuable to present here. In Fig 5. Q5, the performance from the completely random dataset is omitted, because it performs so bad that it will flatten out other information in the graph. The query time for query 5 in the completely random dataset is 330493ms.

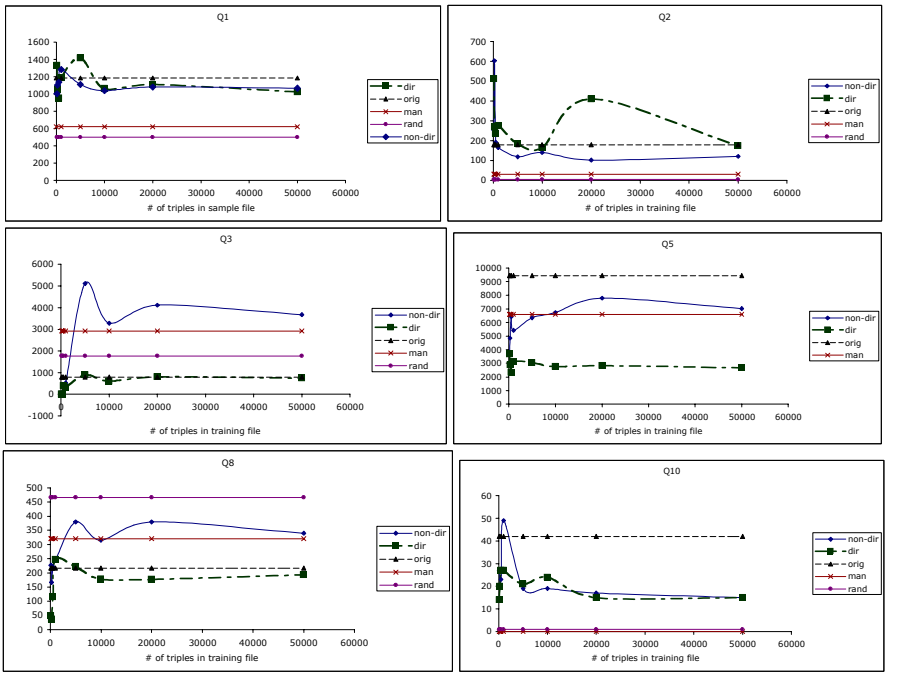

Fig. 5. Query Time verse different sizes of training file in selected queries

For Fig 5 Q3, the data from non-directional scheme returns far more results than the original file, while the data from directional scheme performs reasonably well. This query asks for all InProceedings that have at least one author and one editor. One possible explanation is that while almost all InProceedings have at least one author, the number of InProceedings having editors are relatively fewer. At the same time, the probability that someone is an editor isn't uniform either: some people are more likely to be editors for several publications. In the non-directional selection scheme, as long as we selected a publication with editors, or selected someone being editor, the probability of the data two hops away having the property editor is higher than normal.

In Fig 5 Q5, the non-directional selection scheme outperforms the directional one. The query is asking for people with more than two publications. This might be explained by that when an individual is selected, it is less likely for the directional

scheme to get all the publication from someone because the link's direction is from publication to author/person. Fig 5 Q8 might be explained similar to that in Q3. This query asks for publication with publisher specified. For the non-directional scheme, it is likely to select a bunch of individuals from the same source Bibtex file, thus more likely than normal to have the attribute publisher.

From the figures we can see that the benchmark result of the synthetic data given representative training data, can be almost identical to the original file at best, and still better than completely random data at worst. Furthermore, these results can be achieved with a training set of only 10000 triples. The experiment shows that the synthetic data generator presented here can be a reasonable substitute for benchmark systems if insufficient data is available, or if the user wishes to create a benchmark without the pain of collecting a large amount of data.

Ideally, we would like to repeat the experiment on other ontologies. However, the lack of sufficient real-world data had made this impossible. The most likely next candidate for such experiments is the FOAF data. However, current available FOAF data can only contribute about 60000 triples, which is only a fraction to the size of the current experiment. The difference in the magnitude in the size of the two datasets will make the comparison meaningless.

5 Related Work

The benchmark data generation tries to discover the patterns in the real world data and reflect them in the synthetic data generation. Our work is influenced by the association rule mining in the data-mining research [1]. A classic example of association rule mining is for the supermarket retailers to try to identify association relationships between the items bought in one customer transaction. For example discover if milk is often bought together with bread. We take advantage of the structural nature of the Semantic Web data. The definition of a transaction is analogous to that of the individual in our approach, and patterns are found within the transaction/individual.

There are some other works that exploit similar techniques for the Semantic Web but for different purposes from ours. For example, Maedche and Staab [19] have studied ontology learning for the Semantic Web. They make use of a modification of the generalized association rule learning algorithm for discovering properties between classes.

Our work on benchmarking Semantic Web KB systems has emphasized on the support of evaluating the systems with respect to large amount of data and extensional queries upon the data. This makes our work different from others. For instance, Alexaki et al. [2] have developed some benchmark queries for RDF, however, these are mostly intensional queries. Some attempts have been done by Elhaik et al [10] and Horrocks and Patel-Schneider [14] to benchmark description logic systems. The emphasis of this work is to evaluate the reasoning algorithms in terms of the tradeoff between expressiveness and tractability in description logic. Our work is not a description logic benchmark. Moreover, unlike our approach, such benchmark data generation as used in [10] does not take account of simulating the real world data. Lastly, the Web Ontology Working Group provides a set of OWL test cases [8]. They are intended to provide examples for, and clarification of, the normative definition of OWL

and focus on the completeness and soundness with respect to individual features. Different from our benchmarks, they are not suitable for the evaluation of scalability.

6 Conclusion

In this paper, we have considered the issue of rapid development of benchmarks for Semantic Web knowledge base systems. In our previous work, we have used the Lehigh University Benchmark (LUBM) to evaluate several contemporary systems. LUBM is bound to a specific ontology and its data generation is based on statically encoded rules. The university ontology that we used in LUBM is categorized as a description logic-style ontology. The benchmark represents certain classes of Semantic Web applications but not all. It is difficult and inefficient if not completely impossible to generate benchmarks that will cover the wide variety of Semantic Web applications that are possible. In light of this, in this work, we have moved forward by introducing a method for generating benchmark data of any chosen domain in a very time efficient manner. We have achieved this by developing a data generation approach that does not depend on statically encoded rules. This method constructs a probabilistic model that can extract statistical features from real world data that are important for benchmarking. A Monte Carlo algorithm is used to generate synthetic data that have similar features to that of the real world data based on the model constructed. Experiments have been conducted to show that the benchmark using the synthetic data has a performance very similar to the one that uses real world data at best, and still outperforms the data generated without knowledge of the real-world data at worst. We have shown that our data generation provides a reasonable substitute for large quantity of real world data.

We have used this new approach to create another benchmark called Lehigh BibTeX Benchmark (LBBM) within a considerably short period. LBBM is different from LUBM in the sense that it represents the use of a database schema-like ontology and more RDF-style data and queries. We have used this new benchmark to evaluate two main memory-based systems (memory-based Sesame and OWLJessKB) and two systems with persistent storage (database-based Sesame and DLDB-OWL). They are the same systems in our previous LUBM experiment. We compared both experiments and pointed out the difference between their results. The experiment has verified that the characteristics of ontology and data used in the benchmark can make a difference in the evaluation of the systems. We argue this demonstrates the necessity of choosing a representative benchmark for the intended application of the systems and thus the need for a variety of Semantic Web knowledge base benchmarks. We intend for our approach presented herein to play a promotional role in this regard.

Acknowledgements

This material is based on work supported by the National Science Foundation under account No. IIS-0346963. We thank Paul Koget for suggestions that synthetic data generators could be learned from existing data.

References

[1] Agrawal, R. Imielinski, T., and Swamy, A. Mining association rules between sets of items in large databases. In Proc, of ACM SIGMOD Intl. Conf. on Management of Data, May 1993.

[2] Alexaki, S. et al. On Storing Voluminous RDF Description: The case of Web Portal Catalogs. In Proc. of the 4th International Workshop on the Web and Databases, 2001.

[3] Beall, S and Hodges, R. Application & Systems Program Development: Software Directory Columns. Gartner Corporation Technical Report, 1997.

[4] Bibtex Definition in OWL Version 0.1. http://www.visus.mit.edu/bibtex/0.1/

[5] Bitton, D., DeWitt, D., and Turbyfill, C. Benchmarking Database Systems, a Systematic Approach. In Proc. of the 9th International Conference on Very Large Data Bases, 1983.

[6] Bitton, D. and Turbyfill, C. A Retrospective on the Wisconsin Benchmark. In Readings in Database Systems, Second Edition, 1994.

[7] Broekstra, J. and Kampman, A. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proc. of ISWC2002.

[8] Carroll, J.J. and Roo, J.D. ed. OWL Web Ontology Test Cases. http://www.w3.org/TR/2004/REC-owl-test-20040210/

[9] Cattell, R.G.G. An Engineering Database Benchmark. In Readings in Database Systems, Second Edition, 1994.

[10] Elhaik, Q, Rousset, M.C., and Ycart, B. Generating Random Benchmarks for Description Logics. In Proc. of DL' 98.

[11] Faloutsos M. and Faloutsos, P. and Faloutsos, C.. On Power-law Relationships of the Internet Topology. Pages 251-262, SIGCOMM '99

[12] Guo, Y., Heflin, J., and Pan, Z. Benchmarking DAML+OIL Repositories. In Proc. of ISWC2003.

[13] Guo, Y., Pan, Z. and Heflin, J. LUBM: A Benchmark for OWL Knowledge Base Systems. In Journal of Web Semantics, Vol 3, Issue 2, 2005

[14] Horrocks, I. and Patel-Schneider, P. DL Systems Comparison. In Proc. of DL' 98.

[15] Java BibTeX-To-RDF Converter. http://www.aifb.uni-karlsruhe.de/WBS/pha/bib/

[16] Karvounarakis, G. et al. Querying Community Web Portals. http://www.ics.forth.gr/proj/isst/RDF/RQL/rql.pdf

[17] Kopena, J.B. and Regli, W.C. DAMLJessKB: A Tool for Reasoning with the Semantic Web. In Proc. of ISWC2003.

[18] Lehigh University Bibtex Ontology. http://www.cse.lehigh.edu/~syw/bib-bench.owl

[19] Maedche, A and Staab, S. Ontology learning for the semantic web. IEEE Intelligent Systems, 16(2): 72-79. 2001.

[20] Manno, I. Introduction to the Monte Carlo Method. Budapest, Hungary: Akadémiai Kiadó, 1999.

[21] Pan, Z. and Heflin, J. DLDB: Extending Relational Databases to Support Semantic Web Queries. In Workshop on Practical and Scalable Semantic Systems, ISWC2003.

[22] Stonebraker, M. et al. The SEQUIOA 2000 Storage Benchmark. In Readings in Database Systems, Second Edition. 1994.

[23] Tempich, C. and Volz, R. Towards a benchmark for Semantic Web reasoners–an analysis of the DAML ontology library. In Workshop on Evaluation on Ontology-based Tools, ISWC2003.

[24] Wang, Sui-Yu, Guo, Yuanbo, Qasem, Abir, and Heflin, J. Rapid Benchmaring for Semantic Web Knowledge Base Systems, Technical Report LU-CSE-05-026, Dept. of Computer Science and Engineering, Lehigh University, 2005.