Using Domain Ontologies for Efficient Information Retrieval

Sandhya Revuri, Sujatha R Upadhyaya and P Sreenivasa Kumar

Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai - 600036, India
{sandhya, sujatha, psk}@cs.iitm.ernet.in

Abstract

Being the conceptual models that capture domain knowledge, ontologies can be looked upon for aiding meaningful information retrieval. This paper is an effort to improve the relevancy of results in a search system for a domain by exploiting the domain knowledge captured in an OWL DL Ontology. We propose a system that fits the query terms in the ontology graph in an appropriate way and exploits the surrounding knowledge to derive an enhanced query. The enhanced query is given to the underlying basic keyword search system. The results thus obtained are ranked using our ranking algorithm. To the best of our knowledge, ours is the first approach that tries to make use of more ontological knowledge than IS-A relationships and synonyms for information retrieval. As a result, we find that we can achieve substantial improvement in both precision and recall compared to the basic keyword search system.

1 Introduction

Today's search engines have immensely benefited from sophisticated information retrieval techniques. However, reducing the false positive information and adding false negative information could greatly improve the semantic relevance of information retrieved. In the literature, we see many efforts of using ontologies for query expansion. Some of these consider domain knowledge captured in UMLS as ontology [6], [9], some consider word-net as a linguistic ontology, some use the domain ontologies built as part of their own effort [8]. Most efforts have focused on finding synonymy between terms, some of them exploit the IS-A relationship [4] between terms. Recently, Wollersheim et al [9] have employed a method of exploiting meronymy and holonymy relationships between terms. In our effort, we focus on deriving the relationships between the keywords and use them for query expansion, besides synonymy and IS-A relationship.

One of the ways to represent domain ontologies in a well structured and expressive manner is OWL [3]. The vocabulary offered by OWL is rich enough to capture the relations in the real world. Using an ontology to bring into context, terms that are relevant to the search string and expand the search string with them is the central theme of this paper.

In this paper, we propose and discuss the details of an IR system that works on domain specific documents and exploits the domain ontology for improving the precision and recall. We limit our discussion to search domain specific documents.

We assume that every keyword in the search string corresponds to some word or can be syntactically transformed into some word(s) of the vocabulary provided by the ontology.

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006
©*Computer Society of India, 2006*

2 Ontology Design

For experimental verification, we have chosen the domain "Data Structures and Algorithms" and built an OWL DL [2] ontology for it. This ontology describes the different types of data structures available, their characteristics, the contexts in which they are useful, the well known problems in computer science, the algorithms corresponding to them. We have given only a brief overview of our ontology in the subsequent section.

For building an ontology, first we list all the possible concepts in the domain. Next step involves identifying the properties of each of these concepts. Once the properties are identified the domain and range are defined. The next step would be identifying the characteristics of the properties like Transitive, Symmetric, Functional and Inverse Functional. We can pose value or cardinality constraints on properties to make the concept description more specific. Value constraints are of the form *allValuesFrom*, *someValuesFrom* and *hasValue*. Cardinality constraints are *minCardinality*, *maxCardinality* and *cardinality*. In the next step the relationships between the modeled concepts are identified, which include *IS-A*, *disjointness*, *equivalence*. Once the TBox is defined, data adhering to the TBox needs to be populated. Defining instances need the identification of a proper class, then fill in the property values. All those entities which satisfy the constraints of a particular concept are eligible to be individuals of that concept. Our ontology is checked for consistency using the reasoner Racer [5].

3 System specification

Our system consists of four modules. 1. Query Interface 2. Query Expansion Module 3. Search module 4. Ranking module. The query interface, in our system is a simple google like interface, allows the user to input a query with a maximum of two terms from the ontology. The Query expansion module expands the user query into a semantically well-defined query using the Query expansion algorithm. The expanded query is given to the basic keyword search engine. The results thus obtained are sorted using the ranking module. Finally the ranked results are presented to the user interface. The Query expansion algorithm and the ranking algorithm are explained in subsequent sections.
We use boolean operators *.and.*, *.or.* to glue the components of the expanded query. The precedence of *.and.* is higher than *.or.*. Default operator assumed is *.or.* The associative, distributive, identity laws hold here also.

3.1 Algorithm for Query Expansion

Whenever a user given query maps to a single term in the ontology, three possible cases may arise.
Case 1: (Concept) When the query gets mapped to a *concept* in the ontology, the query is expanded in terms of its *def(Q)* **which captures the necessary and sufficient conditions for describing the query term Q**, *inferred properties*, *inferred subclasses* and *inferred instances*. The expansion in terms of concept *def* guarantees that the semantically relevant documents are retrieved despite not containing the query term. For example *Q(graph)* should also retrieve the documents which speak of *data structure with finite vertices and edges* even in the absence of the query term *graph*. Expanding the concept in terms of other inferred knowledge (i.e, properties, subclasses, instances) ensures that the documents are retrieved in the right context. Following the above example the expanded query now retrieves the documents which discusses the properties of graph like *edges, vertices, cycles, etc* or subclasses of graphs like *Directed graph, undirected graph,..* or instances of graph like *Simple graph, Multigraph*.
Case 2: (Property) If the query maps to a property, the query is expanded in terms of its *domain* and *r*ange. For a query Q(edge), the expanded query will be in the form of *(Graph .and. edge) .or. (Tree .and. edge) .or. (edge .and. directed) .or. (edge .and. undirected)* which ensures that edge is discussed in the intended context.
Case 3: (Instance) A very specific case of a concept in which the expanded query include all those instances which are strongly related to the given instance. In terms of OWL, the given instance is expanded in terms of its property values if the property has characteristic inverse functional. For example, Q(Minimum Spanning Tree) should also give us kruskal's and Prim's algorithm.

Whenever the query maps to two terms in the ontology, relationship between them is identified and exploited accordingly.
Case 4: (Concept, Concept) The possible relationships that hold between concepts are *Sibling-of*, *IS-A*, *disjoint-with*. In case the two given concepts have a common ancestor, the query is expanded in terms of specific properties possessed by each of the concepts along with the common ancestor with its properties. For example in the tourism domain the query *Conference Room, Guest Room* map to the concepts *Conference Room, Guest Room* with the parent concept *Room*. We retrieve documents that discuss these concepts or their parent concept in the context of the common properties such as *Name, Internet-Access, Telephone, TV* or the specific properties of either of the query concepts. The specific properties of the concept *Guest Room* are *Minibar, Terrace, Balcony, Bed..* whereas *Projector, Stage, Video Conference system, Screen,..* are the specific properties of the concept *Conference Room*. We leave out the documents that do not contain none of the common or specific property terms.
When the given concepts do not have a common ancestor, the query is expanded in terms of the intermediate concepts and the connecting properties.
Case 5: (Concept, Property) A specific case of *Property* case where either the domain or range is provided by the user. If the given concept is a domain(range), the expanded query includes the range(domain) of the property. The domain and range are further expanded in terms of their inferred subclasses and instances. For example *Q(Queue-Applications)* bring out the results containing the sets *Queue .and. Applications .and. Schedulers*, *Queue .and. Applications .and. Simulation*. Then the query pull out those containing the instances of the Queue along with the property, *Priority-Queue .and. Applications*, *Dequeue .and. Applications*.
Case 6: (Property, Property) Given two property terms, we proceed with identifying the concept(s) on which these given properties are defined (in the place of domain or range). Whenever the given properties identify a common concept the query is expanded in terms of the common concept along with these prop-

erties. The search string *Audio Equipment, Video Conference-System* corresponds to two properties defined on the concept *Conference Room* in the tourism domain. Once the concept is identified, we discuss these properties in the context of *Conference Room.* So the enhanced search string is {Conference Room .or. (Audio Equipment .and. Video Conference-System)}. We do not consider the range of the given properties because they are datatype properties.

Case 7: (Concept, Instance) This is a specific case of *Concept-Concept* where we have one of the concepts referring to a specific instance of a class. Just as in the mentioned case, the given instance could be an instance of given class , or could be an instance of a sibling of class C, where the siblings could be either overlapping or disjoint.

Case 8: (Instance, Instance) If each of the keywords is treated as an individual of a concept, it would also be the one specific case of Case 4 where the relationships between the instances are taken into consideration. Accordingly, the common properties, the specific properties (properties of instances in general) and the concept to which they belong (in case they belong to a common concept) are brought into context. It is possible that these two instances are related through a set of (object property, value) pairs, in which case all the connecting (object property, value) pairs are brought into context.

Case 9: (Property, Instance) If the given property holds on the given instance then the query is expanded in terms of the values of the property for that individual. A query for *priority-queue Applications*, basically looks for the values of the property *Applications* in the context of *priority queue*, which in this case is *Heap-Construction.*

If the terms in the query are not related by any of the relationships discussed above, the keyword based search is performed which can be treated as default case.

3.2 Effect of Property Constraints on Query Expansion

Although, we have not specifically mentioned, in the above cases, if a property appears with a constraint, we exploit it to improve the search. For instance the query expansion for Q_{def} *(digraph)* as *graph .and. ((each .or. all .or. every) .and. edge) .and. directed.* Here, the terms *(each .or. all .or. every)* are conjuncted with edge to bring the effect of *allValuesFrom directed* constraint on edge. To indicate that every edge in a digraph has to be a directed edge, we use words such as all, every, each of English language. So we bring such words into context. Similarly in the context of a *someValuesFrom* constraint, we use words such as *atleast-one, one, minimum-one.* We handle all the constraints (value and cardinality) in a similar manner.

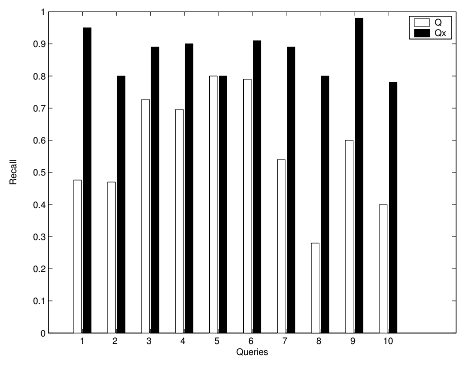

Figure 1: Queries Vs Recall

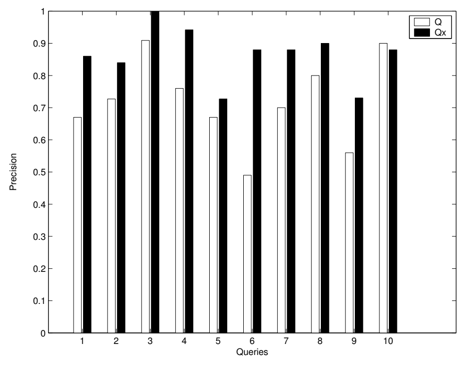

Figure 2: Queries Vs Precision

3.3 Algorithm for Ranking

The popular ranking algorithms take into consideration mostly the frequency of occurrence of query terms in the documents. But, we rather take into consideration the semantic relevance as decided by our query expansion. While expanding the query, we make an effort to bring out the related properties, concepts, sub concepts and individuals relevant to the query. In general, we order those documents that contain the terms whose semantic distance with the terms in the search string is small. Semantic distance of a term refers to the numbers of levels it is away from the search term in the Ontology. Once the documents are grouped by this criterion, we rank the documents within the group by checking the number of keywords present in the document. The documents containing most of the keywords in the expanded query are ranked higher. The last criterion for ranking is the proximity of the relevant terms in the document. The closer the words in the document higher will be the priority. This is very important when we are looking for query expansion of *def(S)*, for example.

Table 1: Test Queries and their expansions

Label	Type	Query(Q)	Enhanced Query(Q_x)
Q1	Property	Traversal	{((Graph .or. Tree .or. Binary-Tree .or. AVL-Tree) .and. Traversal) .or. (Breadth-First-search .or. Depth First Search .or. Inorder Traversal .or. Preorder Traversal .or. Post order Traversal)}
Q2	Instance	Dijkstra's Algorithm	{(Dijkstra's Algorithm .and. (complexity .or. running time .or. algorithm)) .or. Single source shortest path problem}
Q3	Instance-Instance	Inorder-Traversal Preorder-Traversal	{Inorder-Traversal .or. Preorder-Traversal .or. Tree-Traversal}
Q4	Concept	Graph	{(ADT .and. finite .and. non-empty .and. vertices .and. edges) .or. (Graph and. ((edge .and. node) .or. (Applications .or. Representation .or. Implementations .or. Operations) .or. (Acyclic graph .or. Cyclic graph .or. Directed graph .or. Digraph .or. Undirected graph .or. tree)))}
Q5	Concept-Concept	Traveling salesman Problem Complexity	{(Traveling salesman .and. (algorithm .or. problem .or. complexity)}

4 Experimentation and Results

For testing our system we need a well defined ontology. We use the "Data Structures and Algorithms" ontology as discussed in the ontology design section. We used Protege-3.1 [1] for building the Ontology and Racer-1.73 [5] for checking the consistency of the Ontology. The implementation of the system is done in Jena-2.3 [7]. Our corpus consists of chapters and paragraphs from Data Structures text books, stored in Oracle 9i database. We used Oracle 9i's Oracle Text as a keyword based search system. We have tested our system with ten queries. Only few queries with their expanded versions are tabulated in Table 1 due to space limitations. The relevancy of the retrieved results are evaluated manually. It is noticed that the average increase in recall with query expansion is found to be 34% as shown in Figure 1. The average increase in precision is found to be 23% and shown in Figure 2.

5 Conclusion

The proposed system is an effort to retrieve relevant documents in a domain using domain specific knowledge captured in the form of OWL ontology. It can be classified as an automatic query expansion mechanism using an ontology, with the underlying keyword search engine. Experimental results show that the system retrieves the documents which would have been missed and avoids retrieving documents which are irrelevant despite the presence of the keyword. It thus improves the precision and recall substantially.

References

[1] Protege: An Ontology Editor. Technical report, August 2004. available at http://protege.stanford.edu.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Descrption Logic HandBook, Theory Implementation and Application.* Cambridge University Press, Seattle, Washington, USA, 2003.

[3] S. Bechhofer, F. V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference*, February 2004. available at http://www.w3.org/TR/owl-ref/.

[4] D. Bonino, F. Corno, L. Farinetti, and A. Bosca. Ontology driven semantic search. *WSEAS transactions on Information Science and Applications*, 1(6):1597–1605, 12 2004.

[5] V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. In Y. Sure and O. Corcho, editors, *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, volume 87 of *CEUR Workshop*, Montreal, Canada, 2003.

[6] W. Hersh, S. Price, and L. Donohoe. Assessing thesaurus-based query expansion using the umls metathesaurus. In *Proceedings of AMIA Annual Symp 2000*, pages 344–348, 2000.

[7] http://www.hpl.hp.com/semweb/javadoc/. Jena Framework. *Overview documentation for Jena*, June 2004.

[8] G. Nagypál. Improving information retrieval effectiveness by using domain knowledge stored in ontologies. In *OTM Workshops*, pages 780–789, 2005.

[9] D. Wollersheim and J. W. Rahayu. Ontology based query expansion framework for use in medical information systems. *IJWIS*, 1(2):101–115, 2005.