combiSQORE: An Ontology Combination Algorithm

Rachanee Ungrangsi[1], Chutiporn Anutariya[1], and Vilas Wuwongse[2]

[1] School of Technology, Shinawatra University
99 Moo 10 Bangtoey, Samkok, Pathum Thani, 12160 Thailand
{rachanee,chutiporn}@shinawatra.ac.th
[2] School of Engineering and Technology, Asian Institute of Technology
P.O. Box 4, Klong Luang, Pathum Thani, 12120 Thailand
vw@cs.ait.ac.th

Abstract. Automatic knowledge reuse for Semantic Web applications imposes several challenges on ontology search. Existing ontology retrieval systems merely return a lengthy list of relevant single ontologies, which may not completely cover the specified user requirements. Therefore, there arises an increasing demand for a tool or algorithm with a mechanism to check concept adequacy of existing ontologies with respect to a user query, and then recommend a single or combination of ontologies which can entirely fulfill the requirements. Thus, this paper develops an algorithm, namely *combiSQORE* to determine whether the available collection of ontologies is able to completely satisfy a submitted query and return a single or combinative ontology that guarantees query coverage. In addition, it ranks the returned answers based on their conceptual closeness and query coverage. The experimental results show that the proposed algorithm is simple, efficient and effective.

1 Introduction

Ontology is employed as a means for knowledge sharing and reusing in the *Semantic Web* [3]. References [4, 9, 13] discuss two typical scenarios for ontology reuse in the Semantic Web. The first one envisions that a user expresses his/her requirements as a query and submits it to an ontology search engine to retrieve the most appropriate ontology. If the returned result partially satisfies the user requirements, the user is then required to make additional modification efforts which are considerably less compared to those needed to construct a new ontology from scratch.

On the other hand, the second scenario, which is called *automatic knowledge reuse*, addresses the problem of automatically and dynamically finding a single or combinative ontology for *next generation Semantic Web applications* [11], such as Magpie [7] and PowerAqua [9, 13]. Magpie [7] is a semantic browser which assists users while they surf the Web by highlighting instances of chosen concepts in the current Web page based on an internal instantiated ontology. The second application, PowerAqua [9, 13], is an ontology based question answering system that derives answers to questions asked in natural language by exploiting an underlying ontology. Currently, in both tools, the employed ontology is manually selected by the user and only one ontology can be exploited at a time. To allow cross-domain question

K. Aberer et al. (Eds.): ISWC/ASWC 2007, LNCS 4825, pp. 566–579, 2007.

answering in the case of PowerAqua, and enable an extended coverage of the semantic browsing with Magpie, a mechanism for dynamically finding and combining the relevant knowledge among online ontologies and semantic data becomes essential.

Existing ontology retrieval systems, such as *Swoogle* [6], *OntoKhoj* [11], and *OntoSearch* [16], merely return a lengthy list of single ontologies, but none of them can ensure that all query conditions are met by at least one of the returned results. Furthermore, due to the sparseness of knowledge in a Web-accessible ontology database, it is possible that there exists no single ontology which satisfies all user requirements [13]. However, to date there is no algorithm or tool which can deal with these significant complications.

This paper proposes a simple yet efficient and effective algorithm, namely *combiSQORE*. It does not only enable users to check the concept sufficiency of an ontology collection with respect to a given query, but also computes a sub-optimal combination of ontologies that jointly cover the query when no single ontology can fulfill the specified requirements. In addition, it returns the rankings which rank both single and combinative ontologies based on *conceptual closeness* and *query coverage*.

combiSQORE algorithm is developed as an extension of *SQORE* (*Semantic Query based Ontology Retrieval Framework*) [2, 14]. SQORE enables users to precisely and structurally formulate their ontology requirements in terms of a *semantic query*. Each query is evaluated by considering the semantic closeness between the query itself and the resultant ontology which is quantified by SQORE's *similarity measures*. Comprehensive experiments have been conducted on real-world ontologies to evaluate and demonstrate combiSQORE's effectiveness. The results have shown that the proposed algorithm can generate irreducible combinations of ontologies with a reasonable cost and provide useful rankings.

The paper is organized as follows. Sect. 2 reviews related works and Sect. 3 informally introduces SQORE. Sect. 4 develops combiSQORE algorithm and Sect. 5 illustrates the algorithm via an example. Sect. 6 discusses the conducted experiments and their results, and followed by conclusions and future work in Sect. 7.

2 Related Work

Ontology search engines are crucial to enable scientists and practitioners to find and reuse Web-accessible ontologies efficiently. Several ontology retrieval systems have been developed in the last few years (e.g. *Swoogle* [6], *OntoKhoj* [12], and *OntoSearch* [16].) However, these systems mainly focus on automatically crawling the Web for collecting ontologies and employ traditional keyword search mechanisms to retrieve relevant ontologies. As a result, they fail to capture the structural and semantic information about the user-desired domain concepts and relations. Furthermore, they usually return a large number of ontologies, but cannot guarantee query coverage which is a mandatory requirement for *automatic ontology reuse* in Semantic Web applications, such as an ontology-based browser *Magpie* [7], an ontology-based question answering system *PowerAqua* [9, 13], etc.

Another interesting approach is *CORE* [8] and its extension, *WEBCORE* [4], which retrieves keyword-related ontologies from an ontology database, and applies multiple criteria to generate several rankings, and finally combines all the rankings to

obtain the final ranking. However, some of these ranking criteria require users to provide applications and data for the evaluation. Furthermore, in its last step, a user is demanded to manually evaluate the resultant ontologies in order to enable a collaborative assessment. Thus, this approach cannot readily be applied to automatic ontology reuse in Semantic Web applications.

Swoogle [6] and *OntoKhoj* [12] implement their *PageRank*-like algorithms based on the computed ontology referral network. *ActiveRank* [1] introduces several metrics for ontology ranking based on the taxonomic structure information such as class names, shortest paths, linking density and positions of focused classes in the ontology. However, these three approaches cannot be used for ranking the returned result that consists of both single and combinations of ontologies.

PowerAqua [9, 13] proposes a framework to determine ontology combinations for a given query by using *OntoCombination* algorithm and compute ranking based on the generality of ontology concepts. However, such an algorithm produces a set of ontologies ranked by the coverage of each individual ontology, but does not compute an optimal or sub-optimal combination that maximizes the query coverage.

3 SQORE: Architectural Overview

Fig. 1 illustrates *SQORE*'s system architecture [2, 14] which comprises four main components: i) a *semantic query*, ii) a *retrieval engine*, iii) an *ontology database*, and iv) a *semantic lexical database*. It employs *XML Declarative Description* (*XDD*) *theory* [15] as its theoretical foundation for modeling ontology databases and evaluating semantic queries, which does not only facilitate ontology matching and retrieval, but also support reasoning capability to enhance the matching results. Furthermore, when a query term and an ontology term do not exactly match (=), it determines other possible semantic relations between them (i.e. equivalence *(≡)*, broader *(⊇)*, narrower *(⊆)* and unknown *(≠)*) by employing a referenced lexical database, such as *WordNet* [10]. Then, the system computes the *semantic similarity score* between a given query and an ontology in the collection, which ranges from 0 (strong dissimilarity) to 1 (strong similarity).

By enhancing SQORE with the proposed *combiSQORE* algorithm, the system can then determine whether or not an ontology collection is conceptually sufficient for a user query, and recommend a single or combinative ontology which completely cover

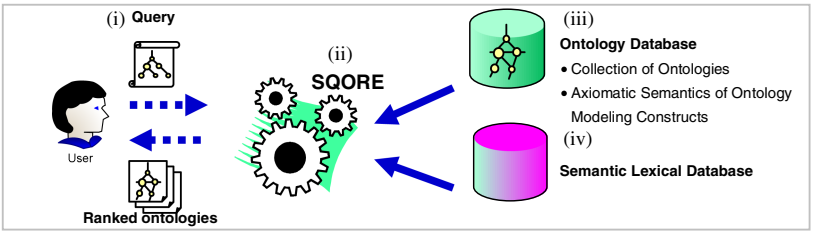


Fig. 1. SQORE System Architecture

the query. Finally, the system computes *semantic similarity scores* between the query and the returned ontologies (either single or combinative) based on *conceptual similarity*, and *query coverage* and uses these scores for the rankings.

SQORE defines four measures used for calculating similarity scores as follows:

- **Element Similarity Score (SS_E):** The similarity score of any two given elements x and y, denoted by $SS_E(x, y)$, depends on their semantic relation determined by the referenced lexical database as explained earlier. For any two given restrictions $r(a_1,b_1)$ and $r(a_2,b_2)$, their similarity is equal to the product of a_1-a_2 similarity score and b_1-b_2 similarity score i.e., $SS_E(a_1, a_2)^* SS_E(b_1, b_2)$. When x and y do not belong to the same type, for instance x is a class name and y a property name, their similarity score is undefined.
- **Best Similarity Score (SS_B):** Based on the element similarity score SS_E, $SS_B(x,O)$ represents the similarity between a given element x of a query and an ontology O by finding the highest similarity score between x and each element y that is semantically defined by O. In other words, the element y in O that is most similar to x, will be used for measuring the closeness between x and O. This measure is a key metric in the combiSQORE algorithm.
- **Satisfaction Score of Mandatory conditions (SS_M) and Optional conditions (SS_O):** In SQORE, a semantic query comprises mandatory conditions and optional conditions. If an ontology semantically satisfies all mandatory conditions of a given query, then that ontology will be included in the answer. Optional conditions, on the other hand, are useful for expressing additional means for measuring the extent of closeness between the ontology and the query.
- **Query-Ontology Similarity Score (SS):** This similarity score represents the semantic closeness between a query and an ontology, which is measured by the satisfaction degree of the ontology with respect to the mandatory and optional conditions of the query.

4 Algorithms: Ontology Combination and Ranking

Formally, the problem of finding an ontology combination is: *Given a semantic query and a set of ontologies, determine a minimal ontology subset that satisfies all conditions in the query, and maximizes the conceptual closeness between the ontology subset and the query*. This problem is equivalent to the *knapsack problem*, which is widely-known to be NP-complete. Therefore, rather than developing an optimal solution, this paper proposes a backward greedy algorithm for construction of an *irreducible* ontology subset, which satisfies all conditions in the query.

4.1 Notations and Definitions

Throughout this section, let $\mathcal{ODB} = \{O_1, \ldots, O_n\}$ be an ontology collection consisting of n ontologies and $\mathcal{Q} = \{q_1, \ldots, q_m\}$ be a semantic query comprising m conditions. As means for measuring the relevance of an ontology O in $\mathcal{ODB}$ with respect to a condition q of $\mathcal{Q}$, SQORE [2, 14] defines $SS_B(q, O)$ as the (best) similarity score between q and O, which ranges from 0 (strong dissimilarity) to 1 (strong similarity).

Based on $SS_B(q, O)$, let $S(q,\mathcal{ODB}) \subseteq \mathcal{ODB}$ be the set of ontologies relevant to a condition q, defined as follows:

$$S(q,\mathcal{ODB}) = \{\ O \in \mathcal{ODB} : SS_B(q, O) > 0, q \in \mathcal{Q}\ \} \quad (1)$$

Definition 1. An ontology collection $\mathcal{ODB}$ is *sufficient* to satisfy a semantic query $\mathcal{Q}$ if and only if

$$\forall q \in \mathcal{Q}, S(q,\mathcal{ODB}) \neq \varnothing\ . \qquad \square$$

Intuitively speaking, if $S(q,\mathcal{ODB})$ is the empty set, one can derive that there exists no ontology in $\mathcal{ODB}$ that can satisfy such a query condition q in $\mathcal{Q}$. Therefore, an ontology collection $\mathcal{ODB}$ is said to be *sufficient for* a semantic query $\mathcal{Q}$, if there exists a non-empty subset of $\mathcal{ODB}$ which jointly satisfies all conditions in $\mathcal{Q}$; otherwise $\mathcal{ODB}$ is *insufficient*.

Definition 2. Let $\mathcal{R} \subseteq \mathcal{ODB}$. $\mathcal{R}$ is a *query result* of $\mathcal{Q}$ if $\mathcal{R}$ is sufficient for $\mathcal{Q}$. $\mathcal{R}$ is a *candidate query result* of $\mathcal{Q}$, if $\mathcal{R}$ is a *query result* and *minimal* (*irreducible*). That is, any subset of a candidate query result $\mathcal{R}$ must not be a candidate query result of $\mathcal{Q}$, and hence removing any ontology O from $\mathcal{R}$ leads to an unsatisfactory of some query conditions q in $\mathcal{Q}$. $\square$

Next, an algorithm, namely *combiSQORE*, which can generate a candidate query result of $\mathcal{Q}$, is devised.

4.2 CombiSQORE Algorithm

Fig. 2 presents combiSQORE algorithm, which takes three input parameters: a semantic query $\mathcal{Q}$, a set of ontologies $\mathcal{ODB}$ and a sequence l, and returns a candidate query result $\mathcal{R}$ of $\mathcal{Q}$. Firstly, it determines whether or not the ontology collection $\mathcal{ODB}$ is sufficient to satisfy $\mathcal{Q}$. If $\mathcal{ODB}$ is insufficient for $\mathcal{Q}$, the algorithm exits and returns the empty set—no query result for $\mathcal{Q}$. If $\mathcal{ODB}$ is sufficient, $\mathcal{ODB}$ itself is a query result for $\mathcal{Q}$. Therefore, $\mathcal{R}$ is initially assigned to be equal to $\mathcal{ODB}$. The next for-loop then minimizes $\mathcal{R}$ by considering each ontology O in $\mathcal{R}$ according to the input sequence l. If $\mathcal{R} - \{O\}$ is insufficient for $\mathcal{Q}$, O cannot be removed from $\mathcal{R}$; otherwise $\mathcal{R}$ is minimized by taking O out. This iteration continues until there is no ontology remaining in the sequence l. The algorithm then returns $\mathcal{R}$ as a candidate query result.

One can see that with a different ontology sequence l, combiSQORE may produce different candidate query result $\mathcal{R}$ for a particular query $\mathcal{Q}$ and ontology collection $\mathcal{ODB}$, since the sequence l determines the order of removing an ontology from an initial query result in order to finally obtain a candidate query result. Note that the conducted experiments show that strategically generated input sequences can improve the algorithm performance (to be discussed in more details in Section 6).

Let m denote the size of a given query $\mathcal{Q}$ and n the size of an ontology collection $\mathcal{ODB}$. The complexity of combiSQORE is $O(mn^2 \log n)$ or $O(n^2 \log n)$ when $m << n$.

```
Algorithm combiSQORE(Q, ODB, l)
Input:   Q: a semantic query,
         ODB: an ontology collection,
         l: a predetermined sequence of ontologies in ODB
Output:  R: a candidate query result
if ∃q ∈ Q such that S(q, ODB) = ∅
    do EXIT  // ODB is insufficient for Q
R = ODB
for each ontology O in the sequence l
    do T = R − {O}
         if  ∃q ∈ Q such that  S(q, T) = ∅
             R = R  // T is insufficient for Q
         else
             R = T  // T is sufficient for Q
return R
```

Fig. 2. combiSQORE: an ontology combination algorithm

Theorem 1. If an ontology collection $\mathcal{ODB}$ is *sufficient* for a given semantic query $\mathcal{Q}$, then $\mathcal{P} \supseteq \mathcal{ODB}$ is also *sufficient* for $\mathcal{Q}$.

Proof: Assume that there exists $\mathcal{P} \supseteq \mathcal{ODB}$ that is insufficient for $\mathcal{Q}$. Then, by definition, there exist $q \in \mathcal{Q}$ such that

$$
\begin{aligned}
S(q, \mathcal{P}) &= \varnothing \\
\{ O \in \mathcal{P} : SS_B(q, O) > 0\} &= \varnothing \\
\{ O \in \mathcal{P} : SS_B(q,O) > 0\} \cap \mathcal{ODB} &= \varnothing \cap \mathcal{ODB} \\
\{ O \in \mathcal{P} \cap \mathcal{ODB} : SS_B(q, O) > 0\} &= \varnothing \cap \mathcal{ODB} \\
\{ O \in \mathcal{ODB} : SS_B(q, O) > 0\} &= \varnothing \quad \text{// since } \mathcal{P} \supseteq \mathcal{ODB} \\
S(q,\mathcal{ODB}) &= \varnothing
\end{aligned}
$$

which contradicts the assumption that $\mathcal{ODB}$ is sufficient for $\mathcal{Q}$. ❑

Theorem 2. A candidate query result $\mathcal{R}$ returned by combiSQORE is *irreducible*.

Proof: For the sake of contradiction, let $X \subseteq \mathcal{R}$ and $X \neq \varnothing$, and assume that $\mathcal{R} - X$ is a query result of $\mathcal{Q}$. For an ontology $O \in X$, let i be the iteration in which combiSQORE considers to remove O and let $\mathcal{R}_i \supseteq \mathcal{R}$ be the query result at the beginning of this iteration. For the ontology O to remain in the query result, it must be that $\mathcal{R}_i - \{O\}$ is insufficient to satisfy all query conditions; otherwise combiSQORE would have removed O from $\mathcal{R}_i$. Therefore, $\mathcal{R}_i - \{O\}$ is not a query result of $\mathcal{Q}$. Since

$\mathcal{R} - X$ is a query result of $\mathcal{Q}$, and $\mathcal{R} - X \subseteq \mathcal{R} - \{O\} \subseteq \mathcal{R}_i - \{O\}$, from Theorem 1 one can obtain that $\mathcal{R}_i - \{O\}$ is also a query result of $\mathcal{Q}$, which contradicts. ❑

4.3 Ranking Mechanism

Two criteria, namely *query coverage* and *conceptual closeness* are considered to compute semantic similarity score which is used for ranking query results generated by combiSQORE. Firstly, query coverage is defined to determine how well an ontology combination $\mathcal{R}$ satisfies a given query $\mathcal{Q}$. Intuitively, it is measured by computing the ratio of the number of conditions satisfied by $\mathcal{R}$ to the total number of conditions in $\mathcal{Q}$, hence its value ranges from 0 to 1. Since a candidate query result produced by combiSQORE guarantees to satisfy all conditions in $\mathcal{Q}$, its query coverage is 1.

Definition 3 (Query Coverage Score: *QS*). The query coverage between a semantic query $\mathcal{Q}$ comprising m conditions $q_1, \ldots, q_m$ and a set of ontologies $\mathcal{R}$ consisting of n ontologies $O_1, \ldots, O_n$ is measured by:

$$QS(\mathcal{Q}, \mathcal{R}) = \frac{\left|\left\{ q \in \mathcal{Q} : S(q, \mathcal{R}) \neq \varnothing \right\}\right|}{M} \tag{2}$$

❑

Next, the conceptual closeness between a query and a candidate query result comprising one or more ontologies will be formalized, by redefining certain semantic similarity measures developed by SQORE [2, 14], which simply capture the conceptual similarity between a query $\mathcal{Q}$ and a single ontology O. Intuitively, based on $SS_B(q, O)$ which defines the *(best) similarity score* between a condition q in $\mathcal{Q}$ and the ontology O, SQORE defines the *query-ontology similarity score*: $SS(\mathcal{Q}, O)$ to represent the conceptual closeness between $\mathcal{Q}$ and ontology O by simply aggregating the similarity scores between all conditions in $\mathcal{Q}$ and O.

Therefore, in order to measure the *conceptual closeness* between $\mathcal{Q}$ and a combination of ontologies $\mathcal{R}$, the *query-combinative-ontology conceptual similarity score*: $SS_C(\mathcal{Q}, \mathcal{R})$ is formalized here by aggregating the maximum similarity score between a query condition q in $\mathcal{Q}$ and an ontology in $\mathcal{R}$ as follows.

Definition 4 (Query-Combinative-Ontology Conceptual Similarity Score: SS_C). The conceptual closeness between a semantic query $\mathcal{Q}$ comprising m conditions $q_1, \ldots, q_m$ and a set of ontologies $\mathcal{R}$ consisting of n ontologies $O_1, \ldots, O_n$ is measured by:

$$SS_C(\mathcal{Q}, \mathcal{R}) = \frac{\sum_{i=1}^{m} \max_{O \in \mathcal{R}} SS_B(q_i, O)}{m} \tag{3}$$

❑

Finally, QS and SS_C are combined in order to measure the semantic similarity between $\mathcal{Q}$ and a combination of ontologies $\mathcal{R}$, as follows.

Definition 5 (Query-Combinative-Ontology Similarity Score: *SS*). The semantic similarity between a semantic query $\mathcal{Q}$ comprising m conditions $q_1, \ldots, q_m$ and a set of ontologies $\mathcal{R}$ consisting of n ontologies $O_1, \ldots, O_n$ is measured by:

$$SS(\mathcal{Q},\mathcal{R}) = QS(\mathcal{Q},\mathcal{R})*SS_C(\mathcal{Q},\mathcal{R}) \quad (4)$$

❑

Next section elaborates more details by means of an example.

5 An Example

Let $\mathcal{ODB}$ be an ontology database comprising eight real-world OWL ontologies from different sources as shown in Table 1. Assume that a query $\mathcal{Q}$ comprising eight conditions is submitted, and the SS_B matrix measuring the similarity between each ontology and query condition is given in Table 2. From the table, one can see that O2, O3 and O6 have the highest similarity scores, and are ranked 1st, 2nd and 3rd, respectively. Moreover, Table 2 also depicts that each query condition is satisfied by more than one ontology in the collection. Thus, $\mathcal{ODB}$ is sufficient to satisfy $\mathcal{Q}$. However, there exists no single ontology that can satisfy all query conditions, which results in a need for combiSQORE algorithm to generate candidate query results and compute the ranking.

Table 1. An example of ontology database $\mathcal{ODB}$

Ontology	URI
O1	http://swrc.ontoware.org/ontology
O2	http://ebiquity.umbc.edu/ontology/person.owl
O3	http://annotation.semanticweb.org/iswc/iswc.owl
O4	http://ontoware.org/frs/download.php/18/semiport.owl
O5	http://morpheus.cs.umbc.edu/aks1/ontosem.owl
O6	http://www.csd.abdn.ac.uk/~cmckenzi/playpen/rdf/akt_ontology_LITE.owl
O7	http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl
O8	http://protege.stanford.edu/plugins/owl/owl-library/ka.owl

Let the input sequence l of combiSQORE be (O5,O2,O3,O7,O8,O6,O1,O4). The algorithm starts with an initial query result $\mathcal{R}$ comprising all ontologies. Then, it iteratively checks whether removing an ontology from $\mathcal{R}$ according to the order of the input sequence makes $\mathcal{R}$ insufficient for $\mathcal{Q}$ or not. If $\mathcal{R}$ remains sufficient, that ontology is removed from $\mathcal{R}$; otherwise, $\mathcal{R}$ is unchanged. For instance, removing Ontology O6 from the query result $\mathcal{R}$ = {O1, O4, O6} will cause q7 and q8 unsatisfied. Thus, O6 cannot be removed from $\mathcal{R}$.

With respect to the given ontology collection $\mathcal{ODB}$, the submitted query $\mathcal{Q}$ and the input sequence l, combiSQORE generates the candidate query result $\mathcal{R}$ = {O1, O6}, which is irreducible because removing either O1 or O6 will make some query

Table 2. SS_B matrix between a query condition in $\mathcal{Q}$ and an ontology in $\mathcal{ODB}$

Query Conditions in $\mathcal{Q}$	**O1**	**O2**	**O3**	**O4**	**O5**	**O6**	**O7**	**O8**
q1: <owl:Class rdf:ID="Student" />	1	1	1	1	1	1	1	1
q2: <owl:Class rdf:ID="PhDstudent"/>	1	1	1	1	0	1	0.6	1
q3: <owl:Class rdf:ID="Professor"/>	0.4	1	0.4	0.4	1	0.8	1	0
q4: <rdf:Property rdf:ID="supervise"/>	1	0	0	0	1	0	0	1
q5: <owl:Class rdf:about="PhDStudent" > <rdfs:subClassOf rdf:resource ="Student"/> </owl:Class>	1	1	1	1	0	0	0	0
q6: <rdf:Property rdf:about="firstname">	0	1	1	0.6	0	0.8	0.6	1
q7:<rdfs:domain rdf:resource ="Student"/>	0	1	1	0	0	0.8	0	1
q8:<rdfs:range rdf:resource ="xsd:String"/>	0	1	1	0	0	0.8	0	0
SEMANTIC SIMILARITY SCORE:	**0.55**	**0.87**	**0.8**	**0.5**	**0.37**	**0.65**	**0.4**	**0.62**

Table 3. Sample input sequences and their output combinations

Input Sequence	**Candidate Query Result**
Seq1: (O5,O2,O3,O7,O8,O6,O1,O4)	{O1,O6}
Seq2: (O2,O3,O6,O8,O1,O4,O7,O5)	{O4,O5,O6}
Seq3: (O5,O7,O4,O1,O8,O6,O2,O3)	{O3,O8}
Seq4: (O8,O7,O6,O4,O3,O5,O2,O1)	{O1,O2}

conditions unfulfilled. In addition, since different input sequences may yield different candidate query results, Table 3 gives other possible results.

In order to rank the top three single ontologies (i.e., O2, O3 and O6) together with the four candidate query results of Table 3, Table 4 illustrates their computed scores: query coverage score, conceptual closeness score and similarity score with the corresponding rankings shown in the followed brackets. With a focus on the final similarity scores, a combinative ontology, namely {O1,O2}, is ranked 1st, because it can satisfy all query conditions with highest conceptual closeness scores, while single ontologies fail to fulfill certain conditions and have lower conceptual closeness scores.

Table 4. Different rankings based on three ranking criteria

Ontologies	**Query Coverage Score (QS)**	**Conceptual Closeness Score (SS_C)**	**Similarity Score ($SS=QS*SS_C$)**
O2	0.875 (5)	0.87 (5)	0.761 (5)
O3	0.875 (5)	0.8 (6)	0.7 (6)
O6	0.75 (7)	0.65 (7)	0.488 (7)
{O1,O6}	1 (1)	0.9 (4)	0.9 (4)
{O4,O5,O6}	1 (1)	0.925 (2)	0.925 (2)
{O3,O8}	1 (1)	0.925 (2)	0.925 (2)
{O1,O2}	1 (1)	1 (1)	1 (1)

6 Experiments and Results

This section evaluates combiSQORE algorithm in terms of its performance and the validity of its rankings by means of experiments. An ontology database used in the experiment comprised 63 ontologies collected from three different domains: computer science, food and stock, while queries were automatically created by randomly selecting usable exact keywords from Wikipedia pages as shown in Table 5. The *total number of keywords* indicates the number of keywords extracted from the Wikipedia pages without considering stop words. The number of *usable exact keywords* represents the number of extracted keywords that can exactly match with concepts (classes) in the ontology database. The number of *usable related keywords* includes synonyms, hypernyms and hyponyms of the usable exact keywords which appear in the ontology collection.

Table 5. Statistics of Wikipedia pages used for generating keywords

Domain	Wikipedia page	Total Keywords	Usable Exact Keywords	Usable Related Keywords
Stock	http://en.wikipedia.org/wiki/Stock	493	202	2027
Food	http://en.wikipedia.org/wiki/Food	672	259	732
Comp.Sc.	http://en.wikipedia.org/wiki/Computer_science	283	107	733
TOTAL		1448	568	3492

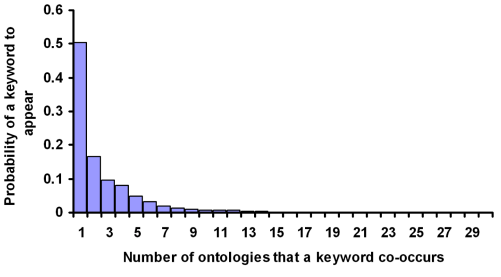


Fig. 3. Richness of knowledge in the ontology collection

Fig. 3 presents the richness of knowledge in the ontology database based on how often exact and related keywords appear in different number of ontologies varying from one to twenty-nine. The graph shows that the probability that a keyword will appear in only one ontology is approximately 0.5. However, the probability of a keyword to co-occur in a higher number of ontologies decreases dramatically. Hence, given a random set of keywords, the chance that they all will co-occur in the same ontology is considerably low.

The experiment has been designed to test not only how well the algorithm performs in average, but also to investigate the impact of input sequences to the algorithm performance. Therefore, the experiment was performed as follows. Firstly,

a set of *n* keywords were randomly selected to formulate an input query, varying from $n = 1$ to 10. Then, obtain the set of relevant ontologies from SQORE system, and apply combiSQORE algorithm with a designated input sequence. Certain analyses on the obtained results were then performed, as illustrated by Fig. 4 and Fig. 5. Note that each data point shown in the graphs represents the average value obtained from at least 50 trials or more.

6.1 Algorithm Performance

Fig. 4 illustrates the average number of relevant ontologies returned from SQORE, the average number of single ontologies that can satisfy all query conditions regardless to the conceptual similarity, and the average size of candidate query results. As expected, when the number of query conditions increases, the number of retrieved ontologies also increases whereas the number of single ontologies that can satisfy all query conditions decreases to zero. This result reflects the need for ontology combinations in order to entirely cover all conditions. In addition, the experimental result has shown that the average size of ontology combinations is approximately 3 for ten query conditions, which is acceptable for ontology integration.

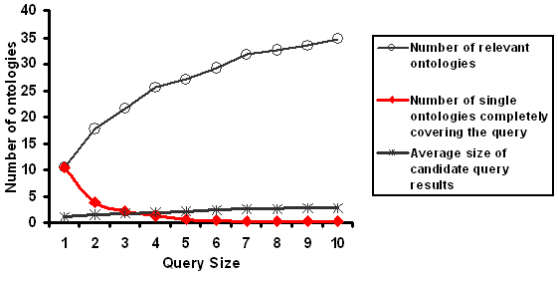
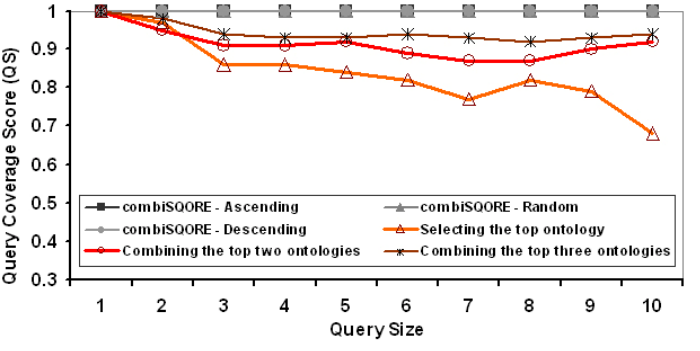


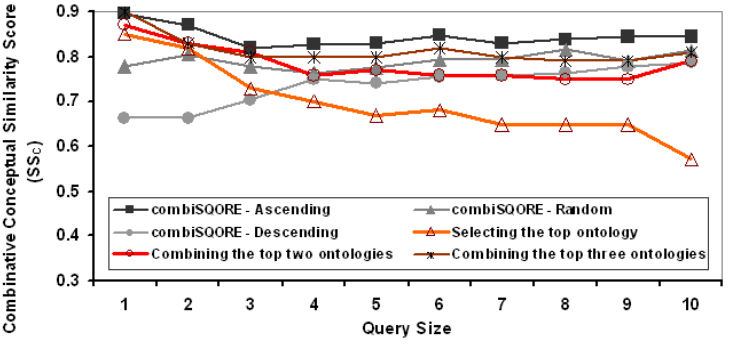
Fig. 4. Comparisons of resultant ontologies, individuals and combinations

As discussed earlier, with different input ontology sequences, combiSQORE may yield different combinative ontologies because a sequence determines the order of removing an ontology from an initial query result in order to finally obtain a candidate query result. Therefore, the algorithm performance is suspected to be improved if such a sequence is strategically generated. Intuitively, to maximize the conceptual closeness, the sequence should be sorted in ascending order of the similarity score. Since the similarity score tends to be proportional to the query coverage, the conducted experiment examined the three types of input sequences: (i) *random* ones, (ii) ones arranged in *ascending* order of the similarity score, and (iii) ones arranged in *descending* order of the similarity score. In addition, to illustrate the effectiveness of combiSQORE algorithm, results are also compared to three common approaches for selecting and combining ontologies regarding to similarity scores: (i) selecting only the highest-scored ontology, (ii) combining the two highest-scored ontologies and (iii) combining the three highest-scored ontologies.

Fig. 5 then presents the average query coverage scores and the average conceptual closeness scores of the computed results based on six different approaches as mention above. As expected, the sequences arranged in ascending order give the best candidate query results, whereas the random ones perform moderately well with the average similarity score of 0.8, which is considerably high. Furthermore, it clearly shows that combiSQORE with input sequences in ascending order of the similarity scores outperforms combining the highest-scored ontologies because the results by combiSQORE always completely satisfy the user query with higher conceptual closeness scores.



a. query coverage



b. conceptual closeness

Fig. 5. Comparisons between the candidate query results of combiSQORE in different input sequences and those of other common approaches

6.2 Ranking Evaluation

In order to evaluate the practicality of the proposed ranking mechanism, a preliminary experiment was conducted. In the experiment, the ontology database and the formulated query of Section 5 was presented to four participants with a request to rank the top three single ontologies (i.e., O2, O3 and O6) together with the four candidate query results of Table 3 based on query coverage, conceptual closeness and similarity scores. Table 6a shows the average rankings proposed by the participants.

Table 6. Ranking evaluation results

a. Average ranks given by participants

Ontologies	Users – Query Coverage	Users - Conceptual Closeness	Users – Overall	combiSQORE- QS	combiSQORE- SS_C	combiSQORE- SS
O2	5	6	6	5	5	5
O3	7	5	5	5	6	6
O6	5	6	7	7	7	7
{O1,O2}	1	2	2	1	1	1
{O1,O6}	3	4	4	1	4	4
{O3,O8}	3	2	3	1	2	2
{O4,O5,O6}	2	1	1	1	2	2

b. Pearson Correlation Coefficient for combiSQORE wrt. participant ranking

combiSQORE	PCC
Query coverage score (QS)	0.815
Conceptual closeness score (SS_C)	0.917
Similarity score (SS)	0.918

Pearson Correlation Coefficient (PCC) [5] is employed to measure the similarity between the average participant rankings and the system rankings. If the calculated PCC value is closer to 1, it indicates a stronger linear relationship between the two rankings. Table 6b shows that the PCC values of the three rankings, based on query coverage, conceptual closeness and similarity scores, are significantly high, which imply that the rankings proposed by combiSQORE are very close to the participant rankings.

7 Conclusions and Future Work

This paper has proposed *combiSQORE*, a novel approach for computing and ranking ontology combinations, which can completely cover the specified user requirements. By integrating a number of ontologies, each partially satisfying the given requirements, the approach generates a minimal query result that can fulfill all requirements. The primary objective of the proposed approach is not only to enable automatic knowledge reuse for Semantic Web applications, but also to offer alternatives for ontology engineers and practitioners during their ontology search and development processes. In addition, it can also be applied to Web-service discovery applications in order to find sub-optimal sets of Web services that can meet all user requirements.

With a focus on a mechanism for ranking the generated ontology combinations, this paper has also developed simple methods to measure the *conceptual similarity*, and *query coverage* of an ontology combination with respect to a given query. These two criteria are then used to compute meaningful and practical rankings with the promising experimental results. In addition, modification (integration) cost is another metric that users are concerned. Future research direction includes an emphasis on discovering the inter-relationships among the ontologies in a combination, and

integrating such information to compute an accurate modification (integration) cost. Moreover, an enhancement by incorporating combiSQORE algorithm into the current system available on-line at http://ict.shinawatra.ac.th:8080/sqore is under way.

References

1. Alani, H., Brewster, C.: Metrics for Ranking Ontologies. In: Proc. 4th Int. EON Workshop, 15th Int. WWW Conference, Edinburgh (2006)
2. Anutariya, C., Ungrangsi, R., Wuwongse, V.: SQORE – A Framework for Semantic Query based Ontology Retrieval. In: DASFAA 2007. Proc. 12th Int. Conf. Database Systems for Advanced Applications, Bangkok. LNCS, vol. 4443, pp. 924–929 (2007)
3. Berners-Lee, T., Handler, J., Lassila, O.: The Semantic Web. Scientific American, Singapore (May 2001)
4. Cantador, I., Fernández, M., Castells, P.: Improving Ontology Recommendation and Reuse in WebCORE by Collaborative Assessments. In: Proc. 4th Int. Workshop Evaluation of Ontologies for the Web, 15th Int. WWW Conf., Edinburgh (2006)
5. Conover, W.J.: Practical Non-Parametric Statistics, 2nd edn. John Wiley and Sons, Chichester (1980)
6. Ding, L., Finin, T., Joshi, A., Pan, R., Scott Cost, R., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the Semantic Web. In: Proc. 13th ACM Int. Conf. Information and Knowledge Management, DC (November 2004)
7. Dzbor, M., Domingue, J., Motta, E.: towards a Semantic Web browser. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 690–705. Springer, Heidelberg (2003)
8. Fernández, M., Cantador, I., Castells, P., CORE,: A Tool for Collaborative Ontology Reuse and Evaluation. In: Proc. 4th Int. Workshop Evaluation of Ontologies for the Web, 15th Int. WWW Conf., Edinburgh (2006)
9. Lopez, V., Motta, E., Uren, V.: PowerAqua: Fishing the Semantic Web. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 394–410. Springer, Heidelberg (2006)
10. Miller, A., WordNet, A.: lexical database for English. Communications of the ACM 38(11) (1995)
11. Motta, E., Sabou, M.: Next Generation Semantic Web Applications. In: Proc. 1st Asian Semantic Web Conf., China (2006)
12. Patel, C., Supekar, K., Lee, Y., Park, E.K.: OntoKhoj: a Semantic Web portal for ontology searching, ranking and classification. In: Proc. 5th ACM Int. Workshop Web Information and Data Management, Louisiana (November 2003)
13. Sabou, M., Lopez, V., Motta, E., Uren, V.: Ontology Selection: Ontology Evaluation on the Real Semantic Web. In: Proc. of the 4th Int. EON Workshop, Evaluation of Ontologies for the Web, 15th Int. WWW Conf., Edinburgh (2006)
14. Ungrangsi, R., Anutariya, C., Wuwongse, V.: SQORE-based Ontology Retrieval System. In: DEXA 2007. Proc. 18th Int. Conf. Database and Expert Systems Applications, Regensburg, Germany, vol. 4653, pp. 720–729 (2007)
15. Wuwongse, V., Anutariya, C., Akama, K., Nantajeewarawat, E.: XML Declarative Description (XDD): A Language for the Semantic Web. IEEE Intelligent Systems 16(3), 54–65 (2001)
16. Zhang, Y., Vasconcelos, W., Sleeman, D.: OntoSearch: An ontology search engine. In: Proc. 24th SGAI Int. Conf. Innovative Techniques and Applications of AI, UK (2004)