Towards Scalable Information Integration with Instance Coreferences

Abir Qasem
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015, USA
abir.qasem@gmail.com

Dimitre A. Dimitrov
Tech-X Corporation
5621 Arapahoe Avenue, Suite A
Boulder, CO 80303, USA
dad@txcorp.com

Jeff Heflin
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015, USA
heflin@cse.lehigh.edu

Abstract

Identifying data instances, that come from different sources but denote the same entity is necessary for effective integration of Semantic Web data. This "instance coreference" identification problem has gained attention in recent years. Although this is a critical aspect of the overall information integration problem in the Semantic Web, we put forward that information integration algorithms also need to be extended in order to work effectively and efficiently in the presence of these coreferenced entities (whether they are discovered by a tool or explicitly stated with an `owl:sameAs` assertion). We describe such an extension to our Goal Node Search algorithm for Semantic Web information integration.

1 Introduction

Record duplication and data linkage have been widely studied in the database area [Elmagarmid *et al.*, 2007]. In the Semantic Web community similar issues are being examined under the "instance coreference" research area [Nikolov *et al.*, 2008]. Solving the instance coreference problem is critical to achieving web-wide information interoperability and integration. So far the Semantic Web community's primary focus in this area has been to identify coreferent entities using ontological information and various machine learning techniques. However, identifying these corefernces is only part of the solution.

Whether they are identified using a tool or explicitly asserted using `owl:sameAs`, handling distributed coreferenced entities poses some interesting challenges. Suppose we want to get a list of academic papers in computer science written by authors who have been advised by Marvin Minsky. We can get information about Minsky's advisees from the the AI Genealogy Project (AIGP)[1] and a list of authors of academic papers in computer science from the computer science bibliography web site DBLP[2]. These two sites use different URI schemes for the records and therefore the same entities (the advisees/authors in this case) will generally be syntactically dissimilar. For example, Eugene Charniak is referred to by `http://aigp.eecs.umich.edu/researcher/show/489` in AIGP and by `http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/c/Charniak:Eugene.html` in DBLP.

Let us assume that each of these websites have a Semantic Web version where they expose their data in OWL[3] format (i.e. they are OWL knowledge bases). There are 4532 researchers listed in the AIGP website and most of them have more than one article in DBLP. Therefore, the number of coreferent entities in these two knowledge bases will be at least around 4000. However, Marvin Minsky has only 20 advisees. Furthermore, coreferent instances for organization, places etc. that are available in the knowledge bases are not relevant to the query. Therefore, most of the coreference information is superfluous in answering the query about papers written by Minsky's advisees. When dealing with large knowledge bases this issue becomes more pronounced. For example, DBPedia[4] and Geonames[5] have about about 80,000 coreferent entities and Geonames and the CIA world factbook[6] has about 100,000 coreferent entitities. Given that coreference is transitive, one can clearly see how this information can grow very fast. Since reasoning is an expensive process it is prudent that we keep the size of our knowledge base to a minimum and develop an approach that can reduce this "wastage" by using only the coreference information relevant to a query.

In this paper we propose an approach that addresses several of the issues described above. Specifically we make the following two technical contributions.

1. We put forward that information integration algorithms need to be extended in order to work effectively and efficiently in the presence of coreferent entities.
2. We describe an extension to our Goal Node Search (GNS) algorithm [Qasem *et al.*, 2008b] that handles coreferent entities in an efficient manner.

The rest of the paper is organized as follows: In Section 2, we describe Ontology Based Information Integrator (OBII),

[1]http://aigp.eecs.umich.edu/
[2]http://www.informatik.uni-trier.de/ ley/db/

[3]http://www.w3.org/TR/owl-ref/
[4]http://dbpedia.org/
[5]http://www.geonames.org/
[6]https://www.cia.gov/library/publications/the-world-factbook/

a distributed query answering system in which we have implemented our extension of the GNS algorithm. In Section 3, we describe our initial solution in details. In Section 4, we discuss some related work with our approach and in Section 5 we conclude and discuss future work.

2 OBII:A Semantic Web query answering system

OBII is a Semantic Web query answering system that uses a "source selection" framework [Qasem *et al.*, 2008a] and answers distributed extensional queries posed in SPARQL[7] syntax.

The source selection framework is an approach for identifying the minimal set of potentially relevant Semantic Web data sources for a given query. These selected sources are then loaded into a Description Logic (DL) reasoner, and processed to obtain the answer(s) to the query. The framework assumes that the Semantic Web is composed of a large number of relatively small data sources (similar to web pages). These sources are files that must be loaded in their entirety or not at all. The relevance of a data source to a query is expressed by a meta data referred to as the "REL" statement. A data source provider can use REL statements to summarize the contents of a data source in terms of classes whose instances the data source has information about and the properties used to relate them.

In the source selection framework "map" ontologies are used to align heterogeneous ontologies. The map ontologies are like any other OWL ontology except they consist solely of axioms that relate concepts from one ontology to concepts of another ontology. The term domain ontologies is used to refer to all other ontologies.

OBII has a plug-in architecture and can use any algorithm that is compatible with the source selection framework. We have extended one such algorithm (the GNS algorithm) to handle instance coreferences in an effective and efficient manner. We will discuss the GNS algorithm in Section 3. This will be done in the context of demonstrating why an extension to the algorithm is needed and explaining our design choices in developing the extension. For an extended exposition of the GNS algorithm readers are referred to Qasem *et al.* [2008b] .

GNS is designed to work with a subset of OWL DL (a decidable and commonly used fragment of OWL) called OWL for Information Integration (OWLII) [Qasem *et al.*, 2008a]. This sublanguage is compatible with Global-As-View (GAV) [Garcia-Molina *et al.*, 1997] and Local-As-View (LAV) [Levy *et al.*, 1996] rules. Let $B\left(\overline{X}\right)$ and $H\left(\overline{X}\right)$ (sometimes subscripted) be unary or binary atoms with a vector $\overline{X}$ of arguments. A GAV rule has the form $H(\overline{X})$:- $B_1(\overline{X}_1) \wedge B_2(\overline{X}_2) \wedge \ldots \wedge B_n(\overline{X}_n)$, where the left hand side of the :- is called the head and the right hand side is called the body. A LAV rule has the form $H(\overline{X}) \subseteq B_1\left(\overline{X}_1\right), B_2\left(\overline{X}_2\right), \ldots, B_m(\overline{X}_m)$ where the left hand side of the $\subseteq$ is called the head and the right hand side is called the body. From a knowledge representation point of view, a GAV rule is essentially equivalent to a Horn clause without function symbols and a LAV rule is a First Order Logic (FOL) implication with a single antecedent and multiple consequents.

Figure 1 shows the architecture of OBII with arrows indicating the flow of information when processing a query. OWLIIRuleProcessor translates the OWL ontologies and REL meta statements into LAV/GAV rules. The SourceSelector implements the GNS algorithm and AnweringEngine loads the selected sources into KAON2[8](a DL reasoner) and obtains the answer to a query after the reasoning.

The REL statements can be expressed as LAV rules (with some minor modifications for denoting the URI of a source). OWLIIRuleProcessor translates the domain and map ontologies into a set of LAV/GAV rules and stores them in a MapView object. OWLIIRuleProcessor translates the set of REL statements into LAV rule and source URL pairs and store them in a SourceView object. A collection of MapView objects and SourceView objects is maintained by MapKB.

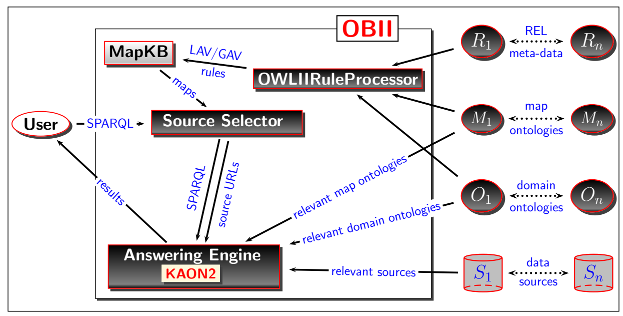

Figure 1: OBII architecture diagram with arrows showing the flow of information when processing a query.

3 GNS Extension: an Initial Solution

In this section we first describe the GNS algorithm and show why it is inadequate in handling instance coreferences. We then describe our proposed extension that handles instance coreferences in an effective and efficient way.

3.1 The GNS

Given a conjunctive query and a set of LAV/GAV rules, the GNS algorithm identifies all possible additional subgoals that can be found by applying the LAV/GAV rules to each query subgoal or its expansions. Identifying these subgoals can be viewed as a search problem where each node of the search tree is either an original or an expanded subgoal; and the search task is to identify all possible paths that can be derived from applying the LAV/GAV rules to the nodes. As the search space for the algorithm is the set of all possible expanded goal nodes, it is referred to as the Goal Node Search algorithm.

In GNS, the search is implemented by maintaining two lists: an open list of nodes to be expanded and a closed list of nodes that have been expanded. The algorithm continues to

[7] http://www.w3.org/TR/rdf-sparql-query/

[8] http://kaon2.semanticweb.org/

expand the open list until it is empty while adding the node that has been expanded to the closed list and adding the expanded new nodes to the open list. The open list is initialized with goal nodes created from the subgoals of the query to give us the starting point of the search. The GNS is shown in Algorithm 1.

Algorithm 1 Goal Node Search.

```
GNS(Query q, MapViews mv, SourceViews sv)
 1: ol = ∅
 2: cl = ∅
 3: selectedSources = ∅
 4: expandedNodes = ∅
 5: ol ← INIT-FROM-QUERY(q)
 6: while ol ≠ ∅ do
 7:   n ← ol.pop()
 8:   if not cl.contains(n) then
 9:     expandedNodes ← ∅
10:     omaps ← {m | (n.ont, m) ∈ mv}
11:     for each v ∈ omaps do
12:       if GAV(v) and UNIFY(n, HEAD(v)) then
13:         expandedNodes ∪ GAV-EXPAND(n, v)
14:       else if LAV(v) and UNIFY(n, b) for some b ∈ BODY(v) then
15:         expandedNodes ∪ LAV-EXPAND(n, v)
16:     ol ← ol ∪ expandedNodes
17:     cl.add(n)
18: for each n ∈ cl do
19:   for each v ∈ sv do
20:     if LAV(v) and UNIFY(n, b) for some b ∈ BODY(v) then
21:       selectedSources ← selectedSources ∪ LAV-EXPAND(n, v)
22: return selectedSources
```

The routines HEAD(v) and BODY(v) return the head or the body of given rule. If the head of a GAV rule unifies with the goal node, the expansion includes a set of nodes corresponding to the body of the GAV rule. If any atom from the body of a LAV rule unifies with the node, then the expansion includes the head of the LAV rule. In both cases, variables from the goal node are substituted into the generated nodes.

In order to guarantee termination, i.e. to avoid cyclic expansion, we check if a node is already in the closed list before expanding it. Furthermore, for efficiency of storage, in addition to not expanding nodes that are already in the closed list, the GNS prunes nodes that are also superseded by a node in the closed list. A node *n* supersedes another node *m* if the result from a query using *n*'s predicate is necessarily a superset of the result from a query using *m*'s predicate. Syntactically, *n* supersedes *m* if there is a unification involving only substitutions to variables from *n*. For example, p(x, y) supersedes p(x,CONST) but p(x,x) does not supersede p(CONST1, CONST2) where CONST1 is not equal to CONST2 etc. Using a supersede relationship between nodes as opposed to a strict match to decide if a node has been expanded allows the algorithm to keep only the most general node in the list. This reduces the size of the list that is maintained. This "special contains" is implemented in the cl.contains routine (line 8). In determining the supersedes relationship, cl.contains essentially performs a one sided unification test.

Query	adviseeOf(x, Minsky) ∧ author(x,p) ∧ livesIn(x, Washington-DC)
GAV	livesIn(x, DC) :- hasHome(x,h) ∧ locatedIn(h, DC)
REL 1	locatedIn(h, District-of-Columbia) From a real state site
REL 2	hasHome(x, h) From a "white page" service

Table 1: Example: Papers By Minsky's Advisees Living in DC

The algorithm has found all possible expansions when the open list is empty. The REL statements (stored in MapKB as SourceView objects) are then used to complete the source selection process. If a node in the closed list unifies with the body of a Source View, the source URL is extracted and added to the list of selected data sources that will be loaded in the reasoner. All the relevant sources are loaded in their entirety into a reasoner and then the original query is issued and the results obtained from the reasoner.

3.2 Handling Instance Coreference

We first motivate the need for special processing of instance coreferences with an example shown in Table 1. Consider a more restricted version of the query about academic papers introduced in Section 1: we want a list of academic papers written by Marvin Minsky's advisees who live in Washington DC. In the example we use lowercase letters to refer to variables and string constants as a shorthand notation for the URIs.

If all coreferent instances used canonical identifiers (i.e. they were not coreferent instances at all) then the livesIn(x, Washington-DC) atom of the query would have been processed by GNS as follows. The atom would unify with the head of the GAV rule and would have been expanded to hasHome and locatedIn atoms. Then those atoms would unify with the body of the two REL statements from the real estate and the white page web site and the two sources would have been loaded into the reasoner. In processing the complete query other atoms would have been expanded similarly by the maps whose heads or bodies unify with those atoms and other relevant sources would have been loaded. Finally, in answering the query all necessary joins from all the selected relevant sources would have been performed by the reasoner (e.g. data about locatedIn(h, District-of-Columbia) and hasHome(x, h) should join in the binding of h etc.) .

However, in the example (just like the Web) there are several coreferent instances. Therefore, if we cannot provide relevant equivalence information to the algorithm it will produce incomplete answers and in some cases will be inefficient. The algorithm needs equivalence information during the unify process of both node expansion (line 12, line 14) and source selection (line 20). For example, livesIn(x, Washington DC) will not unify with livesIn(x, DC) as the second argument of the atoms will not match. Similarly, in selecting relevant sources, locatedIn(h, DC) will fail to match with REL statements atom locatedIn(h, District-of-Columbia).

In addition to completeness, coreferent instances also play a role in the efficiency of the GNS algorithm. Recall from Section 3.1, the closed list uses a special contains function to prune nodes that are superseded by a node that is already in the closed list. Consider, that we have a node author(x, GNS) in the closed list (i.e it has already been expanded) and we are about to expand a node author(x, Goal-Node-Search). We should not expand this node, provided we can determine if author(x, Goal Node Search) is superseded by author(x, GNS). Clearly, if we can provide the equivalence information (GNS is same as Goal-Node-Search) during this determination (line 8 cl.contains function) GNS will not expand author(x, Goal-Node-Search).

We observe that in the presence of coreferent instances, a DL reasoner will fail to identify join conditions that require the knowledge of equivalence information between two syntactically different entities. Therefore, OBII will fail to join data from selected sources if the bindings of the join variable(s) from different sources use different URIs. In our example query, adviseeOf(x, Minsky) ∧ author(x,p) ∧ livesIn(x, Washington-DC), the bindings of x from different sources will most likely be coreferent instances (e.g, in Section 1 we have mentioned how Eugene Charniak is referred to using different URIs in two different data sources) and therefore will not join unless the reasoner is provided with the relevant equivalence information.

It is apparent from the above discussion that in order for the GNS and the DL reasoner to work effectively and efficiently in the presence of coreferent instances, we need to provide equivalence information of all URIs used by the system in answering a given query. We now discuss one possible solution to this problem.

We assume that the equivalence information is available in the Semantic Web as `owl:sameAs` statements that are asserted in various OWL files. We can use a meta data "RELSameAsDoc" to let a system know where to find these `owl:sameAs` statements. A RELSameAsDoc statement of the form RELSameAs(doc, $\{URI_1, URI_2, ..., URI_n\}$) states that doc has equivalence information about the set of URIs $\{URI_1, URI_2, ..., URI_n\}$. Note that there may be many different equivalence classes in the set but in order to keep our meta data compact we have chosen not to express that here.

In our proposed solution we introduce an abstract data type EquivalenceKB. It collects and organizes equivalence information about URIs. EquivalenceKB essentially supports the disjoint set data structure operations [Cormen *et al.*, 2001] on sets of equivalence classes of all known URIs.

EquivalenceKB defines a function GET-ALL-EQUIVALENTS(URI) which returns the equivalence class of the URI (both direct and inferred equivalences). The EquivalenceKB has a variable URI, which stores a list of all URIs whose equivalence information is available in the EquivalenceKB. The EquivalenceKB also defines an UPDATE-EQUAL-KB method (described in Algorithm 2) which updates the EquivalenceKB with newly discovered equivalence information.

The method UPDATE-EQUAL-KB is supported by a private

Algorithm 2 UPDATE-EQUAL-KB

```
UPDATE-EQUAL-KB(EquivalenceKB, list of URI ul)
 1: if ul = ⊘ then
 2:    return
 3: else
 4:    ul ← ul \ EquivalenceKB.URI
 5:    EquivalenceKB.URI ← EquivalenceKB.URI ∪ ul
 6:    for each u ∈ ul do
 7:       listOfDocs ← listOfDocs ∪ DOCS-WITH-SAMEAS(u)
 8:       for each d ∈ listOfDocs do
 9:          sameAsPairs ←EXTRACT-SAMEAS(d)
10:          ADD-TO-KB(sameAsPairs)
11:          newURIs ← newURIs ∪ all individual URIs from sameAsPairs
12:    ul ← newURIs \ EquivalenceKB.URI
13:    UPDATE-EQUAL-KB(EquivalenceKB, ul)
```

method ADD-TO-KB, which adds equivalence information represented as a set of URI pairs to the internal storage of the EquivalanceKB. EquivalanceKB does not specify the exact mechanism of how the equivalence information is stored. That decision is left up to the specific implementation. We describe one such implementation later in this section.

The EquivalanceKB also uses the following auxiliary methods. The method DOCS-WITH-SAMESAS, given a URI, will return the documents that have `owl:sameAs` statements about that URI. This method is used in line 7 in Algorithm 2. We can implement this method by creating an inverted index of RELSameAs statements. The method EXTRACT-SAMEAS used in line 9 will retrieve all the URI pairs that are subjects and objects of `owl:sameAs` statements given a OWL file.

In our extension of the GNS that handles coreferent instances we implement a MATCH-EQ function that takes two URIs as arguments and returns true if it finds a match between the two URIs or any member of their respective equivalent classes (obtained by calling GET-ALL-EQUIVALENTS). We then implement a UNIFY-EQ routine that uses MATCH-EQ to compare the arguments of atoms being unified. We use UNIFY-EQ instead of the regular UNIFY in lines 12, 14 and 20 in Algorithm 1. In addition we use UNIFY-EQ to perform the one sided unification in cl.contains. Since, UNIFY-EQ takes into account the equivalence information of the URIs, it will perform as desired during both node expansion and source selection in the presence of coreferent instances and cl.contains will produce an optimal closed list.

The extension described above is based on the assumption that at system startup the EquivalanceKB contains equivalence information of all URIs known to the system at that time. Therefore, at system startup we call UPDATE-EQUAL-KB with a list of all known URIs to pre-compute all the equivalence classes. This pre-computation approach makes the system less dynamic. The issues related to a more dynamic EquivalenceKB are discussed in Section 5.

In order to ensure that the reasoner has all the relevant equivalence information to perform joins, we need to load all the equivalence information about all the URIs in every relevant source that is being loaded. We do this by first extracting all equivalent URI pairs from a source that is being loaded using EXTRACT-SAMEAS. We then apply GET-ALL-EQUIVALENTS function to each individual URI obtained from the equivalent pairs to get all the equivalence classes for all the URIs in the sources being loaded. We then add this equivalence information directly to the reasoner.

We have implemented EquivalenceKB using a hash table. The key set of the table is the set of all URIs known to the system and the value entries are their respective equivalence classes. The method ADD-TO-KB works as follows. For each equivalent URI pair we union their equivalence classes (obtained from hash table look ups). Then for each URI in the unioned set we replace (or add in case of a new URI) their corresponding entries in the hash table with the unioned set.

We have evaluated the performance of EquivalenceKB using data from the Hawkey project [Pan *et al.*, 2007]. Hawkeye is a knowledge base, that contains 166 million facts from a diverse set of real-world data sources. In addition to the facts, Hawkeye provides several ontology maps and a large number of `owl:sameAs` statements to align the ontologies and the data from these sources . We have used 202,383 `owl:sameAs` statements which align URIs from AIGP, Citeseer and DBLP websites. We ran our experiments on a PC with 3 GB of RAM. The EquivalenceKB took about 3 seconds to build and occupied 7 MB of heap space. One thousand calls to the function GET-ALL-EQUIVALENTS completed in less than half a second.

The number of GET-ALL-EQUIVALENTS call per query will depend on the number of URIs encountered by the system during the source selection and loading. The selectivity of the query, the number of URIs used in the maps that are considered for matching and the number of URIs in the data sources will all contribute to this number. At present we are working on generating synthetic data which will allow us to evaluate the system in a range of scenarios. We however, hypothesize based on our initial analysis of Hawkeye data that a reasonably selective query will require less than 1000 GET-ALL-EQUIVALENTS calls.

4 Related Work

In the database area, the instance corerefence problem is investigated under the overall umbrella of record duplication and data linkage research. Although these are widely studied topics, authoritative surveys [Winkler, 2006; Rahm and Do, 2000] of the research agenda in this area suggest that scalability issues have been addressed mostly for record duplication detection algorithms [Herschel and Naumann, 2008]. The systems that remove detected record duplicates work in batch mode and therefore are not subject to a stringent efficiency requirement. The requirement for such instance-level integration of Semantic Web data are different as the integration needs to occur during query time to account for the dynamic nature of the Web.

The Semantic Web community's primary focus in the area of instance coreference has been to identify coreferent entities using ontological information and various machine learning techniques [Nikolov *et al.*, 2008]. This is a critical element of the overall information integration problem, but as we have argued in the paper, successfully identifying these corefernces is only part of the solution. There is one notable exception. The OKKAM project [Bouquet *et al.*, 2007] proposes a centralized approach to the problem we are addressing. The Entity Name System (ENS) proposed and implemented in the OKKAM project provides global re-usable identifiers for coreferenced entities in the Semantic Web. This is an efficient solution as this does not require a system to load equivalence information during query time. However, the success of this scheme depends on how widely the scheme is adopted by data providers and how quickly the central database can be updated when equivalence information becomes available.

Distributed Description Logic (DDL) offers reasoning service for multiple semantically related ontologies by the use of mappings to combine the inferences of local reasoning of each ontology [Borgida and Serafini, 2003]. The focus of DDL research, however is mostly on ontological queries (queries about classes and properties and their relationships). Serafini *et al.* have recently extended one such DDL system to accommodate data sources and perform instance retrieval queries [Serafini and Tamilin, 2007]. However, it is not clear how the system will perform with a large number of coreferent instances.

Calì *et al.* [2002] have described an algorithm that answers extensional queries (queries about the data as opposed to the schema) posed to a data integration system where the data models are ontological in nature as opposed to relational. Their focus however is in handling complexity in the presence of integrity constraints. Their solution is based on a global schema and they do not address scalability.

We have looked at works on efficient indexing of large RDF repositories. Tous and Delgado [2006] handle a large quantity of RDF information in a sparse matrix. Liarou *et al.* [2007], use Distributed Hash Tables (DHT) to index and locate relevant RDF data sources. Both of these works address issues of general RDF query processing whereas our requirement is very specific: a large collection of `owl:sameAs` statements (possibly) distributed in many physical files over the Web and we have a singular query which is to retrieve all the direct and inferred equivalent URIs of a given URI.

5 Conclusion and Future Work

In this paper we have argued that information integration algorithms need to be extended in order to work effectively and efficiently in the presence of coreferenced entities. We have described one such extension to the GNS algorithm. Our main objective in this paper is to draw attention of the Semantic Web community to the issue of efficient handling of coreferenced instances. We believe this issue has not received the attention it deserves. Although, the extension we propose addresses the problem we pose, we need to examine and evaluate our implementation more extensively to validate our solution.

We note that an alternative solution to the instance coreference handling problem in GNS is to store `owl:sameAs` axioms in the MapKB, in addition to the LAV/GAV rules. We can store equivalence axioms in the form u1=u2, and then allow these rules to match any nodes containing u1(u2) and expand to a node where it is replaced by u2(u1). However, this will significantly increase the size of the KB and the search tree.

At present we pre-compute all the equivalence classes during system startup. However, the system can be more dynamic and the EquivalenceKB more compact, if the EquivalenceKB can be created at query time seeded only by the URIs present in the query (i.e. calling UPDATE-EQUAL-KB with seed URIs that are mentioned in the query). One issue with this approach is that it does not handle the situation where no URIs are mentioned in the query, but equivalence information is needed to establish join conditions between two sources. This could be remedied by calls to UPDATE-EQUAL-KB with new seeds anytime new URIs are used to answer the query, whether as part of the goal node, in a map, or in selected sources. We plan to implement this dynamic approach in the near future and compare its performance with the current implementation and determine if the time penalty for having a more dynamic EquivalanceKB is worth the freshness of the equivalence information.

Although our current implementation provides a very fast lookup time and the in-memory hash table's performance was sufficient for the data set we used, a strict in-memory implementation will not scale up to Web size data. We are exploring other more scalable options.

Acknowledgments

We are grateful to the U.S. Department of Energy for supporting this work under the DE-FG02-05ER84171 SBIR grant.

References

[Borgida and Serafini, 2003] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.

[Bouquet *et al.*, 2007] Paulo Bouquet, Heiko Stoermer, and Daniel Giacomuzzi. OKKAM: Enabling a web of entities. In *Proc. of the WWW2007 Workshop i3:Identity, Identifiers and Identification*, 2007.

[Calì *et al.*, 2002] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. In *CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, pages 262–279, London, UK, 2002. Springer-Verlag.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms: Second Edition*. The MIT Press, Cambridge, MA, 2001.

[Elmagarmid *et al.*, 2007] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, 2007.

[Garcia-Molina *et al.*, 1997] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[Herschel and Naumann, 2008] Melanie Herschel and Felix Naumann. Scaling up duplicate detection in graph data. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1325–1326, New York, NY, USA, 2008. ACM.

[Levy *et al.*, 1996] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *22nd International Conference on Very Large Data Bases*, Bombay, September 1996.

[Liarou *et al.*, 2007] E. Liarou, S. Idreos, and M. Koubarakis. Continuous RDF query processing over DHTs. In *Proceedings of 6th International Semantic Web Conference / 2nd Asian Semantic Web Conference (ISWC/ASWC 2007)*, pages 324–339, 2007.

[Nikolov *et al.*, 2008] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. Handling instance coreferencing in the KnoFuss architecture. In Paolo Bouquet, Harry Halpin, Heiko Stoermer, and Giovanni Tummarello, editors, *IRSW*, volume 422 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[Pan *et al.*, 2007] Zhengxiang Pan, Abir Qasem, Sudhan Kanitkar, Fabiana Prabhakar, and Jeff Heflin. Hawkeye: A practical large scale demonstration of semantic web integration. In *In Proc. of the 3rd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS'07)*, 2007.

[Qasem *et al.*, 2008a] Abir Qasem, Dimitre A. Dimitrov, and Jeff Heflin. Efficient selection and integration of data sources for answering semantic web queries. In *ICSC 08: Proceedings of the Second IEEE International Conference on Semantic Computing*. IEEE Computer Society Press, 2008.

[Qasem *et al.*, 2008b] Abir Qasem, Dimitre A. Dimitrov, and Jeff Heflin. Goal node search for semantic web source selection. In *WI 08: Proceedings of the International Conference on Web Intelligence*. IEEE Computer Society Press, 2008.

[Rahm and Do, 2000] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[Serafini and Tamilin, 2007] Luciano Serafini and Andrei Tamilin. Instance migration in heterogeneous ontology environments. In *Proceedings of 6th International Semantic Web Conference / 2nd Asian Semantic Web Conference (ISWC/ASWC 2007)*, pages 452–465, 2007.

[Tous and Delgado, 2006] Rubén Tous and Jaime Delgado. A vector space model for semantic similarity calculation and OWL ontology alignment. In *20th International Conference on Database and Expert Systems Applications (DEXA)*, pages 307–316, 2006.

[Winkler, 2006] William E Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006.