Computing and Informatics, Vol. 28, 2009, 795–809

ENHANCED SEARCH METHOD FOR ONTOLOGY CLASSIFICATION

Je-Min Kim, Soon-Hyen Kwon, Young-Tack Park

School of Computing, Soongsil University
1-1, Sangdo-dong, Dongjak-Gu, Seoul, 156-743, Korea
e-mail: kimjemins@hotmail.com, kwonshzzang@naver.com, park@ssu.ac.kr

Revised manuscript received 19 January 2009

Abstract. The web ontology language (OWL) has become a W3C recommendation to publish and share ontologies on the semantic web. In order to infer implicit information (classification, satisfiability and realization) of OWL ontology, a number of OWL reasoners have been introduced. Ontology classification is to compute a partial ordering or hierarchy of named concepts in the ontology using the subsumption testing. Most of the reasoners use both top-down and bottom-up searches using subsumption testing for ontology classification. As subsumption testing is costly, it is important to ensure that the classification process uses the smallest number of tests. In this paper, we propose an enhanced method of optimizing the ontology classification process of ontology reasoning. Our work focuses on two key aspects: The first and foremost, we describe classical methods for ontology classification. Next, we present description of the enhanced method of optimizing the ontology classification and the detailed algorithm. We evaluate the effect of the enhanced method on four different types of test ontology. The enhanced search method shows 30 % performance improvement as compared with the classical method according to the result of the experiment.

Keywords: Ontolgy, DL reasoning, tableaux algorithm, classification, subsumption test, top search, bottom search

1 INTRODUCTION

The web ontology language (OWL) has become a W3C recommendation to publish and share ontologies on the semantic web. Ontology is a formal, explicit specification

of a shared conceptualization of a domain of interest [1]. Therefore, many ontology building and reasoning systems can be used to represent taxonomic and conceptual knowledge of a problem domain in a structured and well formed way. These systems must also be able to reason about this knowledge. An important inference capability of an ontology reasoning system is classification. Ontology reasoners are used to classify concepts in ontology, that is to compute a partial ordering or hierarchy of named concepts in the ontology base on the subsumption testing. As subsumption testing is costly, it is important to ensure that the classification process uses the smallest number of tests. Algorithms based on traversal of the concept hierarchy can be used to minimize the number of tests required in order to add a new concept [2]. Most of the reasoners use both top-down and bottom-up searches using subsumption testing for ontology classification. The idea is to compute a concept's subsumers by searching down the hierarchy from the top node (top-down search) and its subsumees by searching up the hierarchy from the bottom node (bottom-up search).

Although this idea has a big advantage, it still does not exploit all possible information. Therefore we propose an enhanced method to optimize the ontology classification process of ontology reasoning. First, during the top-down search, we can take results of subsumption tests that have already been performed and the advantage of the transitivity of the subsumption relation by propagating failed results down the hierarchy or propagating successful results up the hierarchy. Second, in the bottom-up search, we can use the information gained during the top-down search as well. This technique is very effective when used on the kinds of legacy ontology as it avoids performing large numbers of subsumption tests. It also turns out to produce improvement in classification times for some more complex ontology. First and foremost, we aim to describe classical methods for ontology classification. Next, we present description of the enhanced method of optimizing the ontology classification and the detailed algorithm. We evaluate the effect of the enhanced method on four different types of test ontology.

Building the optimization method that came off best into ontology reasoning system greatly enhanced its efficiency. One goal of this paper is to provide such an available algorithm for future implementers of ontology reasoning system.

In the next section we will present the classical ontology classification technique, a description of the enhanced method of optimizing the ontology classification, the detailed algorithm, and the results of running the algorithm on the cases.

2 PRELIMINARIES

2.1 Description Logic

Description Logics (DL) [2] are a family of logic-based knowledge representation formalism. DLs are usually a (decidable) subset of First Order Predicate Logic (FOL), and thus have a well-defined, formal semantics.

The basic building blocks in DL are atomic concepts which correspond to 1-place (unary) predicates in FOL and denote a set or a class of objects, atomic roles

which correspond to 2-place (binary) predicates in FOL and denote relations between objects, individuals which correspond to constants in FOL. A DL provides a set of operators, called constructors, which allow the formation of complex concepts and roles from atomic ones. For example, by applying the concept conjunction constructor ($\sqcap$) on the atomic concepts Person and Male, the set of all "Male People" can be represented as follows: Person $\sqcap$ Male.

Description Logic knowledge bases (KB) typically consist of a TBox containing concept inclusion axioms of the form $C_1 \sqsubseteq C_2$ where both C_1, C_2 are concepts, an RBox containing role inclusion axioms of the form $R_1 \sqsubseteq R_2$ with R_1, R_2 roles, an ABox containing axioms of the form $C(a)$, called concept assertions and $R(a, b)$, called role assertions where a, b are object names. In its simplest form, a TBox consists of a restricted form of concept inclusion axioms called concept definitions: sentences of the form $A \sqsubseteq C$ or $A \equiv C$.

2.2 OWL and Ontology Reasoning

The Web Ontology Language (OWL) [3] is an integral component of the semantic web, as it can be used to write ontologies or formal vocabularies which form the basis for semantic web data markup and exchange. From a modeling and semantic point of view, OWL shares a strong correspondence with Description Logics borrowing many logical constructs. OWL comes in three increasingly expressive sub-languages or "species", OWL-Lite, OWL-DL and OWL-Full. Among these, OWL-DL corresponds to the description logic $SHOIN(D)$.

Reasoning services for OWL are typically the same as those for DLs, and include consistency check; whether an OWL ontology O is logically consistent, class subsumption (Ontology Classification), given a pair of classes C, D in the ontology O, check whether $O \models C \sqsubseteq D$ (also related is the notion of class satisfiability: $C \sqsubseteq \bot$ and class equivalence: C, D which implies $C \sqsubseteq D$ and $D \sqsubseteq C$), when given an individual a and a class C in the ontology O, check(instantiation) whether a is an instance of C.

3 ONTOLOGY CLASSIFICATION

Ontology classification is the process of establishing partial order on the set of named concepts in ontology using the subsumption tests. Beside answering specific subsumption and satisfiability queries, it is often useful to compute and store the subsumption relation of all the concept names in the ontology.

In practice, many ontology reasoners use subsumption test algorithm that are not capable of determining subsumption relations with respect to an arbitrary ontology. In the past years, sound and complete subsumption test algorithms for large concepts of ontology have been developed [4]. Most of these algorithms are designed based on satisfiable checking algorithms. These algorithms use model generation procedures, and are similar to first-order tableaux calculus. Since a concept A subsumes a concept B if, and only if, $\neg A \sqcap B$ is not satisfiable, i.e., there does not

exist an interpretation which interprets $\neg A \sqcap B$ as a non-empty set, a satisfiability algorithm in fact can be used to solve the subsumption problem. In order to check whether a given concept C is satisfiable, the tableaux-based algorithm tries to generate a finite interpretation in which C is interpreted as a non-empty set. This generation process is complete in the sense that if it fails, i.e., an obvious contradiction occurs, it can conclude that C is not satisfiable; otherwise C is satisfiable.

3.1 Definition Order

The number of subsumption tests required for classification can be affected by the order in which concepts are added to the taxonomy. A well-known optimization is to add ontology concepts in *definition order*. *Definition order* is the data structure that reveals obvious subsumption relations and controls the order in which concepts are added to the hierarchy.

Definition order is used in order to minimize the number of subsumption tests needed in classification. For example, when adding concept C to the hierarchy, a top-down search is used that only checks if D subsumes C when it has already been determined that C is subsumed by all the concepts in the hierarchy that subsume D. The structure of TBox axioms is also used to compute a set of *told subsumers* of C. *Told subsumers* are a set of trivially obvious subsumers. For example, if the TBox contains an axiom $C \sqsubseteq D_1 \sqcap D_2$, then both D_1 and D_2, as well as all their told subsumers, are told subsumers of C. As subsumption is told subsumers, no test need to be performed with respect to these concepts. The told subsumer optimization can be used to approximate the position of C in the hierarchy.

To maximize the effect of *told subsumer* optimization, concepts should be classified in *definition order*. This means that a concept A is not classified until all of its *told subsumers* are classified. Therefore, as concepts are classified *definition order*, a primitive concept will always be classified before any of the concepts that it subsumes. In this procedure, all concept names are sorted into *definition order*.

4 TOP-DOWN AND BOTTOM-UP SEARCHES

Traditionally, ontology hierarchy is built iteratively. The order is initialized with the trivial relation $\bot \prec \top$, and on every iteration one new concept name C is added. For each concept name C added to the taxonomy, the set of parents and the set of children are determined. These two sets uniquely identify the place of C in the current taxonomy. The set of parents is defined by a procedure called a top-down search, and the set of children is defined by a bottom-up search.

The top-down search starts at the top of the already computed hierarchy. For each concept $x \in X_i$ under consideration it determines whether x has an immediate successor y satisfying $c \preceq y$. If there are such successors, they are considered as well; otherwise, x is added to the result list of the top-down search.

In order to avoid multiple visits of elements of X_i and multiple comparisons of the same element with c, the top-down search algorithm described in Figure 1

```
top-search(c, x)
  mark(x, "visited")
  for all y with y ≺ x do
     if simple-top-subs?(y, c)
       then Pos-Succ ← Pos-Succ ∪ {y}
     fi
  od
  if Pos-Succ is empty then
     Result ← {x}
  else
    for all y ∈ Pos-Succ do
       if not marked?(y, "visited")
        Result ← Result ∪ top-search(c, y)
       fi
    od
  fi
```

```
simple-top-subs?(y, c)
if marked(y, "positive") then
  Result ← true
elseif marked(y, "negative")
  then
Result ← false
elseif subs?(y, c)
  then
      mark(y, "positive")
      Result ← true
else
      mark(y, "negative")
      Result ← false
fi
```

Fig. 1. Top-down search algorithm

employs one label to indicate whether a concept has been "visited" or not and another label to indicate whether the subsumption test was "positive", "negative" or has not yet been made. The top-down search procedure gets two concepts as input: the concept c, which has to be inserted, and an element x of X_I, which is currently under consideration. For this concept x we already know that $c \preceq x$, and the top-down search looks at its direct successors with respect to $\prec_i$. Initially, the procedure is called with $x = \top$. For each direct successor y of x, it has to check whether it subsumes c. This is done in the procedure *simple-top-subs?*. Since our hierarchy need not be a tree, y may already have been checked before, in which case it has memorized the result of the subsumption test, and thus need not invoke the expensive subsumption test procedure *sub?*. The direct successors for which the test was positive are collected in a list *Pos-Succ*. If this list remains empty, x is added to the result list; otherwise the top-down search is called for each positive successor, but only if this concept has not been visited before along another path.

The bottom-search can be done again in a dual way. The bottom-up search starts at the bottom of the already computed hierarchy. For each concept $x \in X_i$ under consideration it determines whether x has an immediate predecessor y satisfying $y \preceq c$. If there are such predecessors, they are considered as well; otherwise, x is added to the result list of the bottom search.

```
bottom-search(c, x)
  mark(x, "visited")
  for all y with x ≺ y do
     if simple-bottom-subs?(y, c)
       then Pos-Succ ← Pos-Succ ∪ {y}
     fi
  od
  if Pos-Succ is empty then
     Result ← {x}
  else
    for all y ∈ Pos-Succ do
       if not marked?(y, "visited")
        then Result ← Result
∪ bottom-search(c, y)
       fi
    od
  fi
```

```
simple-bottom-subs?(y, c)
if marked(y, "positive")
 then Result ← true
elseif marked(y, "negative")
 then Result ← false
elseif subs?(c, y)
 then mark(y, "positive")
    Result ← true
else
 mark(y, "negative")
 Result ← false
fi
```

Fig. 2. Bottom-up search algorithm

5 ENHANCED TOP-DOWN AND BOTTOM-UP SEARCHES

As subsumption testing is expensive, it is important to ensure that the classification process uses the smallest number of tests. One goal of this paper is to optimize top-down and bottom-up searches for minimizing subsumption test. In order to perform this study, we adapt the following eight methods.

5.1 Optimizing Methods

We propose an enhanced method to optimize the ontology classification process of ontology reasoning. First, during the top-down search, we can take results of subsumption tests that have already been performed and the advantage of the transitivity of the subsumption relation by propagating failed results down the hierarchy or propagating successful results up the hierarchy. Second, in the bottom-up search, we can use the information gained during the top-down search as well.

Negative Information Down Propagation: When classifying a concept A, the top-down search takes advantage of the transitivity of the subsumption relation by propagating failed results down the hierarchy.

$$A \not\sqsubseteq B \text{ and } C \sqsubseteq B \text{ implies } A \not\sqsubseteq C$$

Positive Information Up Propagation: When classifying a concept A, the top-down search takes advantage of the transitivity of the subsumption relation by propagating successful results up the hierarchy.

$$A \sqsubseteq B \text{ and } B \sqsubseteq C \text{ implies } A \sqsubseteq C$$

Negative Information Up Propagation: When finding children of a concept A, the bottom-up search takes advantage of the transitivity of the subsumption relation by propagating failed results down the hierarchy.

$$B \not\sqsubseteq A \text{ and } B \sqsubseteq C \text{ implies } C \not\sqsubseteq A$$

Positive Information Down Propagation: When finding children of a concept A, the bottom-up search takes advantage of the transitivity of the subsumption relation by propagating successful results down the hierarchy.

$$B \sqsubseteq A \text{ and } C \sqsubseteq B \text{ implies } C \sqsubseteq A$$

Positive information gain for top-down search: It concludes, without performing a subsumption test, that if A is subsumed by B, then it can be subsumed by any other concept that is subsuming B.

$$A \sqsubseteq B \text{ and } B \sqsubseteq C \text{ implies no top-search}(A, C)$$

Negative information gain for top-down search: It concludes, without performing a subsumption test, that if A cannot be subsumed by B, then it cannot be subsumed by any other concept that is subsumed by B.

$$A \not\sqsubseteq B \text{ and } C \sqsubseteq B \text{ implies no top-search}(A, C)$$

Positive information gain for bottom-up search: It concludes, without performing a subsumption test, that if A subsumes B, then it can subsume any other concept that is subsumed by B.

$$B \sqsubseteq A \text{ and } C \sqsubseteq B \text{ implies no bottom-search}(A, C)$$

Negative information gain for bottom-up search: It concludes, without performing a subsumption test, that if A cannot subsume B, then it cannot subsume any other concept that is subsuming B.

$$B \not\sqsubseteq A \text{ and } B \sqsubseteq C \text{ implies no bottom-search}(A, C)$$

5.2 Enhanced Top-Down and Bottom-Up Search Algorithm

When trying to take advantage of subsumption tests that have already been performed during the top-down search, one can either concentrate on negative information (*Negative information gain for top-downs earch*) or on positive information (*Positive information gain for top-down search*).

In order to use negative information during processing the top-down search, the enhanced algorithm checks whether for some predecessor z of y the test $c \prec z$ has failed. In this case, we can conclude that $c \not\prec y$ without performing the expensive subsumption test [5]. In order to gain maximum advantage, all predecessors of y should have been tested before the test is performed on y. To use positive information during processing the top-down search, we check whether for some successor z of y the test $c \prec z$ has succeeded. In this case, we can conclude that $c \prec y$ without performing expensive subsumption tests. In order to gain maximum advantage, all successors of y should have been tested before the test is performed on y.

```
top-search(c, x)
  mark(x, "visited")
  for all y with y ≺ x do
     if enhanced-top-subs?(y, c)
       then Pos-Succ ← Pos-Succ ∪ {y}
            propagate-information("Positive", y)
     else
            propagate-information("Negative", y)
     fi
  od
  if Pos-Succ is empty then
     Result ← {x}
  else
    for all y ∈ Pos-Succ do
       if not marked?(y, "visited")
         Result ← Result ∪ top-search(c, y)
       fi
    od
  fi
```

Fig. 3. Enhanced top-down search algorithm

We are only interested in minimizing the number of comparison operations. Therefore, it is more efficient to propagate positive information up (*Positive Information Up Propagation*) and to propagate negative information down (*Negative Information Down Propagation*) through the subsumption hierarchy. This can be achieved by an easy modification of the *procedure top-search?* in Figure 1. When the call "*enhanced-top-subs?(y,c)*" yields true, not only y is marked "*positive*", but

also all of y's predecessors are marked "*positive*". When the call "*enhanced-top-subs?(y,c)*" yields false, not only y is marked "*negative*", but also all of y's successors are marked "*negative*". Obviously, this technique can be combined with the enhanced top-down search described in Figure 3. Figure 4 shows the procedure *enhanced-top-subs?* which adapts *Positive information gain for top-down* search and *Negative information gain for top-down search.*

```
enhanced-top-subs?(y, c)
if marked(y, "positive")
 then Result ← true
elseif marked(y, "negative")
    then Result ← false
elseif exists z with z ≺ y, marked?(z, "positive")
    then mark(y, "positive"), Result ← true
elseif exists z with y ≺ z, marked?(z, "negative")
    then mark(y, "negative"), Result ← false
elseif subs?(y, c)
   then mark(y, "positive"), Result ← true
else mark(y, "negative"), Result ← false
fi
```

Fig. 4. Enhanced top-subs? algorithm

In Figure 5, the comparison of classical top-down search with the enhanced top-down search is shown. The classical top-down search calls "*subs*" eight times; on the other hand, the enhanced top-down search calls "*subs*" six times. Because concept C cannot be subsumed by *X3*, the successors *Y1, Y3, Y4, Z1, Z3, Z4* and B of the concepts *X3* have "*negative*" information. Therefore, it does not need to call *subs?(Y1,C)* and *subs?(Y3,C).*

Now we turn to the enhanced bottom-up search. Of course, optimizations dual to the ones described for the enhanced top-down search can be employed here. In order to use negative information during the enhanced bottom-up search, we check whether for some successor z of y the subsumption test $z \prec c$ has failed. In this case, we can conclude that $y \not\prec c$ without performing the expensive subsumption test. In order to gain maximum advantage, all successors of y should have been tested before the test is performed on y. To use positive information during the enhanced bottom-up search, we check whether for some predecessor z of y the test $z \prec c$ has succeeded. In this case, we can conclude that $y \prec c$ without performing the expensive subsumption test. In order to gain maximum advantage, all predecessors of y should have been tested before the test is performed on y.

It is more efficient to propagate positive information down (*Positive Information Down Propagation*) and to propagate negative information up (*Negative Information Up Propagation*) through the subsumption hierarchy. This can be achieved by an easy modification of the procedure *bottom-search?* in Figure 2. When the call

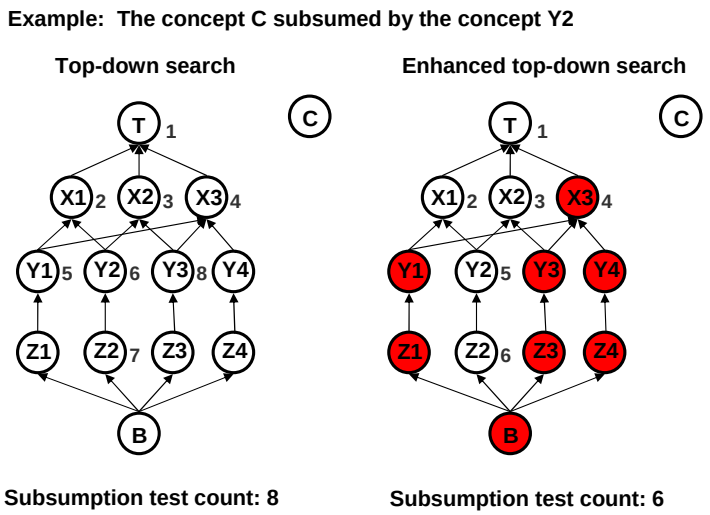

Fig. 5. Example 1 – enhanced top-down search processing

"*subs?(c,y)*" yields true, not only y is marked "*positive*", but also all of *y*'s successors are marked "*positive*". When the call "*subs?(c,y)*" yields false, not only *y* is marked "*negative*", but also all of *y*'s predecessors are marked "*negative*". Obviously, this technique can be combined with the enhanced bottom-down search described in Figure 6. Figure 7 shows the procedure *enhanced-bottom-subs?* which adapts *Positive information gain for bottom-up search* and *Negative information gain for bottom-up search*.

Figure 8 shows a comparison of the classical bottom-up search with the enhanced bottom-up search. The classical bottom-up search calls "*subs?*" nine times; on the other hand, the enhanced bottom-up search calls "*subs?*" six times. Because concept *C* cannot subsume *Z4*, the predecessors *X1, X2, X3, Y2, Y3, Y4* and $\top$ of the concepts *Z4* have "*negative*" information. Therefore, it does not need to call *subs?(C,Y2)*, *subs?(C,Y3)* and *subs?(C,X1)*.

6 EVALUATION

We have tested our implementation using several TBoxes derived from application ontologies. All experiments used Windows XP on a Pentium 4, 1.73 GHz machine with 512 MB of memory. Figure 9 shows results to compare a number of subsumption tests of the enhanced search with the classical search. We decide the optimization level of the algorithm based on a number of subsumption test.

Koala ontology has 21 concepts. The proposed enhanced search performs the subsumption test 124 times. On the other hand, the classical search performs the subsumption test 131 times. The enhanced search performs not only the subsump-

```
bottom-search(c, x)
  mark(x, "visited")
  for all y with x ≺ y do
     if enhanced-bottom-subs?(y, c)
      then positive-down-propagate(y)
Pos-Succ ← Pos-Succ ∪ {y}
     else negative-up-propagate(y)
     fi
  od
  if Pos-Succ is empty
     then Result ← {x}
  else
    for all y ∈ Pos-Succ do
       if not marked?(y, "visited")
        then Result ← Result ∪ bottom-search(c, y)
       fi
    od
  fi
```

Fig. 6. Enhanced bottom-up search algorithm

```
enhanced-bottom-subs?(y, c)
if marked(y, "positive")
   then Result ← true
elseif marked(y, "negative")
   then Result ← false
elseif exists z with z ≺ y and marked?(z, "negative")
   then mark(y, "negative"), Result ← false
elseif exists x with y ≺ x and marked?(x, "positive")
   then mark(y, "positive"), Result ← true
elseif subs?(c, y)
   then mark(y, "positive"), Result ← true
else mark(y, "negative"), Result ← false
fi
```

Fig. 7. Enhanced bottom-subs? algorithm

Ontology	Concept number	Enhanced search	Classical search
koala.owl	21	124	131
Pizza.owl	76	782	1 220
miniEconomy.owl	337	15 931	20 978
miniTransport.owl	444	31 967	33 386

Table 1. The result to compare a number of subsumption test

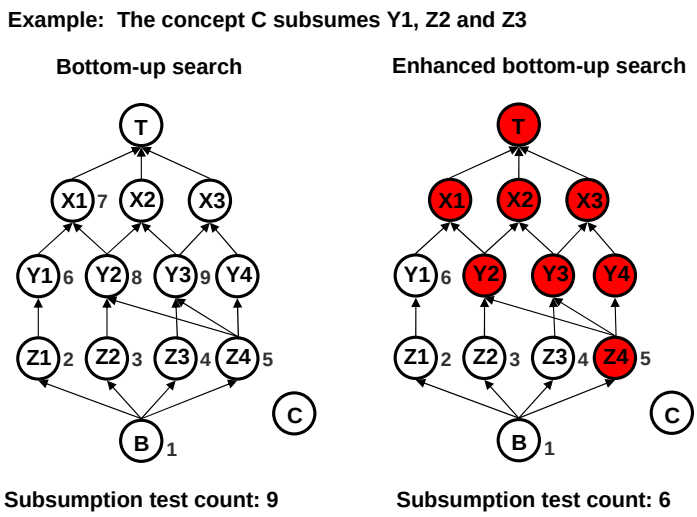

Fig. 8. Example 2 – enhanced bottom-up search processing

tion test but also confirmation about the subsumption relation between predecessors and successors in the concept hierarchy. As we said earlier, the enhanced search applies eight optimization methods. When the enhanced search performs the subsumption test, these methods get data and are able to omit some of the subsumption tests.

The enhanced search method shows a 30 % performance improvement as compared with the classical method according to the result of the experiment. These results lead to the fact that decrement of the subsumption test is not related to ontology size, but is related to the complexity concept hierarchy.

7 CONCLUSION

In this paper, we propose an enhanced method for optimizing the ontology classification process in ontology reasoning. Ontology reasoners are used to classify concepts in ontology, that is to compute a partial ordering or hierarchy of named concepts in the ontology based on subsumption testing. As subsumption testing is costly, it is important to ensure that the classification process uses the smallest number of tests. Most of the reasoners use both top-down and bottom-up searches for ontology classification. One goal of this paper is to optimize top-down searches and bottom-up searches for minimizing subsumption tests. In order to perform this study, we adapted eight methods. First, during the top-down search, we can take results of tests that have already been performed and the advantage of the transitivity of the subsumption relation by propagating failed results down the hierarchy or propagating successful results up the hierarchy. Second, in the

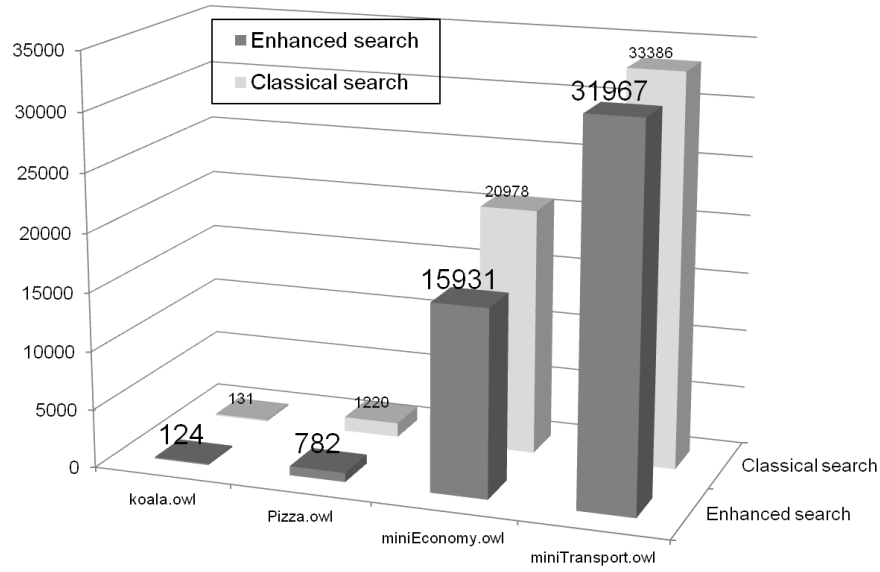

Fig. 9. The result of performance test

bottom-up search, we can use the information gained during the top-down search as well.

As a result of this optimization, a number of necessary comparison operations can be cut down to a fraction compared with the classical top-down search and the classical bottom-up search. The enhanced search method shows a 30 % performance improvement as compared with the classical method according to the result of the experiment.

Acknowledgment

This research has been supported by Soongsil University.

REFERENCES

[1] Gruber, T. R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Presented at the Padua Workshop on Formal Ontology, Vol. 43, November 1995, pp. 907–928.

[2] Baader, F.—Hollunder, B.—Nebel, B.—Profitlich, H.-J.: An Empirical Analysis of Optimization Techniques for Terminological Representation System: Principles of Knowledge Representation and rRasoning. Proceedings of the 3th International Conference, Cambridge (MA). October 1992.

[3] Dean, M.—Schreiber, G.: OWL Web Ontology Language Reference W3C Recommendation. http://www.w3.org/tr/owl-ref/. February 2004.

[4] Schmidt-Schauss, M.—Smolka, G.: Attribute Concept Descriptions with Complements. Artificial Intelligence, Vol. 48, pp. 1–26.

[5] MacGregor, R.: A Deductive Pattern Matcher. In Proceedings of the 7th National Conference of the American Association for Artificial Intelligence, Saint Paul (MI), pp. 403–408.

[6] Aigner, M.: Combinatorical Search. Teubner, Stuttgart, Germany, 1988.

[7] Baader, F.—Nutt, W.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press 2003, pp. 43–95.

[8] McGuinness, D.—Borgida, A.: Explaining Subsumption in Description Logics. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence.

[9] Schlobach, S.—Cornet, R.: Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. Proceedings of IJCAI, 2003.

[10] Sirin, E.—Parsia, B.—Cuenca Grau, B.—Kalyanpur, A.—Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Journal of Web Semantics, Vol. 5, 2007.

[11] Tsarkov, D.—Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In Proc. of the Int. Joint Conf. on Automated Reasoning, IJCAR, 2006.

[12] Horrocks, I.: Optimizing Tableaux Decision Procedures for Description Logics. Ph. D. thesis, University of Manchester, 1997.

[13] Grant, J.: Classifications for Inconsistent Theories. Notre Dame Journal of Formal Logic, Vol. 19, 1978, pp. 435–444.

Je-Min Kim is a Ph. D. student at School of computing, Soongsil University. In 2004 he received his M. Sc. degree in computer science. He worked previously at the Kangnam University as a teacher in the field of computer science. He is the author and co-author of several scientific papers, and participates in semantic web, ubiquitous computing and mobile computing project. He has strong scientific and development expertise in ontology modeling, ontology reasoning and machine learning.

Soon-Hyen Kwon is a Ph. D. student at School of computing, Soongsil University. In 2000 he received his M. Sc. degree in computer science. He participates in semantic web and multi agent project. He has strong scientific and development expertise in agent and semantic reasoning.

Young-Tack Park is a Professor at School of computing, Soongsil University. He received his M. Sc. degree in computer science in 1980. In 1992, he received his Ph. D. degree in artificial intelligence; with thesis focused on blackboard scheduler control knowledge for heuristic classification. He teaches AI at the Faculty of Computer Science. He is also a supervisor and consultant for Ph. D., master and bachelor studies. He has authored and co-authored multiple research papers and participated in national research projects. His research interest is in semantic web, ontology reasoning and machine learning. He has strong scientific and development expertise in ontology reasoning, agent system and mobile computing.