Generating RDF for Application Testing[*]

Daniel Blum and Sara Cohen
{daniel.blum@mail,sara@cs}.huji.ac.il

School of Computer Science and Engineering
The Hebrew University of Jerusalem

Abstract. Application testing is a critical component of application development. Testing of Semantic Web applications requires large RDF datasets, conforming to an expected form or schema, and preferably, to an expected data distribution. Finding such datasets often proves impossible, while generating input datasets is often cumbersome. The GRR (Generating Random RDF) system is a convenient, yet powerful, tool for generating random RDF, based on a SPARQL-like syntax. In this poster and demo, we show how large datasets can be easily generated using intuitive commands.

1 Introduction

Testing is a critical step in application development. For Semantic Web applications, testing is a challenge due to both the large volume of input data needed, and the intricate format that this data must have. While many Semantic Web applications focus on varied and unexpected types of data, there are also many others that target specific domains. For such applications, to be useful, datasets used should have at least two properties:

1. The data structure should have the expected structure needed for the target application (e.g., conform to a specific RDF schema).
2. The data should match the expected data distribution of the target application.

Currently, there are several distinct sources for RDF datasets. First, there are *downloadable RDF datasets* that can be found on the web, e.g., Barton libraries, UniProt catalog sequence, and WordNet. RDF Benchmarks, which include both large datasets and sample queries, have also been developed, e.g., the Lehigh University Benchmark (LUBM) [4] (which generates data about universities), the SP^{2}Bench Benchmark [7] (which provides DBLP-style data) and the Berlin SPARQL Benchmark [1] (which is built around an e-commerce use case). Such downloadable RDF datasets are usually an excellent choice when testing the efficiency of an *RDF storage system*. However, they will not be suitable for experimentation and analysis of a particular *RDF application*. Specifically, since these datasets are built for a single given scenario, they may not have either of the two specified properties, for the application at hand.

Data generators are another source for datasets. A data generator is a program that generates data according to user constraints. As such, data generators are usually more flexible than benchmarks. Unfortunately, there are few data generators available for

[*] This work was partially supported by GIF Grant (2201-1880.6/2008).

RDF (SIMILE [8], RBench [6]) and none of these programs can produce data that conforms to a specific given structure, and thus, again, will not have the specified properties.

In this demo, we present the GRR (Generating Random RDF) system for generating RDF that satisfies both desirable properties given above. Thus, GRR is *not* a benchmark system, but rather, a system to use for Semantic Web application testing. Using intuitive data generation commands with a SPARQL-like syntax, GRR can produce data with a complex graph structure, as well as draw the data values from desirable domains. Data generation commands are translated into a series of SPARQL queries and update commands which are applied directly to an RDF storage system.[1] A video demonstration of GRR is available online,[2] and the system is available upon request.

2 Motivating Example

As a motivating example, we discuss the problem of generating the data described in the LUBM Benchmark. Note that GRR is not limited to creating benchmark data. In our demo, we will demonstrate using GRR to generate other types of data, such as FOAF [3] (Friend of a Friend) datasets, which are used in social network applications.

LUBM [4] is a collection of data describing university classes (i.e., entities), such as departments, faculty members, students, courses, etc. These classes have a plethora of properties (i.e., relations) between them, e.g., faculty members work for departments and head departments, students take courses and are advised by faculty members, etc.

In order to capture a real-world scenario, LUBM defines interdependencies between the entities. For example, the number of students in a department is a function of the number of faculty members. Specifically, LUBM requires there to be a 1:8-14 ratio of faculty members to undergraduate students. As another example, the cardinality of a property may be specified, such as each department must have a single head of department (who must be a full professor). Properties may also be required to satisfy additional constraints, e.g., courses, taught by faculty members, must be pairwise disjoint.

In the next section, we describe the GRR data generation language, and demonstrate commands for producing LUBM benchmark data. Due to space limitations, we do not provide all commands used to reproduce LUBM. However, we note that the number of words needed in all data generation commands (in order to reproduce LUBM), is only about twice as many as used in the intuitive description of LUBM, provided by [4]!

3 Data Generation Commands

Data is generated by a sequence of *data generation commands* (*dg-commands*, for short) $c_1, \ldots, c_n$, when given as input a (possibly empty) RDF dataset R. The first command c_1 is evaluated over R, while each consecutive command c_i is evaluated over the output of the previous command c_{i-1}.

The general syntax of a single dg-command appears below. Note that square brackets are used to denote optional portions, and the "*" indicates a component that can appear any number of times.

[1] The Jena Semantic Web Framework for Java [5] is used in our implementation.

[2] http://www.cs.huji.ac.il/~danieb12/

```
(FOR (EACH | sampling-method)
    [WITH (GLOBAL DISTINCT | LOCAL DISTINCT | REPEATABLE)]
    {list of classes}
    [WHERE {list of conditions}] )*
[CREATE i-j {list of classes}]
[CONNECT {list of connections}]
```

A dg-command contains any number of FOR clauses, and then optionally a CREATE and/or CONNECT clause. Intuitively, the FOR clauses choose portions of the RDF input, the CREATE clause creates new nodes in the RDF graph, and the CONNECT clause connects nodes in the RDF graph. We require that at least one among the CREATE and CONNECT clauses be present in every dg-command. We now describe each clause, briefly. (Full language semantics appears in [2]).

- The FOR Clause: Each FOR clause defines *(1)* a query which will applied against the RDF input, as well as *(2)* a method to choose a subset of the query results. For (1), the user provides a list of classes whose instances should be chosen (similar to a SPARQL SELECT clause), as well as any conditions (similar to a SPARQL WHERE clause). The correspondence to SPARQL is not precise as we allow for certain syntactic shortcuts, which avoid explicit variable use, and make dg-commands more readable. For (2), the user defines both the method with which answers should be sampled, as well as whether the sampling process is with/without repetition.
- The CREATE Clause: The CREATE clause defines nodes that should be created. The user provides both a list of RDF classes, and a range determining how many instances of these classes should be created.[3]
- The CONNECT Clause: The CONNECT clause determines the edges that should be generated in the RDF graph, by providing a list of triples.

Several examples of dg-commands appear below. Explanations follow.

```
(c1) CREATE 1-5 {ub:Univ}

(c2) FOR EACH {ub:Univ}
       CREATE 15-25 {ub:Dept}
       CONNECT {ub:Dept ub:subOrg ub:Univ}

(c3) FOR EACH {ub:Faculty, ub:Dept}
     WHERE {ub:Faculty ub:worksFor ub:Dept}
       CREATE 8-14 {ub:Undergrad}
       CONNECT {ub:Undergrad ub:memberOf ub:Dept}

(c4) FOR EACH {ub:Dept}
         FOR 1 {ub:FullProf}
         WHERE {ub:FullProf ub:worksFor ub:Dept}
         CONNECT {ub:FullProf ub:headOf ub:Dept}
```

[3] Dg-commands do not directly define how textual (or other atomic) properties are created and associated with class instances. This information is provided in a simple auxilliary file, e.g., which associates each textual property with a sampling method or dictionary.

```
(c5) FOR 20%-20% {ub:Undergrad, ub:Dept}
     WHERE {ub:Undergrad ub:memberOf ub:Dept}
       FOR 1 {ub:Prof}
       WHERE {ub:Prof ub:memberOf ub:Dept}
       CONNECT {ub:Undergrad ub:advisor ub:Prof}

(c6) FOR EACH {ub:Undergrad}
       FOR 2-4 WITH LOCAL DISTINCT {ub:UndergradCourse}
       CONNECT {ub:Undergrad ub:takeCourse ub:UndergradCourse}

(c7) FOR EACH {foaf:Person ?p1}
       FOR 15-25 {foaf:Person ?p2} WHERE {FILTER( ?p1 != ?p2 )}
       CONNECT {?p1 foaf:knows ?p2}
```

Command c_1 creates between 1 and 5 universities, and command c_2 adds 15–25 departments as suborganizations for each university. Command c_3 iterates over all pairs of faculty members[4] and departments, and adds 8-14 students, per pair to the department (therby achieving the required 1:8-14 ratio of faculty members to undergraduates). Command c_4 chooses one full professor as the head of each department. Command c_5 adds an advisor for 20% of all undergraduates. Command c_6 assigns 2-4 courses for each undergraduate. Note the use of `WITH LOCAL DISTINCT` which ensures that the set of courses chosen per student does not contain repetition, while allowing different students to be assigned the same courses. Finally, c_7 demonstrates advanced features including variables and a filter command, to connect people (in an FOAF RDF dataset) to one another.

In our poster and demo, we will show how to recreate the LUBM benchmark using 24 dg-commands, of the style seen above. In addition, we will show how to create interesting datasets for the FOAF schema. We will also allow those interested to write their own dg-commands, which we will evaluate in GRR to create an RDF dataset.

References

1. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International Journal of Semantic Web Information Systems 5(2), 1–24 (2009)
2. Blum, D., Cohen, S.: Grr: Generating random RDF. Tech. rep., The Hebrew University of Jerusalem (2010)
3. The friend of a friend (FOAF) project. `http://www.foaf-project.org`
4. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2-3), 158–182 (2005)
5. Jena–a Semantic Web framework for Java. `http://jena.sourceforge.net`
6. RBench website. `http://139.91.183.30:9090/RDF/RBench/index.html`
7. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^{2}Bench: a SPARQL performance benchmark. In: ICDE. pp. 222–233. Shanghai, China (Mar 2009)
8. Simile website. `http://simile.mit.edu/`

[4] The faculty members were created with an additional dg-command, which was omitted due to lack of space.