BRAHMS: A WorkBench RDF Store And High Performance Memory System for Semantic Association Discovery[1]

Maciej Janik and Krys Kochut

Large Scale Distributed Information Systems (LSDIS) Lab
Department of Computer Science, University of Georgia
415 Boyd Graduate Studies Research Center
Athens, GA 30602-7404
{janik, kochut}@cs.uga.edu

Abstract. Discovery of semantic associations in Semantic Web ontologies is an important task in various analytical activities. Several query languages and storage systems have been designed and implemented for storage and retrieval of information in RDF ontologies. However, they are inadequate for semantic association discovery. In this paper we present the design and implementation of BRAHMS, an efficient RDF storage system, specifically designed to support fast semantic association discovery in large RDF bases. We present memory usage and timing results of several tests performed with BRAHMS and compare them to similar tests performed using Jena, Sesame, and Redland, three of the well-known RDF storage systems. Our results show that BRAHMS handles basic association discovery well, while the RDF query languages and even the low-level APIs in the other three tested systems are not suitable for the implementation of semantic association discovery algorithms.

1 Introduction

Semantic Web ontologies are envisioned to represent knowledge bases containing millions of entities [26] interconnected with relationships. The relationships form the foundation of the Semantic Web [28] and enable the discovery and interpretation of semantic associations existing between entities in the ontology. Although, it is known that searching for simple paths in graphs is NP-complete [16], there is a great need for software tools that allow searching for relationship paths in a reasonable time, especially if the paths are of a limited length.

A semantic association path connecting two entities, as defined in [5], is a sequence of meaningful relationships connecting the two entities. The semantic

[1] This research has been supported by the National Science Foundation Grant No. IIS-0325464 entitled "SemDIS: Discovering Complex Relationships in the Semantic Web".

association describes how the two entities relate to each other. We also say that two entities are semantically related to each other if a semantic association path exists between them. Query languages that are available for RDF bases [20] allow the specification of certain patterns of semantic associations between entities as well as expressing various restrictions on the relationships participating in associations. However, they are not designed for the discovery of semantic associations [4]. The main problem is that in the semantic association discovery neither the length of the association path nor the relations included in it or their directionality are known a priori. These features of the semantic association discovery make the current high-level RDF query languages not suitable for this purpose, as the path expressions that they can create specify relationships of fixed length and directionality.

A possible solution to this problem is in the creation of graph-based algorithms that utilize API-level graph primitives, such as the fast computation of a node neighborhood. Several RDF storage base implementations have been described in the literature and a number of such implementations are available on the Web. All of them include high-level query languages such as RQL [13], RDQL [22], SquishQL [17], and SPARQL [29]. However, only a few of them have a lower-level API suitable to operate directly on the internal graph representation structures. Implementations providing such a low-level API include Jena [15], Sesame [8] and Redland [7]. Unfortunately, all of them have certain drawbacks and limitations when it comes to discovering longer semantic associations in large ontologies.

In order to overcome some of the limitations of the current RDF store implementations, we have created BRAHMS – a workBench Rdf store And High performance Memory System, which provides a suitable basis for the implementation and testing of semantic association discovery algorithms.

2 Motivation

Imagine an analyst investigating how a person X is may be related to a person Y, based on the facts that are stated in an RDF description base. Such requirement routinely occurs in examples such as Anti-money Laundering [23], Threat Assessment [27] and Risk Assessment. Such work requires discovering association paths existing between these two people, represented as resources in the RDF base. The semantic associations are usually of unknown and variable length, and the relations that connect the intermediate resources can be of any directionality. Non-directionality is a necessary requirement, as the two resources may not be linked by a directed path, but they may be connected by a path that includes inverse relations, representing potentially vital information about the semantic linkage existing between the resources in question. The discovered association paths should be built from instance resources, as they represent facts in knowledge base. Literal values and schema types represent important and valuable information for understanding the meaning of the path, but in this case they should not be included as the building blocks of the path itself.

The discovery of short (2-3 relations) association paths is quite fast even in larger RDF graphs, due to the limited search space. Obviously, it is not the case when it

comes to finding longer paths in large graphs. The searches may take a much longer time, or be simply infeasible, as searching for simple paths in graphs is NP-Complete. On the other hand, applications such as the anti-money laundering systems [14] search for and favor longer semantic associations while operating on large datasets. Our own experience shows that with a highly optimized implementation, searching for longer semantic associations, even in bigger graphs, may be done in almost real time.

One of the possible solutions is to use a system that keeps the whole graph structure in main memory, since accessing the disk-based or remote databases slows the search process to an unacceptable level. Therefore, an RDF store must have a memory efficient data representation that leaves enough space for the operation of search algorithms. Currently available RDF data stores are not suitable for association discovery due to their unacceptable performance and high memory requirements. During our own work on the semantic discovery project [24], we have created the following list of necessary features needed for the fast semantic association discovery in large graphs. These include the ability to:

- search for associations of variable length and unspecified directionality,
- work on large RDF graphs in main memory with leaving a sufficient amount of memory for the operation of the search algorithms,
- limit traversal paths to instance resources only (or only to schema level resources),
- produce the results within a reasonable time (on the order of a few minutes), and
- allow a fast start-up of the system by utilizing a pre-loaded RDF storage image.

The above set of requirements was the main motivating factor in creating our own RDF storage system, BRAHMS.

BRAHMS has been already used successfully in the Insider Threat project [2], which proved its value as an RDF storage system offering the necessary foundation for the implementation of fast semantic associations discovery algorithms. In this paper, we describe the design of BRAHMS and present its performance results in comparison to a few of the other available RDF store implementations.

3 RDF Storage Systems and Query Languages

3.1 RDF Query Languages in Association Discovery

Presently available RDF/OWL query languages do not directly support association discovery. Languages such as RQL, RDQL, and SquishQL offer support for path expressions but even though it is possible to specify a template search pattern of resources and relations connecting them, they are not suitable for semantic association discovery. The main problem is that the created path expressions can match only paths of a fixed length and of specified directionality of participating relations. Let us demonstrate it on the example of finding all paths of length up to two relations between two resources, startURI and endure, using the RDQL query language:

```
SELECT ?startURI, ?property_1, ?endURI
FROM (?startURI ?property_1 ?endURI)

SELECT ?startURI, ?property_1, ?endURI
FROM (?endURI ?property_1 ?start)

SELECT ?startURI, ?property_1, ?x, ?property_2, ?endURI
FROM (?startURI ?property_1 ?x)(?x ?property_2 ?endURI)
WHERE ?startURI ne ?x && ?endURI ne ?x

SELECT ?startURI, ?property_1, ?x, ?property_2, ?endURI
FROM (?startURI ?property_1 ?x)(?endURI ?property_2 ?x)
WHERE ?startURI ne ?x && ?endURI ne ?x

SELECT ?startURI, ?property_1, ?x, ?property_2, ?endURI
FROM (?x ?property_1 ?startURI)(?x ?property_2 ?endURI)
WHERE ?startURI ne ?x && ?endURI ne ?x

SELECT ?startURI, ?property_1, ?x, ?property_2, ?endURI
FROM (?x ?property_1 ?startURI)(?endURI ?property_2 ?x)
WHERE ?startURI ne ?x && ?endURI ne ?x
```

The above queries represent the following patterns to be matched:

```
[startURI  --property_1->  endURI]
[startURI  <-property_1--  endURI]
[startURI  --property_1->  x  --property_2->  endURI]
[startURI  --property_1->  x  <-property_2--  endURI]
[startURI  <-property_1--  x  --property_2->  endURI]
[startURI  <-property_1--  x  <-property_2--  endURI]
```

As shown above, six different queries are required to find all paths of length at most two that connect two selected resources. As the path length increases, the number of the required queries grows exponentially, due to non-directionality of relationships. Additionally, because we search for simple paths, each query must have conditions, which guarantee that each resource appears only once in a given path. As a result, even though it is possible to discover semantic associations using the current RDF query languages, it is prohibitively expensive.

3.2 Review of Existing RDF Storage Systems

There are already many existing and widely used RDF storage systems. In this paper, we evaluate Jena, Sesame, and Redland from the point of view of their suitability to the semantic association discovery. The three systems are arguably the most popular ones today. In addition, each one of them has an API for direct RDF querying on the storage or model. As discussed previously, such an API is necessary for implementing association discovery algorithms, because higher-level languages are unsuitable for expressing path queries of unknown length and directionality.

Jena (version 2.1) [12] is an RDF/OWL storage and querying engine (RDQL) implemented in Java. It can store graphs in main memory and in a database. The storage is organized in a triple-centric way. To get the neighborhood of a node, we have to use a general method for finding all triples that satisfy a given pattern. For neighborhood triples, the node plays the role of either a subject or an object. Due to

the available indexing of triples, such searches are performed fast. Unfortunately, the in-memory implementation of the data graph requires large amounts of memory.

Redland (version 1.0.0) [21] is an RDF storage and querying engine implemented in C. It can store graphs in main memory, databases, and files. This RDF storage system is also triple-centered, but in the available 'hashes' memory-model, suitable indexes can be constructed to enable a fast computation of the node neighborhood. Surprisingly, the neighborhood search is slower than in the Java implementations of the two other storage systems and the memory consumption for the 'memory-hashes' is also very high. For our tests, we had to patch Redland to optimize its speed with two indexes in order to get a fast lookup of nodes pointing to and pointed by a specific resource. The patch was done according to the suggestions from the author of Redland [6]. In this way, we have avoided full table scans for each node neighborhood search, which was present in the original version of Redland.

Sesame (version 1.1) [25] is an RDF/OWL storage and querying engine (SeRQL) implemented in Java. It can store graphs in main memory, databases and in files. Only this RDF store has an available node-centric organization, where the neighborhood a given node can be directly extracted. The clear architecture of this system makes it easy to understand and use. Unfortunately, the in-memory implementation of the graph/model requires a large amount of memory, which in turn does not leave much space available for the search algorithms to operate on larger knowledge bases.

4 BRAHMS

BRAHMS has been designed to be a fast main memory-based storage system for RDF, capable of storing large description bases and serving as a base for efficient implementation of semantic association discovery algorithms. The first consequence of our design decisions was to make the description base read-only. BRAHMS has not been designed for modifications of RDF bases, but only for querying them. An updated BRAHMS storage image must be recreated from an updated RDF/RDFS description base. Such an approach allows us to optimize the memory usage, and to use specialized data structures and also to create all of the indexes only once.

The memory usage restriction required us to use a compact representation of the triples, nodes, their values, and storing only the most necessary structural data. On the other hand, the speed requirement demanded creating indexes for fast access and search. Taking into consideration the semantic association discovery algorithms that would be implemented using BRAHMS and the memory size limitations, we have created only the hash tables for matching string URIs with resource nodes as well as basic node-centric indexes that can be used to implement more complex queries and algorithms. The triples of instance resources are indexed as follows:

- subject → object, predicate
- object → subject, predicate
- predicate → subject, object.

These indexes are needed for a fast retrieval of node neighborhoods, as well as searching for and merging of neighborhoods during the semantic association discovery process.

Another design decision was to separate the instance resources, properties, literals and classes as they represent different pieces of information. Literals, properties and schema type resources are kept in separate memory structures with their own, similar indexes.

The final design decision addresses the optimization of the startup time. Some of the most time consuming operations while working on RDF data stored in main memory are the loading and parsing of the RDF file, together with the creation of suitable indexes. BRAHMS uses the Raptor RDF parser [19] for the initial load of the RDF file into the internal memory structures. The file load and the construction of indexes can be done only once and the created structures can be written to disk as a memory image of the internal representation for the future use. This requires that the internal memory structures do not use direct memory addresses, as those cannot be preserved in an image file. As an added bonus, this allows us to easily coalesce the memory image fragments into one compact memory block.

BRAHMS has been implemented in C++. All data is stored in a logically contiguous memory block and all internal references are made relative to the origin of the memory space. Each resource, class, property and literal is identified by a unique numeric identifier in its group. To minimize the memory usage, the internal data stores operate on these identifiers, keeping their string values in separate tables. BRAHMS uses the following types of internal data stores:

- the list of triples that contain only numerical identification of resources, properties, classes or literals together with indexes for fast access to them,
- the list of resources, properties, classes and literals that match ID with proper label/URI, and
- the list of resource values (URIs) and literal values.

5 Experiment Design

In our tests, we have compared our own system, BRAHMS, with the three RDF storage systems discussed previously: Jena 2.1 (Java), Sesame 1.1 (Java) and Redland 1.0.0 (C).

5.1 Tested Functions and Algorithms

We have selected the depth-first search and the bi-directional breadth-first exhaustive search algorithms for our tests, since many of the semantic association search methods are based on either of the two basic algorithms. As a result, the performance of these two algorithms offers a good insight into how a variety of other related semantic search algorithms would perform when implemented on the tested RDF storage systems.

We have performed the following three tests on each of the compared RDF stores:

- loading a dataset into memory, in order to estimate the memory consumption and the required load time,
- executing a basic depth-first search (DFS) algorithm in order to find semantic associations up to a given (fixed) length; DFS requires very little memory, as it is restricted by the maximum length of the association path,
- executing a bi-directional breadth-first search (bi-BFS) utilizing a trie representation of the search structures in order to find semantic associations up to given (fixed) length; this algorithm uses an exponential amount of memory as a function of the path length.

All of the above tests have been performed on the main memory storage implementations. The bi-directional breadth-first search is an algorithm that searches for the association paths by growing path frontiers from both endpoints of the search. A join of the two frontiers is performed at each step to find the complete paths.

5.2 Data Sets

In all of the tests, we have used both synthetic and real-life datasets. These included:

- SWETO (Semantic Web Testbed Ontology) [3] which is a dataset that contains real-world information about publications in computer science, including authors and co-authors, conferences, and journals. We have used two sets in our tests:
  - a small set of 14Mb, containing 187,507 statements, 55,876 unique resources with the average node degree of 2.16 in the biggest connected component; this dataset was used in our semantic association ranking experiments [11].
  - a big set of 255Mb, containing 3,196,692 statements, 813,479 unique resources and the average node degree of 3.90 in the biggest connected component.
- a small synthetic dataset, generated to include three ontologies (business, sports, and entertainment); 14Mb in size, containing 104,891 statements, 29,825 unique resources and the average node degree of 3.86 in the biggest component, and
- a big synthetic set, generated as Univ(50, 0) using the Lehigh University Benchmark [10], 556Mb in size, containing 6,888,642 statements, 1,082,818 unique resources and the average node degree of 6.09.

We did not use TAP [9] for testing purposes, because the number of entities in it is relatively low in comparison to SWETO and, what is more important, very few of the entities are linked by longer semantic association paths. Although this dataset represents an important knowledge base of facts and entities, its low connectivity makes it unsuitable for testing of discovering longer semantic associations.

5.3 Endpoint Resources

We have used the following resources as the endpoints in the depth-first search and bi-directional breadth-first search algorithm tests:

Table 1. Datasets and endpoint resources

Data set	Start resource	End resource
Small SWETO	http://lsdis.cs.uga.edu/proj/ semdis/testbed/#SWEET_215003 Chee-Keng Yap	http://lsdis.cs.uga.edu/proj/ semdis/testbed/ #SWEET_949653 Ravi Ramamoorthi
Big SWETO	http://lsdis.cs.uga.edu/proj/ semdis/testbed/#SWEET_215003 Chee-Keng Yap	http://lsdis.cs.uga.edu/proj/ semdis/testbed/ #SWEET_949653 Ravi Ramamoorthi
Small synthetic	http://lsdis.cs.uga.edu/semdis/ sports/Athlete_7271	http://lsdis.cs.uga.edu/semdis/ business/Spokesperson_7611
Big synthetic Univ(50, 0)	http://www.Department0. University0.edu/FullProfessor0	http://www.Department0. University49.edu/ FullProfessor0

When using the small SWETO knowledge base, we chose the same endpoints that were used for the on-line demo of association ranking. We have used the same endpoints when using the big version of SWETO in order to demonstrate the difference in the number of possible paths between the same resources as the size of the knowledge base increases.

For tests on the small synthetic graph, the sample endpoints were taken from [18].

The big synthetic graph was created as Univ(50,0) using the Lehigh University Benchmark, exactly as presented by the authors in their paper. Two professors from two distant universities were chosen as the endpoints. Our choice was random, but in most of the resources we have tested, we were able to find connections to all other resources in the graph using paths of length six or seven.

5.4 Test System Environment

All of the tests were performed on a dual-processor computer system with the following configuration:

- 2 Intel(R) Xeon(TM) CPUs running at 3.06GHz; 4Gb memory (3Gb available for a single user process); 220GB of hard disk available
- Red Hat 9.0 Enterprise Linux operating system,
- Java SDK 1.4.1_02; 1800Mb of maximum heap size for loading the bigger data sets and 512Mb of maximum heap size for loading the smaller data sets,
- gcc (GCC) 3.2.2 20030222 (Red Hat Linux 3.2.2-5), C/C++ code compiled with the '–O6' optimization flag.

6 Experiment Results

In the performed tests, we have concentrated both on the speed and on the memory requirements. The memory requirements of the tested systems varied greatly, and the

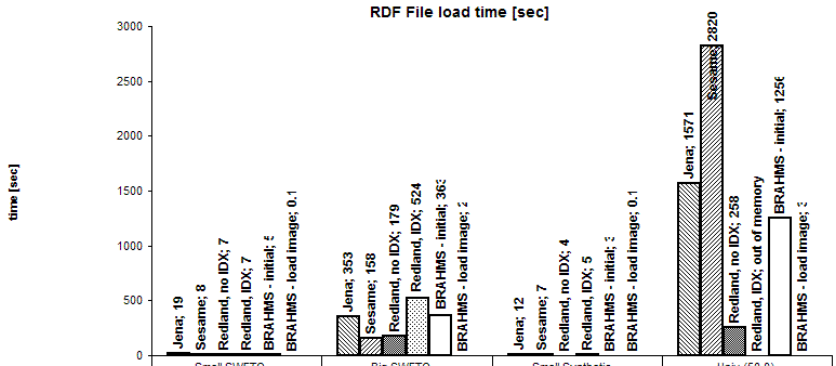
amount of the free memory available for the search algorithms after the data set has been loaded strongly influenced the choice of the search algorithm. The memory efficient but slow DFS, could be used with all storage implementations, as its memory requirements were very small. The bi-directional BFS was much faster, but it had high (exponential) memory requirements.

In the first experiment, we measured the time needed for each system to parse the RDF file and load its contents into memory. This value represents the time needed for a cold start. In addition, we measured the amount of memory needed to load each of the datasets. This showed how compact the memory representation was and which search algorithm was applicable for each of the test data sets.

6.1 RDF Data Load Tests

In this experiment, we measured the time needed to load an RDF file to memory and initialize the system. The time was measured using the time system call under Unix.

The results are shown in Fig. 2.



	Small SWETO	Big SWETO	Small Synthetic	Univ (50,0)
Jena	19	353	12	1571
Sesame	8	158	7	2820
Redland, no IDX	7	179	4	258
Redland, IDX	7	524	5	0
BRAHMS - initial	5	363	3	1256
BRAHMS - load image	0.1	2	0.1	3

Fig. 1. RDF file initial load time

Two different loads were performed in Redland and in BRAHMS:

- "Redland no IDX" represents the unmodified Redland, without the additional indexes.
- "Redland IDX" stands for the patched Redland with two additional indexes: subject to predicate, object and object to predicate, subject.

– "BRAHMS initial" is the time needed by BRAHMS to parse and load the RDF file, create the indexes, and save the memory image file containing the internal data structures to disk.
– "BRAHMS load image" stands for BRAHMS utilizing a previously created memory image file.

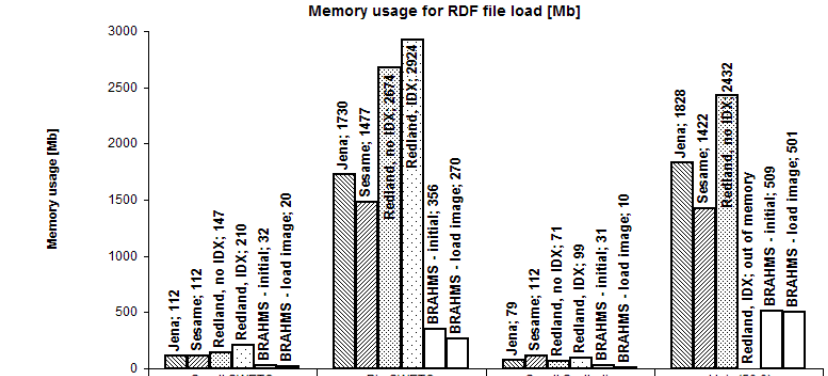
Only one load operation was performed in Jena and Sesame.

As expected, for the smaller datasets the differences are not that significant, and the data load operation taking even twenty seconds is still acceptable. However, significant differences are evident for the bigger datasets, where the load times are on the order of magnitude longer.

This is the reason why we have decided for BRAHMS to be able to create and load a memory image of the internal data structures. Parsing the RDF file and creating the indexes is performed only once, and all of the subsequent experiments require only a very fast load of the previously prepared memory image file.

6.2 Memory Usage Tests

Along with measuring the time needed to load the datasets into memory, we also measured the memory usage of each tested system. The results are shown in Fig. 3.



	Small SWETO	Big SWETO	Small Synthetic	Univ (50,0)
Jena	112	1730	79	1828
Sesame	112	1477	112	1422
Redland, no IDX	147	2674	71	2432
Redland, IDX	210	2924	99	0
BRAHMS - initial	32	356	31	509
BRAHMS - load image	20	270	10	501

Fig. 2. Memory usage for loading an RDF file

The memory requirements for the smaller datasets are insignificant for all the tested RDF stores, as they occupy only a small fraction of the available memory. For the big datasets, this becomes an important issue. The loaded datasets can occupy hundreds of megabytes or even gigabytes of memory. Some systems have almost reached the

hard memory limits in the test computer system. Such high memory usage does not leave much runtime space for any faster algorithms relying on large workspaces.

The restrictions placed on the size of the used data structures allow BRAHMS to use much less memory than required by the other tested systems. This allows BRAHMS to load larger datasets for experiments and still have sufficient memory available for running search algorithms.

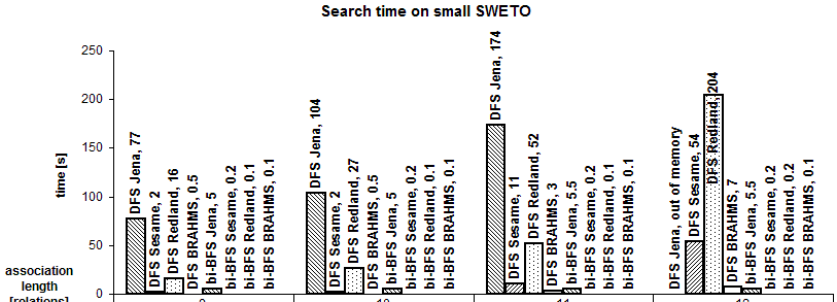
6.3 Semantic Association Search Tests

In this section, we present the timing results from running DFS and bi-BFS algorithms on different datasets using Jena, Sesame, Redland and BRAHMS.

For our tests, we have used Redland patched with the additional indexes in order to reach the speeds comparable to the other tested systems. BRAHMS tests used the memory image file that was created prior to the timing experiments.

Search on Small SWETO Dataset. Using the small SWETO, we tested both the classic DFS and the bi-BFS algorithms. Because it is a relatively small dataset, it is still feasible to perform the DFS and obtain the results within an acceptable time.

The results include the time used only for running the algorithms. The time needed to load the data file to memory (or the memory image in case of BRAHMS) has been excluded.



association length [relations]	9	10	11	12
DFS Jena	77	104	174	0
DFS Sesame	2	2	11	54
DFS Redland	16	27	52	204
DFS BRAHMS	0.5	0.5	3	7
bi-BFS Jena	5	5	5.5	5.5
bi-BFS Sesame	0.2	0.2	0.2	0.2
bi-BFS Redland	0.1	0.1	0.1	0.2
bi-BFS BRAHMS	0.1	0.1	0.1	0.1
Found paths	47	61	61	289

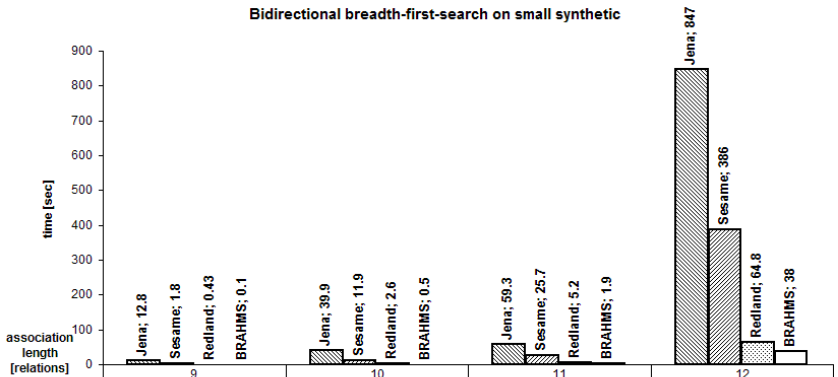
Fig. 3. Timing results from DFS and bi-BFS on small SWETO

As shown in Fig. 4, DFS is significantly slower than the bi-BFS. The differences were up to a few orders or magnitude, but this came with high memory requirements for bi-

BFS. On the smaller or less connected datasets, the differences may be not as significant (seconds or a few minutes), which may be acceptable for the analyst. On the bigger or highly connected datasets, the DFS and related algorithms may be unacceptable.

Search on Small Synthetic Dataset. This synthetically generated dataset has a normal distribution of node degrees, which differs from the real-life SWETO. Although the average node degree is similar and the size of the dataset resembles the size of the small SWETO dataset, the number of located paths is much higher.

In this and further experiments, we have used only the bi-BFS, as it is a much faster algorithm. The timing results produced by DFS would be unacceptable and take in excess of several hours.



association length [relations]	9	10	11	12
Jena	12.8	39.9	59.3	847
Sesame	1.8	11.9	25.7	386
Redland	0.43	2.6	5.2	64.8
BRAHMS	0.1	0.5	1.9	38
Found paths	8559	131009	1680943	24392420

Fig. 4. Timing results from bidirectional BFS on small synthetic dataset

Opposite to the small SWETO, the number of discovered paths grows into millions as the path length exceeds 10. Still, all of the tested RDF stores can compute the paths in a reasonably short time without facing the memory limitation problems. This situation changes drastically for the bigger datasets.

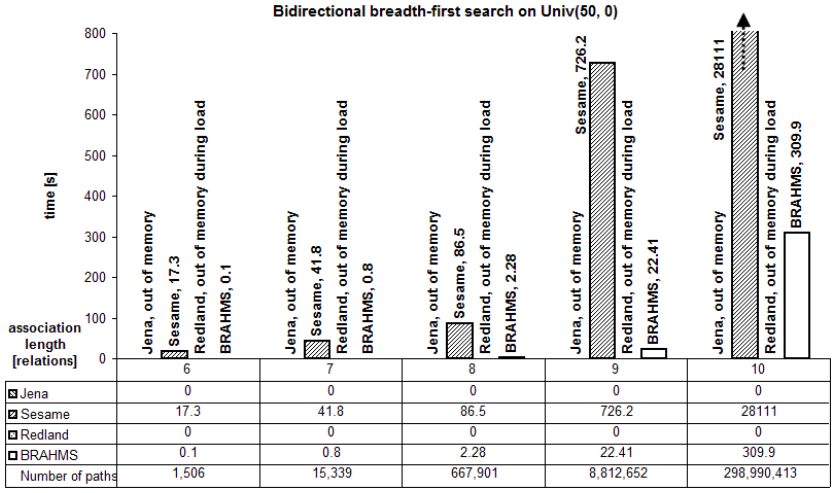
Search on Big Datasets. The path search algorithms tests using the big datasets were limited to paths of length up to 10. Even with this maximum length, the number of all located simple paths between the two selected entities has grown above tens of millions. The search for longer paths caused the expansion of the search space beyond the available memory, in most cases.

Using the big SWETO dataset, we were able to perform the search only using BRAHMS. The other systems have run out of memory during the search, and were

even not able to discover paths of length six or seven. The results are presented in Table 2.

Table 2. bi-BFS results using BRAHMS on the big SWETO dataset

Association length [relations]	6	7	8	9	10
BRAHMS execution [sec]	0.1	0.1	0.8	1.1	66.4
Number of found paths	202	202	214,778	214,778	46,641,867



association length [relations]	6	7	8	9	10
Jena	0	0	0	0	0
Sesame	17.3	41.8	86.5	726.2	28111
Redland	0	0	0	0	0
BRAHMS	0.1	0.8	2.28	22.41	309.9
Number of paths	1,506	15,339	667,901	8,812,652	298,990,413

Fig. 5. Timing results from bi-BFS on Univ(50, 0) dataset

The synthetic dataset Univ(50, 0) required a smaller memory for the search than the big SWETO for association paths up to length 10. Sesame was able to successfully run the bi-BFS search and produce results, but the computation time was much longer than that of BRAHMS.

The modified Redland could not load this dataset into memory and consequently no associations could be discovered in the Univ(50, 0) dataset. These could be still computed using the database storage model, but that resulted in much higher run times (not reported here).

Jena was able to load this dataset to memory, but due to the high memory usage, no associations could be discovered using bi-BFS. The system did not allow allocating enough of the additional memory.

7 Conclusions and Future Work

In this paper we presented BRAHMS, the storage system for RDF, and its applicability for fast discovery of semantic associations in relatively large description

bases. We compared its capabilities with three other, publicly available RDF/OWL storage systems, both in terms of speed and memory requirements. Even for smaller data sets, BRAHMS was able to compute semantic associations faster than the other RDF stores, using the same search algorithms. Longer semantic associations in bigger graphs could not be computed in a reasonable amount of time by the other systems due to their high memory requirements, while BRAHMS has been able to produce results and in within an acceptable time. This shows that semantic association discovery can become a reality, even on relatively large RDF data sets.

The discovery of semantic associations is still difficult for the presently available RDF query languages. Some of them support path expressions, but they are limited to paths of known length and defined relationship directionality.

In the near future, we plan to experiment with a variety of semantic association discovery algorithms, utilizing a language for defining regular paths, similar to [1]. The regular expressions defined over the RDF resources and types (including subsumption and class hierarchy) will enable us to define the association paths of interesting patterns and significantly restrict the search space of the semantic association discovery. Further development of semantic association search algorithms and their improvements may lead to a new perspective for knowledge discovery and searching in the Semantic Web.

Acknowledgements

We would like to thank Matt Perry for his synthetic graph generator and many other members of the LSDIS lab for their feedback and valuable comments on the design and usage of BRAHMS.

References

1. Abiteboul, S. and Vianu, V., Regular Path Queries with Constraints. in *16th ACM Symposium on Principles of Database Systems*, (Tuscon, Arizona, USA, 1997).
2. Aleman-Meza, B., Burns, P., Eavenson, M., Palaniswami, D. and Sheth, A., An Ontological Approach to the Document Access Problem of Insider Threat. in *IEEE International Conference on Intelligence and Security Informatics (ISI-2005)*, (Atlanta, Georgia, USA, 2005).
3. Aleman-Meza, B., Halaschek, C., Sheth, A., Arpinar, I.B. and Sannapareddy, G., SWETO: Large-Scale Semantic Web Test-bed. in *16th International Conference on Software Engineering and Knowledge Engineering (SEKE2004): Workshop on Ontology in Action*, (Banff, Canada, 2004).
4. Angles, R. and Gutierrez, C., Querying RDF Data from a Graph Database Perspective. in *2nd. European Semantic Web Conference (ESWC2005)*, (Heraklion, Greece, 2005).
5. Anyanwu, K. and Sheth, A., r-Queries: Enabling Querying for Semantic Associations on the Semantic Web. in *The Twelfth International World Wide Web Conference*, (Budapest, Hungary, 2003).
6. Beckett, D. Creating additional storage hashes, 2003, redland-dev - Redland development mailing list.

7. Beckett, D., The Design and Implementation of the Redland RDF Application Framework. in *Tenth International World Wide Web Conference*, (Hong Kong, 2001), ACM.
8. Broekstra, J., Kampman, A. and Harmelen, F.v., Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. in *International Semantic Web Conference 2002*, (Sardinia, Italy, 2002).
9. Guha, R.V. and McCool, R. The tap knowledge base.
10. Guo, Y., Pan, Z. and Heflin, J., An Evaluation of Knowledge Base Systems for Large OWL Datasets. in *Third International Semantic Web Conference*, (Hiroshima, Japan, 2004), Spinger, 274-288.
11. Halaschek, C., Aleman-Meza, B., Arpinar, I.B. and Sheth, A.P., Discovering and Ranking Semantic Associations over a Large RDF Metabase. in *30th International Conference on Very Large Data Bases*, (Toronto, Canada, 2004).
12. Jena. http://www.hpl.hp.com/semweb/jena.htm.
13. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D. and Scholl, M., RQL: A Declarative Query Language for RDF. in *The Eleventh International World Wide Web Conference*, (Honolulu, Hawaii, USA, 2002), ACM.
14. Krebs, V. Mapping Networks of Terrorist Cells. *Connections*, *24* (3). 43-52.
15. McBride, B., Jena: Implementing the RDF Model and Syntax Specification. in *Tenth International World Wide Web Conference: Semantic Web Workshop*, (Hong Kong, 2001).
16. Mendelzon, A.O. and Wood, P.T., Finding Regular Simple Paths In Graph Databases. in *15th Conference on Very Large Databases*, (Amsterdam, The Netherlands, 1989), Morgan Kaufman pubs. (Los Altos CA).
17. Miller, L., Seaborne, A. and Reggiori, A., Three Implementations of SquishQL, a Simple RDF Query Language. in *First International Semantic Web Conference on The Semantic Web*, (Sardinia, Italy, 2002), Springer-Verlag, 423 - 435.
18. Milnor, W.H., Ramakrishnan, C., Perry, M., Sheth, A.P., Miller, J.A. and Kochut, K.J., Discovering Informative Subgraphs in RDF Graphs - Preliminary Results (submitted to). in 4th International Semantic Web Conference (ISWC 2005), (Galway, Ireland, 2005).
19. Raptor. http://librdf.org/raptor/.
20. RDF. http://www.w3.org/RDF/.
21. Redland. http://librdf.org/.
22. Seaborne, A. RDQL - A Query Language for RDF, 2004.
23. Semagix. Anti-Money Laundering - CIRAS. http://www.semagix.com/solutions_ciras.html.
24. Semantic Discovery: Discovering Complex Relationships in Semantic Web. http://lsdis.cs.uga.edu/Projects/SemDis/.
25. Sesame. http://www.openrdf.org/.
26. Sheth, A., From Semantic Search & Integration to Analytics. in Dagstuhl Seminar Proceedings 04391, (Dagstuhl, Germany, 2005).
27. Sheth, A., Aleman-Meza, B., Arpinar, I.B., Halaschek, C., Ramakrishnan, C., Bertram, C., Warke, Y., Avant, D., Arpinar, F.S., Anyanwu, K. and Kochut, K. Semantic Association Identification and Knowledge Discovery for National Security Applications. Journal of Database Management.
28. Sheth, A., Arpinar, B. and Kashyap, V. Relationships at the Heart of Semantic Web: Modeling, Discovering and Exploiting Complex Semantic Relationships. in Nikravesh, M., Azvin, B., Yager, R. and Zadeh, L.A. eds. *Enhancing the Power of the Internet Studies in Fuzziness and Soft Computing*, Springer-Verlag, 2003.
29. SPARQL. Query Language for RDF. Prud'hommeaux, E. and Seaborne, A. eds., 2005.