OnEQL: An Ontology Efficient Query Language Engine for the Semantic Web

Edna Ruckhaus and Eduardo Ruiz and María Esther Vidal

Universidad Simón Bolívar
Caracas, Venezuela
{ruckhaus,eruiz, mvidal}@ldc.usb.ve

Abstract. In this paper we describe the OnEQL system, a query engine that implements optimization techniques and evaluation strategies to speed up the evaluation time of querying and reasoning services in the Semantic Web. To identify execution plans that reduce the cost of evaluating a query, we developed a twofold optimization strategy that combines cost-based optimization and Magic Sets techniques. In the first stage, a dynamic programming-based algorithm is used to identify an ordering of predicates in the query that minimizes its estimated evaluation cost. In the second stage, Magic Sets techniques are used to push down query selections into the OnEQL ontology representation, in order to reduce the number of facts inferred during query evaluation. Additionally, we developed three physical operators that execute the sideways passing of bindings during the evaluation of the execution plan. To illustrate the advantages of this approach, we report the results of an experimental study over the most popular health ontologies.

1 Introduction

Ontologies play an important role in the Semantic Web, and provide the basics for the definition of concepts and relationships that make global interoperability possible. Knowledge represented in ontologies can be used to annotate data, distinguish similar concepts, and generalize and specialize concepts.

A great number of ontologies have become available under the umbrella of the Semantic Web. In particular, for the health domain, large ontologies have been defined, for example, MeSH [15], Disease [4], Galen [6], and EHR_RM [5], which are commonly used by the health and bioinformatics community to find solutions for a variety of problems. These ontologies are specified in different standard languages such as XMLSchema [25], OWL [14] or RDFS [2]; and regular requirements are expressed using query languages such as SPARQL [17] or RQL [12]. OWL is a markup language that extends the graph-based model used by RDF and provides complex structures and different levels of complexity on top of XML. OWL is commonly used to share and publish information encoded in an ontology. On the other hand, SPARQL is an RDF-based query language that enables users to select portions of an ontology that satisfy certain patterns or conditions. In the Semantic Web, OWL and SPARQL have become standards to publish and query data, respectively. In this paper, we propose the OnEQL system which interoperates between different ontology representations and query languages. In the current version of OnEQL, we consider OWL Lite ontologies, and a subset of SPARQL that includes basic patterns; we do not consider optional patters, union of patterns and filters.

In OnEQL, an ontology is represented in a canonical form which is independent of the specific language used to define it. Accordingly, each ontology is modeled as a deductive database called a Deductive Ontology Base (DOB) composed by an extensional base EOB and an intensional base IOB. Knowledge explicitly described in the ontology is represented in the EOB, while knowledge implicitly encoded is modeled by a set of deductive rules that comprise the IOB. Additionally, to diminish the impact of large sets of explicit and implicit ontology facts in the performance of reasoning and querying tasks, OnEQL provides query optimization and evaluation techniques.

The OnEQL query optimization technique is a twofold strategy that combines cost-based optimization and Magic Sets approaches. The idea of this technique is to first identify an ordering which corresponds to an optimal top-down evaluation of the query, and then apply Magic Sets to transform the DOB into a program specific for the query. The evaluation of the rewritten DOB imitates a top-down computation using a bottom-up strategy [18]. During a bottom-up computation, each fact is computed once. Therefore, if intermediate facts in a query are inferred several times, the bottom-up computation of the rewritten DOB w.r.t. the optimal ordering, can be more efficient than the top-down computation of this ordering.

To identify an optimal ordering of a query, the cost-based optimization technique is defined in terms of a dynamic programming algorithm. To traverse the space of the plans of a query, the algorithm uses a cost model that estimates the cost of evaluating a plan. The cost is defined as an estimate of the number of facts inferred during the execution of the corresponding plan [21].

On the other hand, the Magic Sets approach transforms the DOB into a program where the selections in the query are pushed down into the program, and the number of intermediate facts required to answer the query is minimized.

In this paper we explain the mechanisms implemented in OnEQL, and report their behavior for the ontologies Galen and EHR_RM. The paper is composed of four additional sections. In Section 2, OnEQL is described. Section 3 reports our experimental results for query and reasoning tasks in synthetic and real-world ontologies. In Section 4, we compare existing approaches. Finally, we give our conclusions in Section 5.

2 The OnEQL System

The OnEQL system develops evaluation strategies, and cost-based and heuristic-based optimization techniques for Web ontologies. Its main features are the following:

- Ontologies may be loaded and browsed. The interface is based on the SWOOP Mindswap Project [11].
- Ontologies are translated to the DOB canonical form according to the language in which the ontology is defined. Currently we work the OWL Lite [14] ontology language.
- It offers two modes for queries: SPARQL queries written by the user, or queries expressed in the canonical form. With SPARQL, a user can query RDF triples that encode ontologies written in OWL Lite. The query engine does inference during query evaluation when it encounters the (translated) intensional IOB meta-predicates. To test the OnEQL optimization techniques, the system presents fifty randomly-generated DOB queries which the user may execute. The original and optimized queries are presented, and also their evaluation cost. Additionally, the number of results is indicated and the first fifty results are displayed.

In Figure 1, we present the OnEQL architecture. The techniques implemented in this system have been previously reported in [21].

The OnEQL architecture is comprised of two main components: a query engine and an ontology manager. The query engine evaluates user queries against a specific OWL ontology, and outputs the set of facts that satisfy the query in the input ontology. It is composed of a query parser, a query optimizer and an execution engine. On the other hand, the ontology manager translates OWL ontologies into the OnEQL canonical representation and extracts the statistics that describe the ontologies.

In OnEQL, an OWL ontology is modeled as a deductive database of meta-level predicates called a Deductive Ontology Base (DOB). The extensional database comprises all the ontology statements that represent the explicit ontology knowledge. The intensional database corresponds to the set of deductive rules that define the semantics of the ontology language. Specifically, we represent an OWL Lite ontology as a DOB knowledge base, and a SPARQL query as a DOB query. It should be noted that our current version of OnEQL only considers SPARQL basic patterns.

Table 1 illustrates the EOB and IOB built-in predicates for an OWL Lite subset[1]. Note that some predicates refer to domain concepts (e.g., `isClass, areClasses`), and some to instances (e.g., `isIndividual, areIndividuals`).

There are two catalogs: one stores the DOB, and the other maintains statistics that describe the facts encoded in the DOB. Among the statistics we can mention: cost of inferring implicit facts, cardinality of explicit and implicit facts, and number of different values of each attribute. The Analyzer extracts these statistics from the ontology for explicit and implicit facts, and stores them into the catalog.

A hybrid cost model is used to estimate the cardinality and evaluation cost of the DOB predicates that represent the ontology's explicit and implicit facts [21]. Explicit fact estimates are computed using traditional relational database cost

[1] We assume that the class *owl:Thing* is the default value for the domain and range of a property.

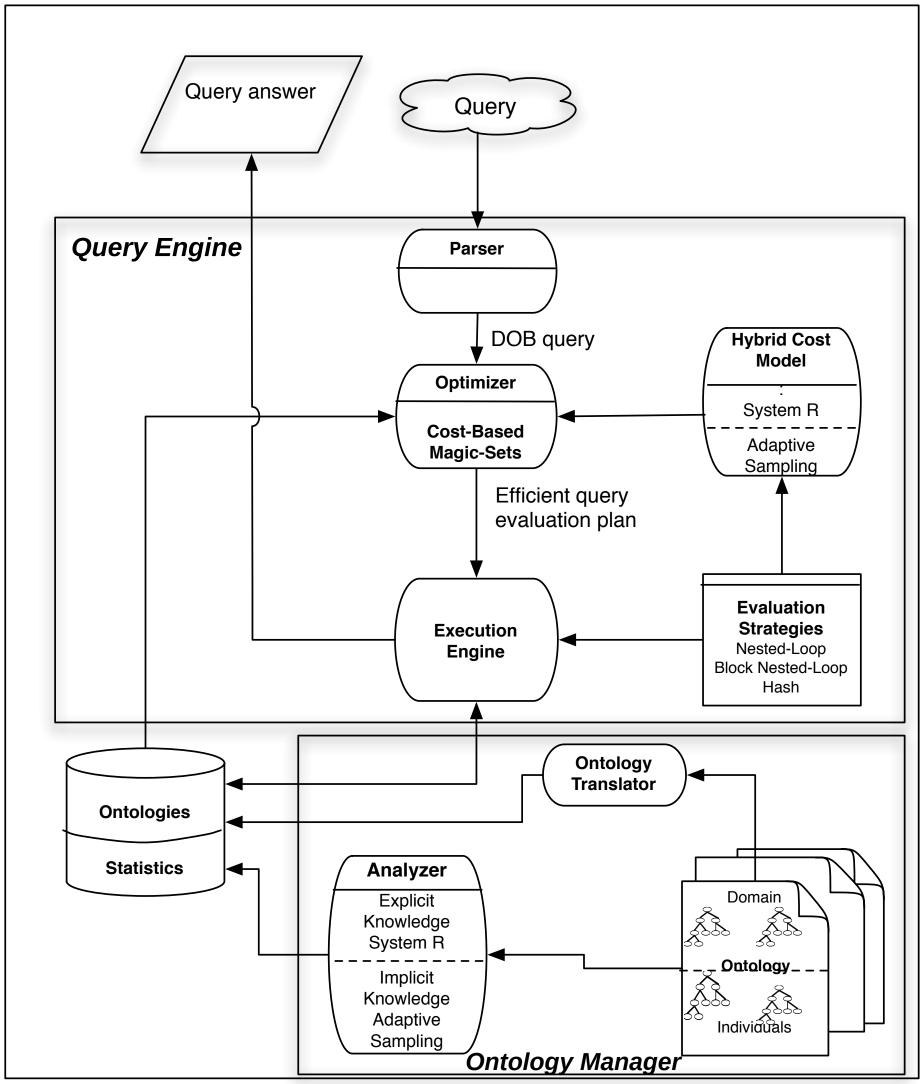


Fig. 1. The OnEQL Architecture

EOB PREDICATE	**DESCRIPTION**
`isOntology(O)`	An ontology has an Uri `O`
`isImpOntology(O1,O2)`	Ontology `O1` imports ontology `O2`
`isClass(C,O)`	`C` is a class in ontology `O`
`isOProperty(P,D,R)`	`P` is an object property with domain `D` and range `R`
`isDProperty(P,D)`	`P` is a datatype property with domain `D`
`isTransitive(P)`	`P` is a transitive property
`subClassOf(C1,C2)`	`C1` is a subclass of `C2`
`AllValuesFrom(C,P,D)`	`C` has property `P` with all values in `D`
`isIndividual(I,C)`	`I` is an individual belonging to class `C`
`isStatement(I,P,J)`	`I` is an individual that has property `P` with value `J`
IOB PREDICATE	**DESCRIPTION**
`areSubClasses(C1,C2)`	`C1` are the direct and indirect subclasses of C2
`areImpOntologies(O1,O2)`	`O1` import the ontologies `O2` directly and indirectly
`areClasses(C,O)`	`C` are all the classes of an ontology and its imported ontologies `O`
`areIndividuals(I,C)`	`I` are the individuals of a class and all of its direct and indirect superclasses `C`; or `I` are the individuals that participate in a property and belong to its domain or range `C`, or are values of a property with all values in `C`

Table 1. Some built-in EOB and IOB Predicates for a subset of OWL Lite

models. Conversely, to estimate the cost and cardinality of data that do not exist a priori, which is the case of the implicit facts, sampling techniques are applied. In our cost model, evaluation cost is measured in terms of the number of intermediate inferred predicates, and the cardinality corresponds to the number of valid instantiations of the predicate. This model estimates the cost and cardinality of explicit and implicit facts, as follows :

– To estimate the cardinality and cost of the intensional predicates that represent implicit facts, we have applied the Adaptive Sampling Technique [13]. This method does not need to extract, store or maintain information about the data that satisfy a particular predicate, and does not make any assumptions about statistical characteristics of the data, such as distribution. Sampling stop conditions are defined to ensure that the estimates are within an appropriate confidence level.
– To estimate the cardinality and cost of the extensional predicates, and the cost of a query plan, we use a cost model *à* la System R [23]. Similarly to System R, we store information about the number of ground facts corresponding to an extensional predicate, and the number of different values (constants) of each predicate variable. Regarding queries, the formulas for computing the cost and cardinality are similar to the different physical *join* formulas in relational queries.

Once a query is received by OnEQL, the parser checks if it is correct. If so, the query is translated into the OnEQL canonical form: the patterns in the *WHERE* clause of a SPARQL query are translated to a conjunctive query, where each pattern corresponds to an EOB or IOB predicate, and the join or conjunction between two predicates represents the '.' (AND) SPARQL operator.

The DOB query is then passed to the optimizer. The optimizer implements a twofold optimization technique and uses the statistics stored in the catalog to identify an efficient query execution plan. Next, the plan is given to the query engine which evaluates it against the ontology. Facts that satisfy the conditions expressed in the query are returned to the user.

The twofold optimization technique combines cost-based optimization and Magic Sets techniques. In the first stage, the cost-based optimization technique extends the System R dynamic-programming algorithm by identifying orderings of the EOB and IOB predicates in a query. During each iteration of the algorithm, the best intermediate sub-plans are chosen based on the cost and the cardinality that were estimated using our hybrid cost model. In the last iteration of the algorithm, final plans are constructed and the best plan is selected in terms of the estimated cost. This optimal ordering reflects the minimization of the number of intermediate inferred facts using a top-down evaluation strategy. For more details refer to [21].

In the second stage, OnEQL applies Magic Set optimization techniques [19] to the execution plan obtained in the first stage. Magic Sets combines the benefits of both, top-down and bottom-up evaluation strategies and tries to avoid repeated computations of the same subgoals, and unnecessary inferences. The DOB program is rewritten w.r.t. the optimal execution plan, and then evaluated with a bottom-up strategy. "Magic predicates" are inserted into the program to represent bounded arguments in the query, and "Supplementary predicates" are included to represent sideways information-passing in rules. It should be noted that we implemented the general Magic Sets technique for Datalog with the two improvements suggested by [1] to eliminate the first and last redundant supplementary predicates, and to merge consecutive sequences of EOB predicates in rule bodies.

Finally, three different physical operators or evaluation strategies can be used by OnEQL to implement the sideways passing of bindings between two predicates in an execution plan: nested-loop join, block nested-loop join and hash join [18]. The nested-loop join corresponds to a top-down Datalog evaluation strategy where the join variables in the second predicate are instantiated through the sideways passing of information. In the worst case, all of the predicate will be searched; however, more efficient search options may index the predicates by one or more of their arguments. The hash join strategy takes into account the availability of a hash function; it limits the number of pairs of predicate instantiations that need to be compared; nevertheless, it is restricted by the amount of main memory available. These algorithms were developed in the same spirit of relational join operator algorithms; accordingly, relational cost formulas have been modified to reflect the behavior of our operators and to measure the number of intermediate inferred facts; also, implementation details like the use of main memory and pipelining, and the availability of physical structures were represented in these formulas [21]:

- Nested-Loop Join
  For each valid instantiation in the first predicate, we retrieve the matching instantiations in the second predicate, i.e., the join arguments[2] are instantiated in the second predicate through the sideways passing of bindings.

[2] The join arguments are the common variables in the two predicates.

– Block Nested-Loop Join
  The first predicate is evaluated into blocks of fixed size, and then each block is joined with the second predicate.
– Hash Join
  A direct access table is built for the first predicate according to its join argument values. The valid instantiations of both predicates with the same key are joined.

We illustrate the functionality of OnEQL with the following example. In Fig. 2, we present a portion of the Galen ontology expressed in OWL and visualized using the OnEQL interface. A portion of the Galen translated DOB ontology can be seen in Table 2.

DOB predicate
isClass('factkb:Abdomen','Ontologies:galen.owl')
isClass('factkb:AbdominalAorta','Ontologies:galen.owl')
isFunctional('factkb:hasAbnormalityStatus')
isProperty('factkb:actsOn','Ontologies:galen.owl')
isTransitive('factkb:hasCause')
someValuesFrom('factkb:AdhesivePericarditis','factkb:hasOutcome','factkb:Adhesion')
subClassOf('factkb:AdductorMagnus','factkb:NAMEDMuscle')
subClassOf('factkb:AdductorTubercle','factkb:Eminence')
subPropertyOf('factkb:hasLayer','factkb:StructuralPartitiveAttribute')
subPropertyOf('factkb:hasLeftRightSelector','factkb:hasPositionalSelector')

Table 2. Portion of Galen DOB Ontology

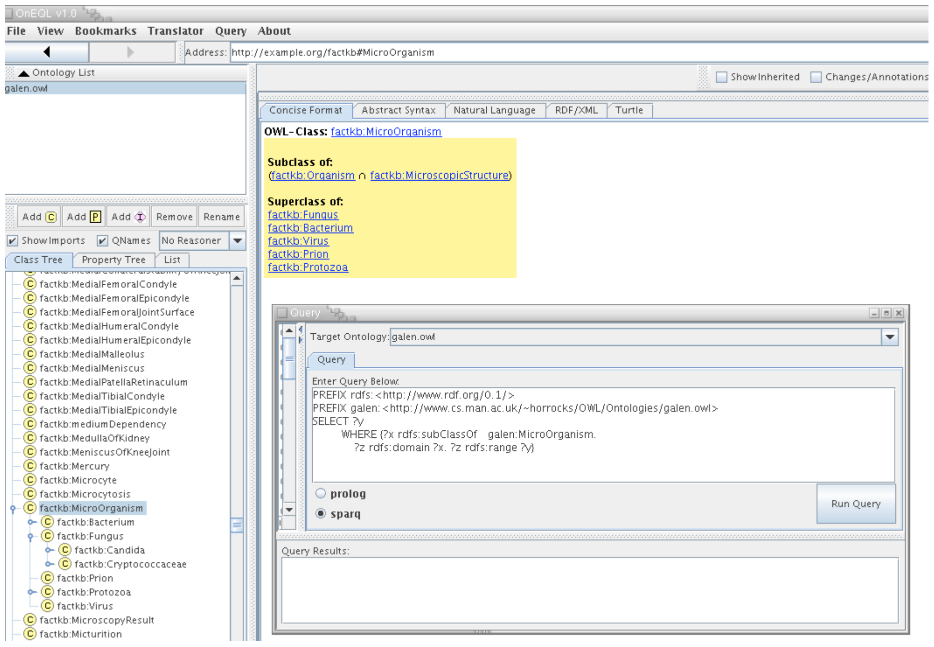
Consider the simple query: "Name all the drugs that act on the pathologies caused by the Helicobacter Pylori bacteria". The SPARQL representation of this query is as follows:

```
PREFIX rdfs:<http://www.rdf.org/0.1/>
PREFIX galen:<http://example.org/factkb#>
SELECT ?y
WHERE  {galen:actsOn rdfs:domain ?y.
        galen:actsOn rdfs:range ?x.
        galen:isCauseOf rdfs:domain galen:HelicobacterPylori.
        galen:isCauseOf rdfs:range ?x}
```

This example can be expressed as the following DOB query:
$q(Y) \leftarrow isDomProperty('actsOn', Y), isRanProperty('actsOn', X),$
$isDomProperty('isCauseOf', 'HelycobacterPylori'),$
$isRanProperty('isCauseOf', X).$

The `WHERE` clause is comprised of four triple patterns: the first and second patterns denote the relationship between a drug and a pathology, and the third and the fourth patterns represent the relationship between the pathologies caused by the Helicobacter Pylori bacteria. The result of the query evaluation is a set of solutions, i.e., the matchings of the query patterns and the RDF data.

Considering the triples encoded in Galen, and without taking into account any optimization technique, the evaluation of this simple query will require



Address: http://example.org/factkb#MicroOrganism

galen.owl

[✓] Show Imports [✓] QNames No Reasoner

[ ] Show Inherited [ ] Changes/Annotations

- factkb:MedialFemoralCondyle
- factkb:MedialFemoralEpicondyle
- factkb:MedialFemoralJointSurface
- factkb:MedialHumeralCondyle
- factkb:MedialHumeralEpicondyle
- factkb:MedialMalleolus
- factkb:MedialMeniscus
- factkb:MedialPatellaRetinaculum
- factkb:MedialTibialCondyle
- factkb:MedialTibialEpicondyle
- factkb:mediumDependency
- factkb:MedullaOfKidney
- factkb:MeniscusOfKneeJoint
- factkb:Mercury
- factkb:Microcyte
- factkb:Microcytosis
- factkb:MicroOrganism
  - factkb:Bacterium
  - factkb:Fungus
    - factkb:Candida
    - factkb:Cryptococcaceae
  - factkb:Prion
  - factkb:Protozoa
  - factkb:Virus
- factkb:MicroscopyResult
- factkb:Micturition

OWL-Class: factkb:MicroOrganism

Subclass of:
(factkb:Organism ∩ factkb:MicroscopicStructure)

Superclass of:
factkb:Fungus
factkb:Bacterium
factkb:Virus
factkb:Prion
factkb:Protozoa

Target Ontology: galen.owl

Enter Query Below:

```
PREFIX rdfs: <http://www.rdf.org/0.1/>
PREFIX galen: <http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl>
SELECT ?y
      WHERE {?x rdfs:subClassOf   galen:MicroOrganism.
         ?z rdfs:domain ?x. ?z rdfs:range ?y}
```

prolog [✓ sparq]

Query Results:

Fig. 2. Galen in OnEQL

727,547 intermediate inferred facts. In this naive plan, first all the combinations of drugs and pathologies are considered, and then the pathologies caused by the Helicobacter Pylori are selected. To reduce the number of intermediate computed facts, a cost-based optimization technique estimates the cost of the different orderings of the evaluation of the query, and recognizes a better way to evaluate the query. Therefore, it produces an execution plan where first, the different pathologies caused by the Helicobacter Pylori are selected; following this, the drugs that act on these pathologies are projected out. On the one hand, there are seventeen instances of the relationship between drugs and pathologies that require 726,980 inferences to be produced; on the other hand, the Helicobacter Pylori is only related to five pathologies and 72,743 inferences are needed to project out these pathologies. Thus, this new execution plan is less expensive in time and in the number of intermediate inferred facts, 291,371.

The optimal DOB query ordering follows:

$$\begin{aligned} q(Y) \leftarrow\ & isDomProperty('isCauseOf',' HelycobacterPylori'),\\ & isRanProperty('isCauseOf', X),\\ & isDomProperty('actsOn', Y), isRanProperty('actsOn', X). \end{aligned}$$

Once an optimal query ordering has been selected, query bindings can be used to rewrite the canonical representation of the ontology and simulate the pushing of selections that occurs in a top-down evaluation strategy. In this example, the query has nine bindings, e.g., "rdfs:domain" and "galen:actsOn" in the first pattern. Besides rewriting the program with supplementary and Magic predicates, the query is also rewritten to include the "seed" that represents the variable bindings. The rewritten program and query are then evaluated using a semi-naive bottom-up evaluation strategy. The Magic Sets rewritten DOB of the optimal ordering required 111,071 intermediate inferred facts during the bottom-up evaluation, while the top-down evaluation of this ordering required 291,371 inferred facts.

3 Experimental Results

In this section we report the behavior of the OnEQL query techniques in ontologies commonly used in the health domain. We consider the ontologies Galen [6] and EHR_RM [5].

The Galen ontology is a repository of medical terms and procedures. It provides a set of modeling conventions and patterns that have proved sufficiently robust to be applied in practical developments such as surgical terminologies, drug information and data entry systems. Particularly, it has been used for the development of the French national classification of surgical procedures CCAM [20] and for the development of the drugs ontology in the UK [24].

The EHR_RM ontology is a controlled vocabulary for electronic health records that maintains all the information required to facilitate the flow needed for patient care. EHR_RM is comprised of two levels: level one corresponds to a set of classes and relationships that represent properties in the whole world; level two is composed of a set of clinical concepts which are related to the general concepts in level one.

In Table 3, these ontologies are described in terms of the number of classes, the average properties associated with a class, the maximal fan out, the height of the ontology, and the number of parents. We can observe that Galen is a hierarchy of concepts where each class can have a large number of sub-classes; almost no relationships or properties are associated with each class. EHR_RM is simpler and there are some relationships and properties related to a class. These characteristics impact on the evaluation cost of queries that require recursive traversals of the data.

Ontology	#Classes	Fan out	Height	# Parents
Galen	2749	18	2	13
EHR_RM	187	2	2	7

Table 3. Ontology descriptions

We conducted an experimental study to analyze the behavior of our query techniques on synthetic ontologies and the above-mentioned real-world ontologies. A synthetic ontology document was generated with ten related ontologies and a total of 4350 basic facts. Each ontology has between twenty to thirty classes, around twenty relationships, three to five attributes for each class, and around sixty sub-class relationships. All the numbers described above were randomly chosen following a uniform distribution. Additionally, we randomly generated sixteen chain queries[3] for each ontology. Experiments were executed on a Sun Fire V440 equipped with two UltraSPARC IIIi processors running at 1.593 GHZ with 16 GB RAM. The OnEQL system was implemented in Java 1.4 and SWI-Prolog 5.6.1. In this paper we report the predictive capability of the cost model, and cost improvements from using the twofold optimization strategy.

First, we report the correlation between the estimated cost of the top-down evaluation of 384[4] orderings, and the actual cost of evaluating the Magic Sets rewritings w.r.t. these orderings using a bottom-up strategy. The idea is to measure if the estimated cost of the top-down ordering is correlated to the actual cost of applying Magic Sets to this ordering, i.e., Magic Sets emulates a top-down evaluation strategy but tries to avoid repeated computations of the same subgoals. The correlation for the real-world ontology Galen is 0.53, while for EHR_RM it is 0.43.

Additionally, we studied the benefits of the twofold optimization strategy. For each query we applied Magic Sets to all its orderings, and we compared:

- The percentile of the Magic Sets optimal ordering actual cost, i.e., the cost of applying Magic Sets to the optimal ordering. For the three ontologies we can observe that the cost of the Magic Sets optimal ordering falls in at least the 74th percentile, indicating that three quarters of all the orderings are worse than this cost (Table 4).
- The average ratio of the cost of the Magic Sets optimal ordering to the worst cost (resp. median cost), i.e., the number of times the worst cost (resp. median cost) contains the optimal cost; it is expressed as a percentage (Figures 3 and 4).

Ontology	Percentile
Synthetic	74th
Galen	77th
EHR_RM	75th

Table 4. Percentile of the cost of the optimal ordering

For the synthetic ontologies, the average of the ratio of the optimal cost with respect to the worst-case is 45%; Galen and EHR_RM have averages of 15% and 32%, respectively. The averages of the ratio of the optimal cost with respect to the median are 74%, 63% and 81% for synthetic, Galen and EHR_RM ontologies respectively. From these results we can conclude that the cost of an optimal ordering is always better than the median cost, while the optimal cost

[3] Queries where bindings are propagated from left to right in a chain-like fashion

[4] Each of the sixteen queries has four sub-goals, $16 \times 4! = 384$

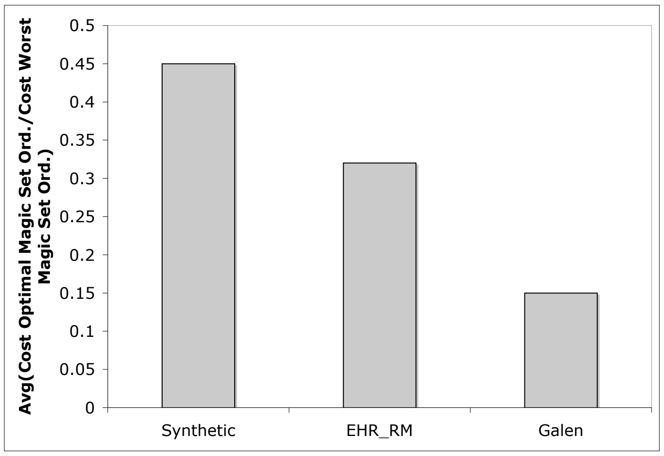


Fig. 3. Average ratio of the cost of the Magic Sets optimal ordering to the worst cost

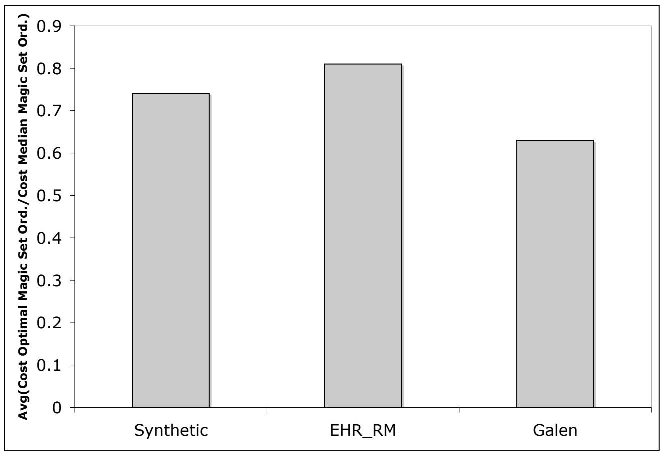


Fig. 4. Average ratio of the cost of the Magic Sets optimal ordering to the median cost

corresponds to a small fraction of the worst cost. To explain these results, recall that classes in Galen are more connected than classes in EHR_RM and, in consequence, the same fact may be generated several times during the computation of the transitive closure of Galen's subsumption relationship. Therefore, our proposed optimization techniques may have a better chance of causing an impact on queries against Galen (by minimizing the number of intermediate and duplicated inferred facts), than on queries against simpler ontologies like EHR_RM.

4 Related Work

Efficient query evaluation techniques against ontologies have been proposed in [3, 7–10, 16, 21, 22]. In [3, 8, 9], relational query techniques and Description Logics reasoning services have been combined to efficiently solve querying and reasoning tasks over individuals of an ontology stored in a database. These systems are built upon relational DBMS and they do not develop optimization techniques that use the semantics encoded in the ontology to identify good evaluation plans.

In [22], ontology segmentation techniques are proposed to approach the problem of querying large ontologies such as Galen. These techniques exploit the semantic connections between ontology terms to enable users to create new sub-ontologies with the portion of the original ontology that is relevant to the application or query. On the other hand, the projects described in [10, 16] developed Magic Sets query rewriting techniques to generate new programs that evaluate the input query more efficiently. In none of these two techniques cost or cardinality estimations are considered, and the new portion of the ontology or the evaluation strategy may be inefficient depending on the shape of the ontology.

5 Conclusions and Future Work

In this paper we have described OnEQL, a tool that evaluates SPARQL queries against OWL Lite ontologies. We implemented these two standards because they achieve good trade-offs between expressiveness and computational tractability.

To enhance the performance of the reasoning and querying tasks, we propose a twofold optimizer which combines the benefits of cost-based and Magic Sets approaches. Additionally, a hybrid cost model is implemented. This cost model integrates estimation techniques used in traditional relational DBMSs [18, 23] with adaptive sampling to estimate the cost or cardinality of explicit and implicit classes [13]; the cost model allows the precise estimation of these metrics.

In our experiments we observed that correlations between estimated and actual values are not greater than 0.53. From these values, we can conclude that our cost model overestimates the cost if the top-down evaluation produces a large number of repeated inferences, because the actual cost of applying Magic Sets emulates a top-down evaluation without repeated inferences.

Also, the implemented optimization techniques allow the identification of optimal query plans whose cost is less than 45% of the cost of the worst plan.

In the future, we plan to conduct experiments on other large ontologies, and to define cost metrics that provide a better estimate of the behavior of the Magic Sets technique.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.
2. D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004.
3. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tailoring OWL for data intensive ontologies. In *Proc. of the Workshop on OWL: Experiences and Directions*, 2005.
4. Disease Ontology. http://diseaseontology.sourceforge.net.
5. EHRRM Ontology. http://trajano.us.es/ isabel/EHR/EHRRM.owl.
6. GALEN Common Reference Model. http://www.openclinical.org/dld_galenCRM.html.
7. B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the WWW2003: World Wide Web Conference*, 2003.
8. I. Haarslev and R. Moller. Optimization techniques for retrieving resources described in OWL/RDF documents, First results. In *Proc. of KR2004: International Conference on the Principles of Knowledge Representation and Reasoning*, 2004.
9. I. Horrocks and D. Turi. The OWL Instance Store: System description. In *Proc. of CADE2005: International Conference on Automated Deduction*, 2005.
10. U. Hustadt and B. Motik. Description Logics and Disjunctive Datalog The Story so Far. In *Proc. of DL 2005 - International Workshop on Description Logics*, 2005.
11. A. Kalyanpur and E. Sirin. SWOOP - A Hypermedia-based Featherweight OWL Ontology Editor. http://www.mindswap.org/2004/SWOOP/, 2004.
12. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *Proc. of the WWW2002: World Wide Web Conference*, 2002.
13. R. Lipton and J. Naughton. Query size estimation by adaptive sampling (extended abstract). In *Proc of SIGMOD1990: Special Interest Group on Management of Data Conference*, 1990.
14. D. McGuinness and F. van Harmelen. OWL Web Ontology language overview. *W3C Recommendation*, 2004.
15. Medical Subject Heading (MeSH). http://www.nlm.nih/gov/mesh.
16. B. Motik, R. Volz, and A. Maedche. Optimizing Query Answering in Description Logics using Disjunctive Deductive Databases. In *Proc. of the KRDB2003: International Workshop on Knowledge Representation meets Databases*, 2003.
17. E. Prudhommeaux and A. Seaborne. SPARQL Query Language for RDF. In *http://www.w3.org/TR/rdf-sparql-query*, 2006.
18. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. Mc Graw Hill, 2003.
19. R. Ramakrishnan and J. D. Ullman. A Survey of Research on Deductive Database Systems. *Journal of Logic Programming*, 23(2):125–149, 1993.
20. J. Rodrigues, B. Trombert-Paviot, R. Baud, J. Wagner, P. Rusch, and F. Meusnier. Galen-In-Use: an EU Project applied to the development of a new national coding system for surgical procedures: NCAM. In *Medical Informatics Europe*, 1997.

21. E. Ruckhaus, E. Ruiz, and M. Vidal. Query Evaluation and Optimization in the Semantic Web. In *Proc. of ALPSWS2006: International Workshop on Applications of Logic Programming to the Semantic Web and Semantic Web Services*, 2006.
22. J. Seidenberg and A. Rector. Web Ontology Segmentation Analysis, Classification and Use. In *Proc. of WWW2006: World Wide Web Conference*, 2006.
23. P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. *Proc. of SIGMOD1979: Special Interest Group on Management of Data Conference*, 1979.
24. M. Stearns. SNOMED clinical terms: overview of the development process and project status. In *Proc. of AMIA2001: American Medical Informatics Association (AMIA) Symposium*, 2001.
25. XML Schema. http://www.www.w3.org/XMLSchema.