Trust and Hybrid Reasoning for Ontological Knowledge Bases

Hui Shi
Department of Computer Science
Old Dominion University
Norfolk, VA. 23529
001-(757) 683-6001
hshi@cs.odu.edu

Kurt Maly
Department of Computer Science
Old Dominion University
Norfolk, VA. 23529
001-(757) 683-6001
maly@cs.odu.edu

Steven Zeil
Department of Computer Science
Old Dominion University
Norfolk, VA. 23529
001-(757) 683-6001
zeil@cs.odu.edu

ABSTRACT
Projects such as Libra and Cimple have built systems to capture knowledge in a research community and to respond to semantic queries. However, they lack the support for a knowledge base that can evolve over time while responding to queries requiring reasoning. We consider a semantic web that covers linked data about science research that are being harvested from the Web and are supplemented and edited by community members. We use ontologies to incorporate semantics to detect conflicts and resolve inconsistencies, and to infer new relations or proof statements with a reasoning engine. We consider a semantic web subject to changes in the knowledge base, the underlying ontology or the rule set that governs the reasoning. In this paper we explore the idea of *trust* where each change to the knowledge base is analyzed as to what subset of the knowledge base can still be trusted. We present algorithms that adapt the reasoner such that, when proving a goal, it does a simple retrieval when it encounters trusted items and backward chaining over untrusted items. We provide an evaluation of our proposed modifications that show that our algorithm is conservative and that it provides significant gains in performance for certain queries.

Categories and Subject Descriptors
I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving – *Answer/reason extraction, Inference engines, Logic programming.*

Keywords
Semantic web, ontology, backward chaining, trust

1. INTRODUCTION
Many knowledge bases organize information using ontologies. Ontologies can be used together with a reasoning engine to infer new relations. Consider a potential Ph.D. student who is trying to find out what are the schools and who are the professors doing groundbreaking research in his thesis area. Consider a faculty member who might ask, "Is my record good enough to be tenured at other schools?" The system implied by these queries is an example of a semantic web where the underlying knowledge base covers linked data about science research. This data would be continually harvested from the Web and supplemented and edited by community members.

The query examples given above also imply that the system not only supports querying of facts but also rules and reasoning as a mechanism for answering queries. Furthermore, many of the rules underlying such reasoning would be particular to an individual or small group making the query. A relationship such as "doesGroundbreakingResearchIn(person,field)" is necessarily an expression of a subjective and individual perspective. We envision, therefore, a system in which groups may add, employ, and sometimes share custom rules that describe properties of interest when posting new queries.

We consider a semantic web subject to frequent changes in the underlying knowledge base (for instance, the addition of new researchers and or new papers). We also consider less frequent changes in the underlying ontology (for instance, addition of a new data type 'ResearchLab') or in the rule set that governs the reasoning (for instance, some members of the community want a relation for researchers such as 'TopResearcher' in a field).

Queries may be composed of mixtures of clauses answerable directly by access to the knowledge base or indirectly via reasoning applied to that base. Backward chaining reasoners work well with changes to the knowledge base but do not scale well beyond in-memory systems although the size of problems that can be handled can be improved via optimization techniques [15-17].

In this paper we explore the idea of 'trust' where each change to the knowledge base is analyzed as to what subset of the knowledge base can still be trusted, assuming that it was fully materialized before the change. We present algorithms that adapt a reasoner so that, when proving a goal, it does a simple retrieval when it encounters trusted items and backward chaining over untrusted items.

2. BACKGROUND AND RELATED WORK
A number of projects (e.g., Libra [1] and Cimple [2]) have built systems to capture limited aspects of community knowledge and to respond to semantic queries. However, they lack the level of community collaboration support that is required to build a knowledge base system that can evolve over time, both in terms of the knowledge it represents as well as the semantics involved in responding to qualitative questions involving reasoning.

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579033

Many knowledge bases [3-4] organize information using ontologies. An ontology represents knowledge as a set of concepts within a domain and by relationships between pairs of concepts. The ontology is used to model a domain, to instantiate entities, and to support reasoning about entities. Common methods for implementing rule based reasoning over ontologies are forward chaining and backward chaining [5].

Forward chaining (materialization) is a form of data-driven reasoning, which starts with the known data in the knowledge base and applies modus ponens in the forward direction, deriving and adding new consequences until no more inferences can be made. Forward-chaining can answer queries efficiently by looking up result set directly without any additional reasoning, but at the cost of an expensive pre-computation.

Backward chaining is a form of goal-driven reasoning, which starts with goals from the consequents, matching the goals to the antecedents to find data satisfying the consequents. Backward-chaining does not require expensive pre-computation, but increases the cost of reasoning when answering each query.

As a general rule, forward chaining is a good method for a static knowledge base and backward chaining is good for the more dynamic cases.

Owlim [6] and Oracle 11g [7] inferred facts using forward-chaining technique. 4store [8] applied the RDFS rules using backward-chaining. Virtuoso [9] implemented a mixture of forward-chaining and backward-chaining. Jena [10] supported three ways of inferencing: forward-chaining, limited backward-chaining and a hybrid of these two methods.

Hybrid reasoning [10-12], combining forward-chaining and backward-chaining, has been adopted to find a tradeoff between pre-computation before querying and reasoning when answering each query, in different ways. Urbani et al. [11] performed partial materialization of a fixed set of selected queries before running their backward-chaining algorithm. Jena [10] also implemented a hybrid rule reasoner. Song [12] assembled two kinds of reasoning technology: tableau and resolution together to improve reasoning performance.

To our best knowledge, our research is the first that analyzes what part of derived facts can still be directly retrieved (trusted) after changes have been made to a knowledge base. By retaining the unaffected facts obtained from materialization and using backward chaining for facts affected by the change we obtain a hybrid reasoning engine that can cope with change in a knowledge base.

3. A HYBRID REASONING ALGORITHM

Let us assume that the knowledge base contains all known and derived facts such that any query can be directly answered by retrieving the corresponding truths. In such an environment we now introduce a change and analyze the impact of that change.

We consider three categories of changes: changes to the ontology, changes to custom rules and changes to instances.

- Changes to the ontology may entail adding, removing or modifying concepts or properties in the ontology. Such changes would likely be rare in a mature knowledge base.
- Changes to custom rules, such as the afore-mentioned rule to define "groundbreaking researcher," are likely to be more common, as groups posing queries might iteratively tweak and debug their formulations to produce more satisfactory results.
- Changes to instances include adding, removing or modifying instances (facts) in the storage, such as updating a publication list of a professor. Such changes are likely to be very common if the knowledge base is populated by actively harvesting information from the web.

In a knowledge base that relies on materialization via forward-chaining, all derivable conclusions from known facts and rules are assumed to have been written into the knowledge base. This leads to fast responses to queries because all queries can be answered by direct search and retrieval. No reasoning is required at the time of the query.

The same queries could presumably be answered in a non-materialized knowledge base containing only harvested facts and rules via backward chaining from the properties mentioned in the query. This would typically be considerably slower than a direct lookup in a materialized knowledge base.

The drawback to relying on materialization is a loss of agility in responding to changes. Materialization of a large knowledge base is potentially time-consuming. It may require deferring queries for many hours or, alternatively, issuing results that are incomplete or incorrect.

Consider the following example. A student, student0 has enrolled in a course, Course0. This piece of information has been discovered by a harvester and added to the knowledge base. Now if someone were to pose a query "Who is enrolled in Course0?", this query could be answered immediately by direct lookup in the knowledge base. No reasoning would be required. However let us also posit that the knowledge base already contains the fact that Prof0 teaches Course0. If a query were immediately posed "Who is being taught by Prof0?", such a query could not be answered without reasoning about the implications of a rule

```
enrolledIn(?Student,?Course?), teaches(?Faculty,?Course)
    :- isTaughtBy(?Student,?faculty)
```

But a search for materialized isTaughtBy instances in the knowledge base will fail to turn up the relationship between Professor0 and the newly enrolled Student0, until the knowledge base is re-materialized to incorporate this and all other recent changes.

A hybrid reasoning algorithm can use backward chaining over "untrusted" portions of a knowledge base while using direct lookup to recover previously materialized conclusions from the "trusted" portion. Such an algorithm is shown here:

```
substitutions prove (Goal g)
{
  if (g is trusted)
     retrieve substitutions M from knowledge base by
       direct lookup of g;
     return M;
  else {
       substitutions M = empty;
       for each rule R and substitution σ1 such that
          the head of R σ1 matches g {
            M1  = proveTheRuleBody (R.body, σ1);
            if (M1 is not empty) // proof of rule succeeded
               {
                 substitutions M2 = all substitutions in M1
                    for variables in the head of R;
```

```
                M += M2;
            }
        }
    }
  return M;
}
```

In the above algorithm, R denotes a rule, consisting of a rule body (premises) and a rule head (conclusion), the rule head is true only when the rule body is true.

Our prove function returns all substitutions for the variables in the goal for which that goal is provable. It does this by consulting each rule matching the goal and attempting to find substitutions satisfying the body of that rule. A key step is the test to see if the proof goal is "trusted." The results of this test determine whether we simply look up previously materialized instances or engage in a backward-chaining proof.

```
bool, substitutions proveTheRuleBody (body, substitution)
{

  substitutions M  = empty;
   for each goal g in body from substitution {
      M1 = prove(g);
      if (M is empty)
         return false,empty;
      M = joinSubstitutions(M,M1);
   }
   return true, M;
}
```

When we need to prove a clause in the rule body, we attempt to prove each goal in the rule body one by one, which is a recursive process. In our actual implementation [17], we employed OLDT [14] and memorization to avoid deep recursion. The bindings from earlier goals would be substituted into the current goal for subsequent proof. After we prove the current goal, we join the new substitutions from that proof with the prior substitutions. In joinSubstitutions, we iterate over two sets of substitutions and compose every pair of substitutions in the cross product with common values.

The advantage of such a hybrid algorithm is that it allows the knowledge base maintainers to defer expensive re-materializations for long periods of time (long, at least, compared to the frequency of changes) while still permitting accurate and timely responses to incoming queries.

4. CONSERVATIVE TRUST ASSESSMENT

The preceding hybrid algorithm depends upon the idea of knowing when the currently materialized instances corresponding to a proof goal can be trusted to be correct and complete.

We will say that a proof goal p(?X,?Y) is *trustworthy* if all instances of that goal derivable from facts and rules in the knowledge base are present in that knowledge base as instances.

In practice, we are unlikely to be able to identify precisely all goals that are trustworthy except by materializing the knowledge base, which, by definition, forces all goals to be trustworthy. We therefore seek less expensive options to approximate the set of trustworthy goals, a less expensive option for dividing the set of possible proof goals into trusted and untrusted sets.

A partition into trusted and untrusted sets is called *conservative* if no untrustworthy goals are trusted. If we are conservative in such an approximation to the collection of trustworthy goals, then our hybrid reasoner can be relied upon to give accurate responses.

An apparently plausible approach to a conservative trust rule would be *property-based trust*: assume that any property P that was involved in a change (e.g., if a new instance P(x0,y0) was added to the knowledge base) is itself untrusted and then to take the closure of the "is used as a premise of" relation, that is, if an untrusted property Q occurs in the body of a rule used to prove R

```
…, Q(x,y), … :- R(w,z)
```

then R is also untrusted.

For example, suppose that a student, student0 has enrolled in a course, Course0, taught by Prof0. The addition of a new fact enrolledIn(Student0,Course0) to the knowledge base would cause the property enrolledIn to be untrusted. In addition, given the rule presented earlier deriving isTaughtBy(?Student,?Faculty) from (in part) enrolledIn instances, the property isTaughtBy would also untrusted.

The attraction of this definition of "untrusted" is that it requires analysis of only the rules in the knowledge base without consulting with the far more numerous instances. A knowledge base of many millions of triples might be expressed in terms of a few hundreds of properties and a comparable number of rules, making this definition of trust far easier to compute than the true trustworthy set.

Unfortunately, this simple procedure breaks down in the face of "meta-rules" in the knowledge base, rules that permit reasoning about properties themselves. For example, suppose that student, Student0, just got his degree from University0. The instance "degreeFrom(student0, University0)" is added to the knowledge base. We will posit that there are rules such that the property degreeFrom is an inverse property of hasAlumnus. The inverse rule implies that

```
?P(?X, ?Y), inverse(?P,?Q) :- ?Q(?Y,?X)
```

Immediately after adding the new fact to the knowledge base, queries such as "what alumni/alumnae does university0 have?" would not respond with Student0. The hasAlumnus property is not trustworthy, but it would actually be left as trusted by our initial trust approximation. A more sophisticated definition of trust is required.

One possibility would be to expand the set of untrusted properties via special handling of the meta-rules common to RDF and OWL. As we will show, however, in our experimental results below, prototypes of such an expanded definition of property-based trust demonstrated that many simple changes to a knowledge base could then result in significant fractions of the knowledge base being marked as untrusted. We concluded that properties do not offer a detailed enough discrimination to serve as a practical basis for trust.

We propose instead a concept of *pattern-based trust:* a pattern P(X,Y) (where X and Y could be ground instances or free variables) is untrusted if it matches a change to the knowledge base or if it can be derived from a rule with an untrusted pattern as a premise.

For example, suppose again that a student, student0 has enrolled in a course, Course0, taught by Prof0. The addition of a new fact enrolledIn(Student0,Course0) to the knowledge base would cause the pattern enrolledIn(Student0,Course0) to be untrusted. In addition, given the rule presented earlier deriving

isTaughtBy(?Student,?Faculty), the pattern isTaughtBy(Student0,Prof0) would also be untrusted.

In our hybrid prove algorithm, presented in the prior section, the test to see if a goal g is trusted is now interpreted as "if g cannot be unified with any untrusted pattern". Hence queries and proof goals involving patterns such as isTaughtBy(?S,Prof0) and isTaughtBy(?S,?P) would also be treated as untrusted.

Of importance is the fact that patterns (and therefore potential queries and proofs) involving other students and other faculty (e.g., "who is taught by Prof1?") remain trusted and so could be answered by direct lookup with no reasoning.

The pattern-based trust marking algorithm below will work with these meta-rules as well as customized rules. We accumulate a set of already untrusted patterns by running the collectUntrustedDueTo algorithm iteratively on each new change.

```
setOfPatterns collectUntrustedDueTo
    (oneChange, existingUntrustedSet)
{
  untrustedSet = {oneChange};
  for each rule R (p1^p2...^pi...^pn => q) and
    each pi matching oneChange
    {
      retrieve substitutions M from knowledge base by
        direct lookup of pi;
      untrustSet+= propagateUntrustForward
        ([p0..pi-1,pi+1...pn],q, M}, existingUntrustedSet)
    }
  existingUntrustedSet += untrustSet;
  discard from existingUntrustedSet  any patterns
    that are specializations of other elements.
  return existingUntrustedSet;
}
```

The collectUntrustedDueTo function collects untrusted patterns for one single change to the knowledge base, assuming we have already had an existing untrusted pattern set. The first time this algorithm runs after materialization that set will be empty. The one single change would be added to the untrusted set first. Then we check each rule in the rule set to see if we can propagate the "untrust" forward by a limited, specialized analogue of forward chaining. At last, we add our untrusted set produced from the above one change to the existing untrusted set, discarding any patterns that are specializations of other elements.

```
setOfPatterns propagateUntrustForward (premises, conclusion,
substitutionSet, existingUntrustedSet)
{
  untrustedSet = {};
  for each premise p  in premises
    {
     patternSet = {};
     goalList = get unified goals by replacing variables
       in p from substitutionSet;
     for each goal g in goalList
      {
        retrieve instances r from knowledge base
          by direct lookup of g;
        retrieve results r1 from realized patterns in
           existingUntrustedSet  by direct lookup of g;
        r += r1;
        if (r is empty)
          return empty;
        if (the size of r <threshold)
           untrustedpatternSet + = r;
         else if (the size of  r >=threshold)
           untrustedpatternSet + = g;
      }
     substitutions M = unify p with untrustedpatternSet;
     substitutionSet =
       joinSubstitutions(substitutionSet,M);
    }
  untrustedSet = substitute variables in conclusion
    with substitutionSet;
  return untrustedSet;
}
```

The propagateUntrustForward function goes through each premise of the rule to compose and propagate the untrusted substitutions. The untrusted substitutions from the earlier solutions are substituted into the upcoming premise to yield multiple instances of that clause as goals for subsequent proof. When we prove a premise, we need to prove each unified goal produced from untrusted substitutions by replacing variables in the premise. Upon proving a unified goal, we retrieve matched instances from the knowledge base and existingUntrustedSet by a direct lookup of the unified goal.

We use a threshold to determine whether the actual matched results or the unified goal itself should be added to the untrusted substitutions. If the number of matched instances is comparatively large, we use a pattern that can represent the whole set of matched instances in the untrusted substitutions instead of the large set of matched instances itself. Including all the untrusted instances in the untrusted set would make it inefficient when we determine if a goal is trusted or not in this marking algorithm, Finally, the rule's conclusion (head), with appropriate substitution, is added to the untrusted set.

For example, consider a scenario based on LUBM [13]. A university, University0 has hired professor, Fullprofessor0. This piece of information worksFor(Fullprofessor0, University0) has been discovered by a harvester and added to the knowledge base. Given the OWL Horst rule set, according to our pattern-based marking algorithm, the untrusted patterns are:

worksFor (Fullprofessor0, University0)
member (University0, Fullprofessor0)
memberOf (Fullprofessor0, University0)

Now if someone were to pose a query "Who are members of University0?", given that pattern memberOf(Fullprofessor0, University0)" is untrusted, we need to reason using backward chaining.

On the other hand, if someone were to pose a query "Who are members of University1", given that the pattern memberOf(?x, University1) is trusted, all the members of University1 can still be retrieved by a direct look up in the knowledge base.

5. EXPERIMENTS

In this Section, we further explore the impact of trust rules on reasoning by conducting and presenting reports involving two sets of benchmarking experiments. We also describe preliminary experiments that explore how the percentage of trusted facts in the knowledge base affects the performance of the hybrid algorithm.

First, we compare the number of objects in the knowledge base marked as untrusted by our property-based algorithm to what should be really untrusted. This provides an indication of how many unnecessary reasoning steps the hybrid algorithm would have to go through.

Table 1. Results for property-based marking algorithm

Changes	Actual # new properties	Actual # new facts	# untrusted properties
Adding a new class	2	3	12
Add a subclass relationship between two new classes	2	6	12
Add new Class as subClass of existing class	2	5	12
Adding a new Property	2	2	12
Add a new Property as subPropertyOf of another new Property	2	4	12
Add new Property as subPropertyOf of existing Property	2	3	12
Add new Class as domain to a new Property	3	5	13
Add new Class as range to a new Property	3	5	13

Table 1 shows that property-based marking greatly exaggerates the number of untrusted properties. It compares the number of properties that should be untrusted after a change vs. the numbers the property-based algorithm produces. Even in a knowledge base with many millions of triples, the number of distinct properties is likely to be counted in the tens to (very) low hundreds, so the increase shown there in the number of properties marked as untrusted would likely have a significant effect on query processing. This is made worse by the fact that the experiments showed that many properties marked as untrusted were ontological meta-rules such as all subclass relations. As an example, under LUBM(1) the properties marked as untrusted match an average of 97,300 triples out of 149,894, which is about 65% of the knowledge base. These results led us to set aside property-based trust marking in favor of the finer discrimination afforded by pattern-based marking.

In contrast the average number of patterns added by the pattern-based marking algorithm for the same changes as in Table 1 produces the same number of properties as the 'actual' columns show.

Table 2. Query response time (ms) after adding student

	Hybrid	Backward	Forward (load +query)
LUBM1	93	490	960+3.4
LUBM10	546	1,060	7,800+150
LUBM40	2,548	9,100	350,000+5,100

The second set of experiments provides a comparison of performance of our hybrid pattern-based proof algorithm against our regular, optimized backward chaining algorithm [17] and against the OWLIM forward chaining algorithm using benchmark knowledge bases LUBM1, LUBM10, and LUBM40, of size 100,839, 1,272,871, and 5,307,754 objects respectively.

Table 2 shows the comparison of response times for LUBM query 2 [13] for these three algorithms after adding an existing student as a member of an existing department to the knowledge base.

Table 3 shows the comparison of response times for LUBM query 6 [13] for these three algorithms after adding a new undergraduate student.

Both tables show that the hybrid algorithm, though slower on a query-by-query basis than forward chaining, is faster than backward chaining and at least an order of magnitude faster than re-materializing the knowledge base (which is the time that would be required to re-materialize a knowledge base after a change).We have performed preliminary experiments on determining the factors that affect the performance of the hybrid algorithm. We investigated the ratio of trusted to untrusted facts in the knowledge base after a set of changes and compared the performance of the three algorithms (backward chaining, forward chaining, hybrid) for different sizes of the knowledge base and selected queries. The following sample scenarios illustrate the results of the preliminary experiments.

Scenario 1
Changes:
MiddleWork rdf:type rdfs:Class
MiddleWork rdfs:subclassOf Work
Course rdfs:subclassOf MiddleWork
Query: ?x rdf:type Work

Scenario 2
Changes:
SupermemberOf rdf:type rdf:Property
memberOf rdfs:subPropertyOf SupermemberOf
Query: ?x SupermemberOf ?y

Scenario 3
Changes:
MiddledegreeFrom rdf:type rdf:Property
MiddledegreeFrom rdfs: subPropertyOf degreeFrom
undergraduateDegreeFromrdfs: subPropertyOf MiddledegreeFrom
Query: ?x degreeFrom ?y

The percentage of untrusted facts in the knowledge base after executing the pattern-based marking algorithm ranges from close to 0 to a high of 10% in all experiments. The percentage of untrusted patterns ranges from close to zero to 5%. For scenarios

Table 3. Query response time (ms) after adding undergraduate student

	Hybrid	Backward	Forward (load +query)
LUBM1	452	180	960+240
LUBM10	1,575	1,170	7,800+1,200
LUBM40	3,525	43,000	350,000+5,300

1 and 2, the hybrid algorithm outperforms the backward chaining algorithm in terms of query response time significantly. For LUBM(50), the query response times in scenario 1 are 1,887ms and 3,229ms respectively. The query response times in scenario 2 are 9,937ms and 11,216ms, respectively. In scenario 3 the backward chaining algorithm outperforms the hybrid algorithm in terms of query response time (7,238ms and 5,054ms respectively). The reason for this is that, in scenario 3, the percentage of untrusted triples in the whole knowledge base is about 10% and the queries require resolving patterns mostly located in the untrusted list. For the queries selected, the hybrid algorithm outperforms the regular backward chaining algorithm by 30 percent on the average for the selected set of experiments.

The experiments reported here suggest that a hybrid reasoner based on trust can be effective on some moderately sized knowledge bases. In considering the likely scalability of this approach, we may consider scaling to both larger knowledge bases and to bases subject to more increasingly frequently change.

Because the pattern-based trust markup is similar in structure to forward chaining, as the size of the knowledge base increases, the time to assess trust should grow at a rate no higher than, and possibly lower than, the increase in time to re-instantiate. An important contributory factor will be the overall degree of inter-connection within the knowledge base semantics. A loosely connected network will lead to faster termination of the trust marking algorithm. The experiments reported here may actually understate the potential savings, as we expect that LUBM is more tightly inter-connected than many practical knowledge bases.

6. CONCLUSION

To improve the performance of reasoning over a large knowledge base that is frequently changed, we have introduced the concept of trust. After a change has been made to a previously materialized knowledge base, we mark all patterns in the knowledge base we believe are related to the change.

We then have modified a backward chaining algorithm to reason only when it encounters untrusted patterns in proving a goal. Otherwise the algorithm will simply do a retrieve operation. We have implemented these modifications and in two sets of experiments have shown that a pattern-based marking algorithm errs on the conservative side at an acceptable level and that the performance comparison to a forward chaining algorithm and a pure backward chaining algorithm clearly shows that our hybrid algorithm is better in almost all cases tested. In a benchmark test [13] with 5 million objects, the running time was reduced by as much as a factor of 2 when compared to straight backward chaining and was better than forward chaining when including materialization time.

We have not yet explored the impact of long sequences of individual changes on the marking algorithm time nor subsequently on the hybrid reasoner. The marking algorithm includes a concept of generalization of related untrusted patterns, which may prove critical when scaling to more frequent changes. The degree of inter-connection may prove to be a relevant factor here as well. In future work, we plan to explore the performance of the trust marking algorithm and of the hybrid reasoner as a function of the fraction of the knowledge base that is untrusted, a measure that would combine both the number of changes and the extent of their impact throughout the semantic graph.

7. REFERENCES

[1] Nie, Z., Zhang, Y., Wen, J. and Ma, W. 2005. Object-level ranking: bringing order to web objects. In *Proceedings of the 14th international World Wide Web conference* (Chiba, Japan, May 10-14, 2005). WWW2005. ACM, New York, NY, 567-574. DOI= http://doi.acm.org/10.1145/1060745.1060828.

[2] Doan, A., et al. 2006. Community information management. *IEEE Data Eng. Bull*. 29, 1 (Mar. 2006), 64-72.

[3] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S. and Becker, C. 2009. DBpedia-A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web.* 7, 3 (Sep. 2009), 154-165, DOI= http://dx.doi.org/10.1016/j.websem.2009.07.002.

[4] Suchanek, F., Kasneci, G. and Weikum, G. 2008. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*. 6, 3 (Sep. 2008), 203-217, DOI= http://dx.doi.org/10.1016/j.websem.2008.06.001.

[5] Russell, S. J. and Norvig, P. 1995. *Artificial intelligence: a modern approach, 1st ed*. Prentice hall, Inc., Upper Saddle River, New Jersey.

[6] Kiryakov, A., Ognyanov, D. and Manov, D. 2005. OWLIM–a pragmatic semantic repository for OWL. In *Proceedings of the 6th international conference on Web Information Systems Engineering* (New York, USA, November 20-22, 2005). WISE'05. Springer, New York, NY, 182-192. DOI= http://dx.doi.org/10.1007/11581116_19.

[7] Oracle Corporation. 2013. Oracle Database 11g R2 [retrieved: Feb., 2014], Available from: http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/.

[8] Garlik. 2009. Scalable RDF Storage [retrieved: Feb., 2014], Available from: http://4store.org/.

[9] Erling, O. and Mikhailov, I. 2009. RDF Support in the Virtuoso DBMS. *Networked Knowledge-Networked Media*, vol.221. Springer, New York, NY, 7-24. DOI= http://dx.doi.org/10.1007/978-3-642-02184-8_2.

[10] The Apache Software Foundation. 2013. Apache Jena [retrieved: Feb., 2014], Available from: http://jena.apache.org/.

[11] Urbani, J., Prio, R., Harmelen, V. F. and Bal, H. 2013. Hybrid reasoning on OWL RL. *Semantic Web Journal* (Sep. 2013).

[12] Song, W., Spencer, B. and Du, W. 2011. Hybrid reasoning for ontology classification. In *Proceedings of 24th Canadian Conference on Artificial Intelligence* (St. John's, Canada, May 25-27, 2011). Canadian AI 2011. Springer, New York, NY, 372-376. DOI= http://dx.doi.org/10.1007/978-3-642-21043-3_44.

[13] Guo, Y., Pan, Z., and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*. 3, 2-3 (Oct. 2005), 158-182. DOI= http://dx.doi.org/10.1016/j.websem.2005.06.005.

[14] Tamaki, H. and Sato, T. 1986. OLD resolution with tabulation. In *third international conference on logic programming* (London, United Kingdom, July 14-18, 1986). ICLP1986. Springer, New York, NY, 84-98. DOI= http://dx.doi.org/10.1007/3-540-16492-8_66.

[15] Shi, H., Maly, K., Zeil, S. and Zubair, M. 2011. Comparison of Ontology Reasoning Systems Using Custom Rules. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics* (Sogndal, Norway, May 25-27, 2011). WIMS'11. ACM, New York, NY. DOI= http://doi.acm.org/10.1145/1988688.1988708.

[16] Shi, H., Maly, K. and Zeil, S. 2013. Query Optimization in Cooperation with an Ontological Reasoning Service. In *the Fifth International Conferences on Advanced Service Computing* (Valencia, Spain, May 27-June 1, 2013). Services Computation 2013, IARIA XPS, 26-32.

[17] Shi, H., Maly, K. and Zeil, S. 2014. Optimized Backward Chaining Reasoning System for a Semantic Web. In *the Fourth International Conference on Web Intelligence, Mining and Semantics* (Thessaloniki, Greece, June 2-4, 2014). WIMS'14. ACM, New York, NY. (submitted)