A Web Service Discovery Scheme Based on Structural and Semantic Similarity

SHIRIN AKTHER KHANAM AND HEE YONG YOUN+
College of Information and Communication Engineering
SungKyunKwan University
Suwon, 440-746 Korea
E-mail: {shirinkhan; youn7147}@skku.edu

With the increasing adoption of Web Services and service-oriented computing paradigm, matchmaking of web services with the request has become a significant task. This warrants the need to establish an effective and reliable Web Service discovery. Here reducing the service discovery time and increasing the quality of discovery are key issues. This paper proposes a new semantic Web Service discovery scheme where the similarity between the query and service is decided using the WSDL specification and ontology, and the improved Hungarian algorithm is applied to quickly find the maximum match. The proposed approach utilizes the structure of datatype and operation, and natural language description used for information retrieval. Computer simulation reveals that the proposed scheme substantially increases the quality of service discovery compared to the existing schemes in terms of precision, recall rate, and F-measure. Moreover, the proposed scheme allows consistently smaller discovery time, while the improvement gets more significant as the number of compared parameters increases.

Keywords: web service, service discovery, matching, similarity matrix, Hungarian algorithm, bipartite graph

1. INTRODUCTION

Web Service is a programmatic interface for the applications (i.e., business logic) available over the WWW infrastructure, and XML is used to encode the communications to the Web service. Web service enables the interoperability between heterogeneous systems and the reuse of the functions in the application development of distributed system [26]. It is a new branch of self-contained, self-describing, modular Web applications [4]. Universal Description, Discovery and Integration (UDDI) registry, which is a collection of all registered Web services available on the Internet, allows the discovery of Web service providers. A Web Service Description Language (WSDL) [7] document is attached to each Web service, which contains the information on the Web Service and how to access it using XML tags.

Currently, there exist mainly two kinds of strategies for Web Service discovery which are classified depending on the type of Web Service description [1]: (i) a particular Web Service is searched based on the keywords or predefined taxonomies such as UDDI, (ii) the semantic similarity is calculated when Web Services are described in se-
mantic language. Moving beyond the syntax opens the door to more advanced applications and functionalities on the web, and computers will be able to search, process, integrate, and present the content of these resources in a meaningful manner. Service designers describe the services using natural language, e.g. WSDL descriptions, which is often too imprecise and inadequate. Hence, the essential life cycle of a service such as discovery, execution, and composition requires manual effort, and the operations are at best semi-automated. Among them, the discovery operation is crucial because user cannot use a service unless the user is aware of its existence or discovers it. Automatic Web Service discovery involves automatically locating a set of appropriate Web Services meeting the user requirements. Service discovery consists of three interrelated phases: matching, assessment, and selection. During the first phase, the description of a service is matched to that of a set of available resources. Next, the result of matching (typically a set of ranked Web Services) is assessed and filtered based on a set of criteria. Finally, services are selected which are subsequently customized and combined with others [6].

The UDDI allows only text-based keyword search which leads to the lack of necessary semantic support and description of the relationship between different services. It thus cannot provide the service of high precision [11].

With the development of semantic web, ontology is used with data exchange between heterogeneous data sources in semantic web as a conceptual model, and it is persuaded by many researchers. OWL-S is a specific OWL ontology designed to provide a framework for semantically describing the services from several perspectives (e.g., discovery, invocation, composition). However, this has serious limitations due to natural language descriptions. Current approaches for describing services on the semantic web (e.g., OWL-S) do not support for establishing semantic correspondences between the ontologies. Semantically enhancing Web Service descriptions with ontologies help overcome this drawback. Lately, new frameworks such as WSDL-S have been proposed to provide the support for inter-ontology translation [29]. The current set of Web Service specifications defines how to specify reusable operations through WSDL, how these operations can be discovered and reused through the UDDI API, and how the requests to and responses from Web Service operations can be transmitted through the Simple Object Access Protocol (SOAP) API [28]. The WSDL developed independently of OWL-S, provides a well-organized means of specifying these kinds of details, and has already acquired considerable visibility within the commercial Web Service community. Therefore, OWL-S has chosen to define the conventions using WSDL to ground the OWL-S services. These conventions are based on the observation that the concept of grounding of OWL-S is generally consistent with the concept of binding of WSDL. As a result, it is a relatively straightforward task to ground the OWL-S atomic process [30].

Service providers advertise a service by adding the WSDL specification to the appropriate UDDI directory category. Service-discovery based on only the category is insufficient. Analyzing the existing service discovery methods, most of them are found to consider only the semantic information of elements in the ontology and ignore the structure of the ontology. Recently, there has been a significant increase of research on semantic similarity. However, its stationary discovery model has a shortcoming of limited adaptability. The number of Web Services is growing day by day at an explosive speed, which brings great challenges to accurate, efficient, and automatic retrieval of target services for the users [2]. One of the approaches for making the execution of a process effi-
cient is to simplify the structure of the entire system. The service consumer can be not only a user but also another service or program [7]. Therefore, use of automated mechanism of discovering the service is very important.

In this paper, thus, a combination of similarity-based method with WSDL specification and ontology are proposed that can be used to effectively support a more automated service-discovery process. Here the similarity between the requesting service and web services published in UDDI repository is calculated. The semantic Web Service discovery operation is modeled as an assignment problem with bipartite graph, and the improved Hungarian algorithm [13] is applied. The quality of Web Service is usually measured in terms of throughput and latency. High throughput and low latency indicates high quality Web Service. The proposed scheme takes less time than the existing scheme in finding the maximum matching. The proposed scheme consists of two steps. In the first step, the similarity between a desired service and a set of advertised services are measured. Combining the similarity of textual description, semantic structure and concept, the overall scores of similarity are calculated. In the second step, the matching for semantic Web Service is modeled using bipartite graph of the nodes defined based on the ontology. The scheme is evaluated in terms of speed and accuracy represented by precision, recall rate, and F-measure. Computer simulation reveals that the proposed scheme is better than the existing scheme employing the Hungarian algorithm. The scalability is also much higher than the existing scheme.

The rest of the paper is organized as follows: Section 2 addresses the related work including the state-of-the-art discovery methods for Web Service. The proposed scheme for Web Service discovery is presented in Section 3. Section 4 describes the experimental results, and finally Section 5 gives some discussions and remarks to be investigated in the future.

2. BACKGROUND

The structure of Web Service is based upon the interactions between three roles: service provider, service registry and service requestor. The interactions between them involve publish, find and bind operation. Together, these roles and operations act upon the Web Service artifacts: the Web Service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web Service). The service provider defines a service description for the Web Service and publishes it to a service requestor or service registry. The service requestor uses ‘find’ operation to retrieve the service description locally or from the service registry. It uses the service description to bind with the service provider, and invokes or interacts with the Web Service implementation. The role of service provider and service requestor is logical construct, and a service can exhibit both characteristics. Fig. 1 illustrates these operations, along with the components providing them and the interactions [10].

2.1 WSDL Specification

A WSDL specification usually includes a set of natural-language descriptions of the
service and its elements. It has the following constructs to represent service descriptions: interface, operation, message, binding, service and endpoint. Among them the first three constructs deal with abstract definition of the service, while the remaining three constructs are for service implementation.

2.2 Ontology

Ontology is comprised of concepts, concept properties, relationships between concepts and constraints. Ontologies are defined independently from actual data and reflect a common understanding of the semantics of the domain of discourse. It is an explicit specification of a representational vocabulary for a domain; definitions of classes, relations, functions, constraints and other objects. OWL has evolved as a standard for the representation of ontologies on the Web. Both advertisements and search queries are expressed in terms of OWL-S descriptions. The OWL-S Service Profile defines a service in terms of Inputs, Outputs, Pre-conditions and Effects (IOPE). In OWL-S, three essential types of knowledge for a service are provided, which are Service Profile, Service Model, and Service Grounding. They cover the description, functionality, and access mechanism, respectively [9].

- **Service Profile**: Service profile facilitates service provider to describe its service. The information provided by Service Profile can be categorized as:
  - Provider’s information: This includes the name of the provider and contact details.
  - Functional description: Inputs, outputs, preconditions, and effects are described here.
  - Profile attributes: The parameters specifying the service, e.g. QoS, service.
- **Service Model**: This describes the service as a process, either atomic or composite: receives and sends a single message or retains/changes the state through a sequence of messages. A service produces some outputs and sets the conditions, and thus changes the surrounding world. A model includes:
  - Input and output parameters: Expressed as a subclass of the parameter in OWL-S.
  - Preconditions and effects: Modeled as logical formulas or expressions treated as either string literals or XML literals depending on the language used.
- **Service Grounding**: It provides concrete details required to invoke the service such as message format, transfer protocol, etc. OWL-S uses WSDL standard for service groun-
AN EFFICIENT WEB SERVICE DISCOVERY SCHEME

157

ding, which provides a wrapper and carries OWLS message on standard networking protocol. WSDL cannot capture the semantic of message, while OWL-S alone is not capable of dealing with the standard transfer protocol. Both the languages overlap in the description of a message at abstract level, and mapping from OWL-S to WSDL is done in three steps:

- **Step 1**: An OWL-S atomic process is mapped to the corresponding WSDL operation.
- **Step 2**: The inputs and outputs of OWL-S process are mapped to the corresponding inputs and outputs of WSDL message, respectively.
- **Step 3**: The inputs and outputs of OWL-S process are mapped to the corresponding abstract type of WSDL.

2.3 Wordnet

WordNet is an online lexical database system developed at Princeton University. In WordNet Nouns, verbs, adverbs and adjectives are organized by a variety of semantic relations into synonym sets (synsets), which represent one concept. Examples of semantic relations used by WordNet are synonymy, autonomy, hyponymy, member, similar, domain, and cause and so on. Relationship between the concepts such as hyponyms (i.e. more specific terms) is represented as semantic pointer linking the related concepts.

2.3 Web Service Discovery

Web Service discovery is the process of finding an appropriate service provider for a service requestor through a service matchmaker. Natural language description of desired Web Services and WSDL specifications of all available services published through UDDI are given to discover a service. Here words should be extracted from WSDL, which are pre-processed and assigned the weight. According to the weights, the similarity between the given description and a description of web-service operation can be measured. The weighted words are used to build a matrix containing the information on all Web Services which have common words in their description.

3. RELATED WORK

The discovery is based on a matchmaking algorithm, which allows the finding of the Web Service descriptions having a semantic correspondence between the functional parameters defined in the descriptions and those introduced in the search query. The process of service match is to extract the inputs and outputs of the advertisements, and then match them with those of the search query. There exist several proposals increasing the accuracy and efficiency of service discovery. The semantic correspondence between two concepts is based on the relationship between the ones in their respective OWL ontologies. The advertisements and search queries are both expressed in OWL-S [11]. The algorithm identifies four levels of semantic correspondence between any two concepts, namely: Exact, Plug in, Subsume, and Disjoint [16]. Web Services are classified by the level of semantic correspondence between the output parameters of the description and those cited in the query. A noticeable trend in Web Service matchmaking is related to
input-output matching, where a service request is regarded to match a Web Service if the inputs and outputs are identical.

Recently, several approaches have been proposed to find the similarity between user query and advertisement for the OWLS [31-32] or WSDL [33] based services. A noticeable direction of the current research on Web Services is semantic Web Service such as OWL-S [19], WSDL-S [20], and SWSO [21]. The initial aim of semantic Web Service is to complement the Web Service beyond WSDL. The researchers have proposed Web Service discovery framework using various ontologies [34-37] to use semantics in WSDL. The ontology-based modeling renders semantic models to be conceptual framework for the semantic description of Web Services, in which the ontologies are regarded as the semantic annotations [22]. In fact, several matchmaking algorithms rely only on the matching of Inputs and Outputs of Service Profiles. The result of matchmaking operation can be judged by the degree of semantic similarity offered by the algorithm as proposed in [14]. An optimal match based on the semantic similarity of Web Service can be found using bipartite graph, where an assignment algorithm needs to be applied. The Hungarian algorithm [15] is a classic assignment algorithm applied to get an optimal match with bipartite graph. Several semantic Web Service matchmaking schemes have been proposed [1-3, 5, 8, 12, 17, 18]. The problem of service matching and discovery is analogous to the problem of information retrieval and component retrieval for which the performance is measured in terms of throughput and latency.

Using bipartite graph, each parameter of the requestor can be assigned to a parameter of the advertisement to find a perfect matching. This type of matching is efficient and flexible in terms of throughput and latency.

4. THE PROPOSED SCHEME

In this paper we propose a Web Service discovery method using bipartite graph to find the most relevant Web Services to the user query. At first, the method of computing the semantic similarity with a Web Service is proposed, and then a matchmaking algorithm based on the notion of matching bipartite graph is introduced.

4.1 Semantic Similarity

The approach for computing the similarity between the request and service needs to divide the ontology information into two types: lexical information attached to each entity (e.g., label, comment and the instance data-type property) and structural information hidden in the links between the entities such as ‘is-a’ relation between the concepts and properties, and instances of object-type properties.

4.1.1 Textual similarity

First the similarities based on WSDL textual descriptions are found for their services, types and operations, grouped under <documentation> tags. The query and service descriptions are elaborated with semantic information regarding the synonyms and homonyms with WordNet.
a) **Lexical Similarity**: Several lexical similarity metrics can be used to compare the entities. Here vector-based similarity is used. The vector-based similarity is calculated between the feature vectors of ontology entities. For a word \( i \) in an entity’s virtual document \( j \), the weight of the word is computed as

\[
W_q = tf_i \cdot \log \frac{N}{df_i}
\]  

(1)

The lexical similarity between two words is defined as follows:

\[
Sim^l(e, e') = K_0 \frac{\sum_{i=1}^{M} w_i \cdot w_{i'}^2}{\sqrt{\sum_{i=1}^{M} w_i^2} \cdot \sqrt{\sum_{i=1}^{M} w_{i'}^2}}
\]  

(2)

b) **Semantic Similarity**: Semantic similarity between two words of synonyms needs to be decided when both words do not belong to the same synset in wordnet but the corresponding sense of them can be found in wordnet. The semantic similarity between two words is defined as a distance function from individual to lowest common ancestor (LCA). In addition, if one of the two words does not appear in wordnet, the string-based similarity is used to compute the semantic similarity. The semantic similarity between two concepts, \((e, e')\), in synonyms level and string-based similarity level [25] is as follows.

\[
Sim^s(e, e') = K \frac{2 \cdot \text{depth}(\text{LCA}(e, e'))}{\text{length}(e, e') + 2 \cdot \text{depth}(\text{LCA}(e, e')) + (1 - K)} \frac{\text{length}(\text{LCA}(e, e'))^2}{\text{length}(e) \times \text{length}(e')}
\]  

(3)

Here \( \text{depth}(e) \) is the length of the path to synset \( e \) from the global root entity, and \( K \) represents adjustment factor.

Now the combined lexical similarity and semantic similarity of two concepts, \( Sim_{LS}(e, e') \), is defined below:

\[
Sim_{LS}(e, e') = a \cdot Sim^l(e, e') + (1 - a) \cdot Sim^s(e, e'),
\]  

(4)

where \( a (\leq 1) \) is weight factor. A higher overall matching score indicates higher similarity between the source and target specification.

After calculating textual similarity, a similarity matrix representing similarity score of all pair-wise combination of source and target element needs to be generated. Then the maximum similarity is found as follows.

\[
Sim_{TS}(e, e') = \max(Sim_{LS}(e, e')) \forall i, j
\]  

(5)

### 4.1.2 Structural similarity

The structural matching is extended by taking the identifiers of the service elements into consideration, in addition to the type of programming-language and syntactic rela-
tion. It involves the comparison of the operation set offered by the services, which is based on the comparison of the structures of the input and output message of the operation. It, in turn, is based on the comparison of the data types of the messages.

a) **Data type similarity**: Here the data types involved in the two WSDL specifications are compared. To assess the degree of similarity between two service data types, \((T_1, T_2)\), the summary on the XML Schema data type is used. Table 1 presents the data type similarity, \(DtSim(T_1, T_2)\), obtained for six typical attribute types in the XML Schema.

After calculating the datatype similarity, the maximum matching needs to be found by constructing the matrix representing the similarity score of all pair-wise combination of source and target elements, i.e. message and operation.

<table>
<thead>
<tr>
<th>Table 1. Data type similarity for XML schema.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
</tr>
<tr>
<td>date</td>
</tr>
<tr>
<td>Decimal</td>
</tr>
<tr>
<td>Integer</td>
</tr>
<tr>
<td>Float</td>
</tr>
<tr>
<td>Lang</td>
</tr>
</tbody>
</table>

b) **Name similarity**: To compute the name similarity between two names represented as strings, first, each string is broken into a set of tokens \(T_1\) and \(T_2\) using a customizable tokenizer with punctuation, upper case, special symbols, and digits, e.g. `getDataService` → \{get, Data, Service\}. Name similarity defines similarity (relatedness) between the two tokens.

Then synthetic similarity between the two sets of name tokens \(T_1\) and \(T_2\) is determined as the average of best similarity of each token with a token in the other set [24]. It is computed as follows:

\[
NSynSim(T_1, T_2) = \frac{\sum_{i \in T_1} \max_{t_i \in T_2} Sim(t_i, t) + \sum_{j \in T_2} \max_{t_j \in T_1} Sim(t, t_j)}{T_1 + T_2}.
\]  

(6)

The structural semantic similarity of the names using wordnet is measured by the following equation.

\[
SemSim(T_1, T_2) = \frac{2 \cdot depth(LCA(T_1, T_2))}{length(T_1, T_2) + 2 \cdot depth(LCA(T_1, T_2))}
\]  

(7)

Name similarity is then measured as

\[
NSim(T_1, T_2) = k \cdot NSynSim(T_1, T_2) + (k - 1) \cdot NSemSim(T_1, T_2).
\]  

(8)

Here \(k\) is adjustment factor. The semantic similarity of WSDL specification, i.e. message of services, operations, is given below.
AN EFFICIENT WEB SERVICE DISCOVERY SCHEME

\( Sim_{nm}(T_1, T_2) = b \cdot NSim(T_1, T_2) + (1-b) \cdot Dtsim(T_1, T_2) \)

(9)

Here \( b (\leq 1) \) is weight factor of name similarity.

Now, a matrix representing the similarity score of all pair-wise combination of source and target is needed to be contracted. And then the maximum matching of similarity is found.

\[ Sim_{nm}(T_1, T_2) = \max (Sim_{nm}(T_1, T_2)) \forall i,j \]

(10)

The structural similarity of services with the messages, operations, and port is calculated respectively from the equation above.

By combining the message, operation, and port similarities, \( Sim_{s}(e_1, e_2) \), between the web services of WSDL is obtained as follows.

\[ Sim_s(e_1, e_2) = c \cdot Sim_{msgSim}(e_1, e_2) + d \cdot Sim_{opSim}(e_1, e_2) + (1-c-d) \cdot Sim_{portSim}(e_1, e_2) \]

(11)

where \( c \) and \( d \) are weight factor of ‘Message’ and ‘Operation’ similarity, respectively, for the services, and \((c + d) \leq 1\).

4.1.3 Concept semantic similarity

The next stage is called concept semantic matching. Ontologies are organized as concept hierarchies. In order to utilize the similarity of a concept hierarchy, the similarity measure for concept hierarchy is introduced, which is calculated by the path from the root to the concept node. It matches the schema elements based on the similarity of their context (position) and the nearest elements.

- **Similarity based on the distance between the terms**: Given a pair of two terms, \( e \) and \( e' \), a well-known method with intuitive explicitness for assessing their similarity is to calculate the distance between the nodes corresponding to the terms in the ontology hierarchy; the shorter the distance, the higher the similarity [23]. The formula calculating the similarity between \( e \) and \( e' \) is denoted as

\[
Sim_{Dist}(e, e') = e^{-\alpha \cdot \text{depth}(\text{iso}(e, e')) + \beta \cdot \text{depth}(\text{iso}(e, e'))} - e^{-\alpha \cdot \text{depth}(\text{iso}(e, e')) + \beta \cdot \text{depth}(\text{iso}(e, e'))} + e^{-\alpha \cdot \text{depth}(\text{iso}(e, e')) + \beta \cdot \text{depth}(\text{iso}(e, e'))}.
\]

(12)

where \( \text{iso} \) stands for the shortest path between the two concepts in the ontology, and \( \text{depth}(\text{iso}(e, e')) \) stands for the depth of the closest common ancestor of the two concepts. Here \( \alpha (\geq 0) \) and \( \beta (\geq 0) \) are the parameters scaling the contribution of the shortest path between the two concepts and the depth of closest common ancestor in the concept hierarchy, respectively.

- **Subsumption similarity**: During the matching, the advertisement and request of service are compared with each other. The comparison is made according to the concepts (classes of the arguments). If two concepts are stated to be complement or disjoint, zero score is assigned. Otherwise, the subsumption relation is checked and a score in the \([0, 1]\) range is assigned as described in Table 2. Four (ranked) degrees of match between outQ
and \(outA\) are used, corresponding to the outputs of service request and advertisement, respectively. If there are several outputs with different degree of match, the minimum degree is taken. The same approach is employed to compute the matching between the inputs, with the order of request and advertisement reversed.

<table>
<thead>
<tr>
<th>Condition</th>
<th>match(outQ, outA)</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>outA Equivalent to outQ</td>
<td>Exact</td>
<td>1</td>
</tr>
<tr>
<td>outASuperClass of outQ</td>
<td>Exact</td>
<td>1</td>
</tr>
<tr>
<td>outA Subsumes outQ</td>
<td>Plugin</td>
<td>.7</td>
</tr>
<tr>
<td>outQ Subsumes outA</td>
<td>Subsume</td>
<td>.3</td>
</tr>
<tr>
<td>None of the above</td>
<td>Fail</td>
<td>0</td>
</tr>
</tbody>
</table>

Finally, the set of service advertisements is sorted by comparing output matches first, if equally scored, considering the input matches. Applying the concept similarity, the services are sorted according to the degree of match as Exact > Plug in > Subsumes > Fail, where \(x > y\) indicates that \(x\) is ranked higher (is a more desirable match) than \(y\).

Now the concept similarity based on the ontology is found which is defined as follows.

\[
Sim_{\text{con}}(e_1, e_2) = f \times Sim_{\text{dist}}(e_1, e_2) + (1 - f) \times Sim_{\text{sub}}(e_1, e_2)
\]

where \(Sim_{\text{dist}}\) and \(Sim_{\text{sub}}\) denote the similarity based on the distance and subsumption, respectively, \(f (\leq 1)\) is the weight factor of distance similarity.

After calculating the semantic similarity between the request and advertisement of service, a matrix can be constructed with the results of this step which represents the matching scores of all pair-wise combinations of request and advertisement of the service. The parameter values are decided on the basis of how similar their parameter lists are, in terms of the operation type and organization.

Now, the maximum similarity between \(e_1\) and \(e_2\) is found.

\[
Sim_{\text{con}}(e_1, e_2) = \max(Sim_{\text{con}}(e_1, e_j)) \forall i,j
\]

4.1.4 Combined similarity

Here we propose to combine the similarity scores based on the ontology descriptions and ontology structure for reaching to a better similarity measure. The compound similarity between two services, \(Sim(S_1, S_2)\), is calculated as follows:

\[
Sim(S_1, S_2) = g \times Sim_{\text{TS}}(S_1, S_2) + h \times Sim_{\text{S}}(S_1, S_2) + (1 - g - h) \times Sim_{\text{con}}(S_1, S_2)
\]

where \(g\) and \(h\) \((g + h \leq 1)\) are weight factor of textual similarity and structural similarity, respectively.

4.2 Web Service Matching

For Web Service matching, bipartite graph matching is employed. First, the request
and advertisements are loaded into memory. Then for each advertisement, the inputs are compared with those of the request. It uses the module calculating the similarity of each parameter pair. The semantic similarity is calculated with the Web Service specification based on the ontology structure discussed in Section 3.1. The scores between the request and advertisement pair are used as the weights of the bipartite graph which is adopted for finding the match between the pairs. Here the service and request parameters are represented as vertices of two disjoint sets of the graph. Each vertex of a set is connected to every other vertex in the other set with a weight assigned to the edge.

The matching problem can be transformed to finding a subgraph, where each vertex is used only once and the sum of weights on the edges of subgraph is maximum. This problem is then called maximum weight bipartite matching. The services requiring more inputs than the request are first omitted since they cannot operate with insufficient number of inputs. The scoring is obtained by calculating the maximum cardinality with maximum weight matching of the bipartite graph. This score becomes the input score of the advertisement, and it is kept in the input score list. Output scoring is done similarly, but this time the services having smaller number of outputs than the request are omitted. At last, the ranked set of services is generated as the output.

In this section the proposed matchmaking algorithm is presented which maximizes the matching rate by using bipartite graph and minimizes the time complexity using improved Hungarian algorithm. We first discuss the basic concept of bipartite graph and then introduce the proposed algorithm.

### 4.2.1 Bipartite matching

**Bipartite Graph:** A Bipartite Graph is a graph $G = (V, E)$ in which the set of vertices are partitioned into two disjoint sets, $V_0$ and $V_1$ ($|V_0| \leq |V_1|$), such that every edge $e \in E$ has one vertex in $V_0$ and another in $V_1$. Fig. 2 shows an example of weighted bipartite graph with a weight assigned to each edge. Here $r_i$ and $a_i$ represent the parameters of service request and advertisement, respectively.

**Matching:** A matching of a bipartite graph $G = (V, E)$ is subgraph $G' = (V', E') \subseteq E$, such that no two edges $e_1, e_2 \in E'$ share the same vertex. Vertex $v$ is said to be matched if it is incident to an edge in the matching. For a bipartite graph $G$ and its matching $G'$, the matching is complete if and only if all vertices in $V_0$ are matched. The best matching is the one whose sum of the weights is maximum. Fig. 3 shows the best matching graph for Fig. 2.

Fig. 2. An example of weighted bipartite graph.  
Fig. 3. An example of best matching $G'$ of Fig. 2.
A Web Service has typically two sets of parameters, input parameters and output parameters. Let $Out_r(r_1, r_2, \ldots, r_n)$ denote $n$ output parameters of service request, $WS_r$, and $Out_a(a_1, a_2, \ldots, a_m)$ does output parameters of service advertisement, $WS_a$, respectively [4]. Fig. 4 shows the relationship between the set of inputs or outputs in the advertisement and request for an example of hospital service. In Fig. 4, $r_i$ and $a_i$ denote the input parameter of service request and advertisement, respectively. Similarly, $ro_i$ and $ao_i$ do the output parameters. The arrows indicate the relationship between the input and output of request and advertisement.

**4.2.2 Semantic matchmaking**

Mapping the problem of matching of the outputs of a service request and service advertisement into the problem of matching over a bipartite graph involves two steps:

**Step 1:** Constructing a bipartite graph. The Web Service matching problem, either input matching or output matching, could be considered as the problem of optimal matching with bipartite graph. An advertisement is said to match a request when all the outputs of the request match the outputs of the advertisement and all the inputs of the advertisement match the inputs of the request. Let $Out_A$ and $Out_R$ represent an output of the advertisement and request, respectively. A graph $G = (OutR \cup uA, E)$ is constructed in this step.

**Step 2:** Semantic matchmaking. Let $OutQ$ represent an element of the list of outputs of the Query. In case of output matching, the ‘match ($OutQ, OutA$)’ function accepts $OutQ$ and $OutA$ as inputs and returns the degree of match between them [14]. Recall that four degrees of match are defined in Table 2.

The Hungarian algorithm can be used to obtain a completely matched bipartite graph. In the example of Fig. 5 (a), the $SPP$ matrix is obtained; capturing the semantic similarity between all pairs of attributes. For example, assume that there are five output parameters for service advertisement, $Out_a(a_1, a_2, a_3, a_4, a_5)$, and five parameters for service request, $Out_r(r_1, r_2, r_3, r_4, r_5)$, of Web Service. Then a 5×5 similarity matrix is given below in Fig. 5 (b).
4.3 Hungarian Algorithm

Hungarian algorithm is a classic algorithm applied to obtain an initial optimal match with bipartite graph. In the original Hungarian algorithm the scanned rows of the similarity matrix are initially unassigned. The updating operation is expensive, and thus it is desired to be avoided. With a large initial set to scan, one first finds all currently available zero-cost augmenting paths.

The Hungarian algorithm can also be used for a maximization problem for which the matrix first needs to be transformed. For example, the inputs of request need to match the inputs of advertised service. A request is more or less similar to a service, and the closeness can be defined as similarity. The higher the score, the closer the similarity. A higher similarity score indicates a closer similarity between the request and service specification. In Fig. 5 (b), the maximal similarity of the matrix is 1. If the value of each element of the matrix is subtracted from 1, we have the matrix of Fig. 6 (a). Now the matching can be applied which minimizes the amount of similarity. Fig. 6 shows an example of the process of Hungarian algorithm, which consists of three steps:
**Step H.1:** Reduce the rows (columns) by subtracting the minimum value of each row (column) from that row (column).

**Step H.2:** Cover the zero elements with the minimum number of lines it is possible to cover. If the number of lines covering the zero elements is not equal to the number of rows, return to Step H.3. Otherwise, terminate the process.

**Step H.3:** Determine the minimum of the uncovered elements, which is subtracted from all the elements, not crossed out and added to the element crossed out twice. Return to Step H.2.

In Step H.1 of Iteration 1, the matrix is same as the original matrix because the minimum value is zero. With the final solution, the maximal similarity can be computed as in Fig. 6 (g). Observe that the element with ‘*’ symbol represents the matched element for each request:

\[ r_1 \Rightarrow a_4 \text{ (the weight of this matching of Fig. 5 (b) is .25), } r_2 \Rightarrow a_3(1), r_3 \Rightarrow a_1(1), r_4 \Rightarrow a_3(.5), r_5 \Rightarrow a_4(.8) \]

The sum of the weights is 3.55, which is maximum among all the complete matching.

### 4.4 Improved Hungarian Algorithm

With efficient assignment of the objects, the time complexity of Hungarian algorithm can be reduced [13]. First, the rows and columns are scanned separately to cover the zero elements corresponding to Step H.1 of the original algorithm. The decision on whether checking the column or not is made while the rows are scanned.

Next, in Step H.2, the zero elements are covered with the minimum number of lines required to cover them, and the unassigned rows except a one are crossed out. The step is repeated until no change occurs. With Step H.3 the minimum element is determined from the elements not crossed out yet in all the assigned rows and only one of the unassigned rows which has not been crossed out.

![Fig. 7. The process of improved matching algorithm for the example of Fig. 6.](image)
Fig. 7 is the process of improved Hungarian algorithm for the example of Fig. 6. Note that the matrix is same as that of Fig. 6 (b) after covering the zero elements.

The improved approach considers the minimum value not crossed out yet from all assigned rows and only one of the unassigned rows, say row 1. Therefore, besides column 1 and row 2, row 4 and row 5 are crossed out. The minimum value is subtracted from all the elements not crossed out, and added to every element crossed out twice. The process repeats until all rows are assigned. In the improved algorithm the row or column cannot be reduced anymore if the element of zero value does not exist. The process is repeated until all rows are assigned. As another improvement the dual variables are updated which waste the space.

4.5 Procedure of Improved Hungarian Algorithm.

**Step 0:** Initialize.

**Step 1:** and Step 2: Determine the shortest augmenting path.

1.1: If all rows are assigned, go to Step 4.

1.2: Select any unassigned row \( i^* \) and label it; set \( d[j] = \infty \) (\( j = 1, \ldots, n \)). \( d_{\min} := 0 \).

1.3: Select an unscanned labeled row \( i \), and scan it: for all \( j \in A[i] \) with \( d_{\min} + cred[i, j] < d[j] \):

1.3.1: \( \begin{align*} d[j] &= d_{\min} + cred[i, j], \\
& \text{pred}[j] := i. \end{align*} \)

1.3.2: If \( cred[i, j] = 0 \): if \( j \) is assigned, then label row \( y[j] \), else go to Step 3.

1.4: If all labeled rows are scanned go to Step 2, else go to Step 1.3.

2.1: Determine \( d_m = \min\{d[j] \mid j = 1, \ldots, n; d[j] > d_{\min}\}; d_{\min} := d_m \).

2.2: For every column \( j \) with \( d[j] = d_{\min} \): if \( j \) is assigned, then label row \( y[j] \), else go to Step 2.

**Step 3:** Update and augment.

3.0: \( u[i^*] := u[i^*] + d_{\min} \) for every column \( k \) with \( d[k] < d_{\min} \):

\[
\begin{align*}
\gamma[k] &= \gamma[k] + d[k] - d_{\min}, \\
\iota := y[k], \quad u[i] := u[i] - d[k] + d_{\min}.
\end{align*}
\]

3.1: Augment the solution along the alternating path from column \( j \) to row \( i^* \).

3.2: Remove all labels and return to Step 1.

Using the matching algorithm above the best match can be achieved efficiently. This paper adopts the improved Hungarian algorithm to further reduce the time complexity of matching operation.

4.6 The Entire Model of the Proposed Scheme

The entire model of the proposed scheme for Web Service discovery is illustrated in Fig. 8. This model finds the similarity between the request and advertisement parameters of Web Service. For this, the textual, structural, and concept similarity need to be decided. First, textual description parameters of service are used for deciding the textual similarity, where lexical and semantic similarities are computed using Eqs. (2) and (3), respectively. Textual similarity is calculated by combining lexical and semantic similarity using Eq. (4) as discussed in Section 4.1.1. After calculating the textual similarity, the best similarity is found by applying the similarity score to the bipartite graph and improved Hun-
Fig. 8. The model of the proposed scheme of Web Service discovery.

garian algorithm as discussed in Section 4.2. Also, the operation, message, and port type parameter are used for deciding the structural similarity. For each parameter, data type and name similarity are calculated. Name similarity is computed by combining syntactic and semantic similarity, which are calculated using Eqs. (6) and (7) as discussed in Section 4.1.2. After that, the best similarity is found by bipartite and improved Hungarian algorithm as discussed in Section 4.2. Datatype similarity is found using Table 1. After calculating the structural similarity of operation, messages, port parameter, total structural similarity is found by combining them. To find the concept similarity, distance and subsumption similarity are computed using Eq. (12) and Table 2, respectively. Concept similarity is calculated by combining distance and subsumption similarity using Eq. (13) as discussed in Section 4.1.3. And then the best similarity is found by combining textual, structural, and concept similarity. The similarity between the request and advertisement is found using Eq. (15). The label from ‘a’ to ‘h’ denote the weight factor for different type of similarity.
5. PERFORMANCE EVALUATION

5.1 Dataset

A dataset of related WSDL specifications is used to evaluate the effectiveness of the proposed Web service matchmaking method. It contains OWL-S service retrieval test collection from medical care, travel, education domain in OWLS-TC v24 and weather and hotel service from X-method used for search whose WSDL speciation are available. Textual descriptions and XML schemas of input/output data types can be obtained from this dataset. The employed dataset represent a variety of Web services such as book search, hospital, weather, travel, and Hotel. Some services are semantic dependent, while the others are syntactic dependent. Since the five Web services of different domains have diverse characteristics in terms of the service access patterns, they will be comprehensive enough for validating the effectiveness of the proposed scheme. In this collection total 596 service descriptions are identified from the five categories. Among them 286 services are for book search, 197 for travel, 8 for weather, 75 for hospital, and 20 for hotel search, respectively. Each of the services was used as the basis for the desired service; different aspects of the desired service were matched then against the complete set to identify the best target service.

5.2 Experimental Results

The performance of the proposed scheme is evaluated in two steps. First, the maximum matching algorithm identifying the best values for the parameters of Web Services is investigated. Then the effectiveness of discovery with the proposed scheme is compared with the existing schemes.

To validate the efficiency of the proposed scheme, the response time, precision of mapping, $P$, recall rate, $R$, and F-measure are considered. Observe from the equation that the precision indicates the degree of correct discovery, while recall rate does the degree of reused correct discovery. Note that F-measure is a combination of precision and recall rate. The values of the weighting factor in each similarity equation are assumed to be same, while the sum of them is equal to one.

\[
P = \frac{|\text{relevant service} \cap \text{retrieved service}|}{|\text{retrieved service}|} \quad (16)
\]

\[
R = \frac{|\text{relevant service} \cap \text{retrieved service}|}{|\text{relevant service}|} \quad (17)
\]

\[
F\text{-measure} = \frac{2 \times P \times R}{P + R} \quad (18)
\]

At first, the similarity between the service requirement and advertisement are generated. Then the proposed algorithm is executed to find the maximum match and the computation time is measured. 100 sets of the parameters are used to test the proposed algorithm. It is also compared with that of the original Hungarian algorithm. The proposed algorithm and original Hungarian algorithm are tested for $3 \times 3$ to $8 \times 8$ matrices. The results of 10 matching problems of different similarities are averaged for each
square matrix. Fig. 9 shows the results of the response time of the two algorithms. Observe from the graph that the proposed algorithm is consistently faster than the original Hungarian algorithm. Moreover, it illustrates that the proposed scheme is less sensitive to the size of the similarity matrix than Hungarian algorithm. The proposed scheme is obviously quite scalable, and it can thus be applied to a large scale Web Service matching problem.

The proposed matching algorithm is also compared with a Web Service discovery scheme [27], which is based on only ontology concept. In Figs. 10 and 11, the five marks on the x-axis stand for five different Web services, travelling, book search, Hospital, Hotel, and Weather. From the figures it is clear that the proposed algorithm allows much higher precision and recall rate than the existing scheme.

The performances of the algorithms with different similarity are evaluated for the OWL-S with WSDL service retrieval. Here five different Web Services are queried with textual similarity and concept similarity. From Fig. 12 of F-measures, we can see that the ontology similarity allows better efficiency than textual similarity. It turns that the similarity of ontological concept is more important than textual matching for service discovery.
Four different service discovery schemes are tested which are textual matching, structural matching, concept-based matching, and the combination of them. The results from the five service categories are averaged for the four approaches, and they are shown in Fig. 13. Observe that the precision and F-measure of combination matching are better than other schemes. Interestingly, recall rate of the textual matching is comparable with the combination matching.

Table 3 shows the characteristics of different web service. The next step is to evaluate the proposed scheme with different weight. In Table 4, different weights are assigned to similarity method by giving the priority to concept, semantic, name matching according to the importance of the parameter. Table 5 shows weighting scores for different method of similarity for our model. As it is clear from Fig. 12 that ontology matching gives more accuracy than textual, larger weight is assigned to ontology matching than textual matching. Also notice that textual matching, structural, and concept-based matching increasingly allows higher accuracy. Therefore, largest weight is assigned to concept-based matching, then structural matching and so on.

Now, the performances of the proposed scheme different weight are evaluated using Table 4. In Figs. 14 and 15, the proposed scheme with same and different weight, and the precious scheme are compared in terms of precision and recall rate, respectively. Observe from the figure that Traveling, Hotel, and Weather service are sensitive to the weight, while Book Search and Hospital service are not as
In the proposed scheme the textual matching and ontology matching are tested again with different weight and compared with other scheme. Fig. 16 shows that textual matching slightly decreases while ontology matching increases with the changed weight. The four methods of textual similarity, structural similarity, concept-based similarity, and combined method are also evaluated with different weight. In Fig. 17, we can see that the overall similarity increases except textual matching.

### 5.3 Weight Adjustment

The weights of the parameters of the similarity are adjusted as shown in Table 5, which were decided considering the importance of similarity parameter based on the type of Web service. For calculating the combined similarity, three weight factors are used for concept \((1-g-h)\), structural \((h)\), and textual similarity \((g)\). As can be seen from Fig. 14, concept similarity displays more importance than structural similarity, which it is more important than textual similarity. Therefore, in the combined similarity, the highest weight is assigned to concept, medium to structural \((h)\), and the smallest one to textual \((g)\) similarity, respectively. In textual similarity, both lexical similarity and name similarity are important. Observe from Figs. 15 and 16 that Traveling, Hotel, and Weather service are sensitive to the weights which are lexical or syntactic dependent, while Book Search and Hospital service are not as they are semantic dependent. This is because, in Table 3, the weight of lexical and syntactic factor is large and that of semantic factor is small. Depending on the importance of the similarity parameters, the weight of textual similarity is set. The second row is for textual similarity, in which the weight of lexical similarity, \(a\), is assigned by the importance of Web service. In the third row of name similarity, \(b\), and datatype similarity, \((1-b)\), the weight is assigned to calculate the similarity of each...
identifier (operation, message, port) of structural parameter. Usually, same importance is
given to name and datatype similarity because both of them are equally important in Web
service. Depending on the importance of datatype similarity, the weight is adjusted. As
both the parameters are important equally, the weights are set to be equal. After calculat-
ing the similarity of each identifier of them (message, operation, port) from the third
row, the total structural similarity is calculated using the weights. Usually, the operation
parameter is most important, message parameter next, and then port parameter in typical
Web service. As shown in the fourth row, thus, the weight is assigned by this order for
operation (c), message (d), and port (1–c–d) parameter. The weights are tuned according
to the importance of the parameters of the target Web service using α and β in the fourth
row. The fifth row shows that a larger weight is assigned to subsumption (1–f) than dis-
tance (f).

| Table 5. Weight adjustment for the parameters of different web service. |
|-----------------|-----------------|
| Similarity      | Weight Name     | Weight Value |
| 1. Combined     | Textual         | g=a where a=.25 |
|                 | Structural      | h=a+1, h=.35 |
|                 | Concept         | 1-g, h=.45 |
| 2. Textual      | Lexical         | a = (6, 6, 6) |
|                 | Semantic        | 4, 4, 4 |
| 3. Service      | Name            | k = {45, 1-α |
| element         | Data Type       | else 50 |
| Structural      | Message Type    | 1-α |
|                 | Operation Type  | 1-β |
|                 | Port Type       | 1-c-d |
| 5. Concept      | Distance        | β=.45, |
|                 | Subsumption     | 1-f |

Note from Figs. 15 and 17 that the proposed scheme for weight adjustment allows
better performance than simple equal weighting for all the parameters.

6. CONCLUSION

In this paper we have proposed a scheme utilizing the structural similarity, semantic
similarity and concept similarity with bipartite matching for Web Service discovery. The
approach first identifies an initial set of matching by combining multiple lexical match-
ing strategies. Then semantic and structural similarities are calculated between a desired
service and a set of advertised services. The efficiency of the propose service discovery
scheme was evaluated in terms of speed and accuracy represented by precision, recall
rate, and F-measure. The matching for semantic Web Service was modeled using bipar-
tite graph of the nodes defined based on the ontology. Experiment with typical services
reveals that the proposed scheme improves the precision and recall rate compared to the
UDDI technique. It also demonstrated that the proposed scheme substantially reduces the
response time compared to the scheme employing Hungarian algorithm. The scalability
is also much higher.
The efficiency of service discovery can be further improved by parallelizing some steps. The weight factors used in calculating the similarity are crucial for achieving the best performance of Web Service discovery. The effect of the weights needs to be more thoroughly investigated. Also, the future research will emphasize the semantic information from a variety of sources such as context-aware, reasoning and integrated data.

REFERENCES


Shirin Akther Khanam received MS degree in Computer Engineering from Sungkyunkwan University, South Korea, in 2015. Her current research interests are semantic web and cloud computing.

Hee Yong Youn received the BS and MS degree in Electrical Engineering from Seoul National University, Seoul, Korea, in 1977 and 1979, respectively, and the Ph.D. degree in Computer Engineering from the University of Massachusetts at Amherst, in 1988. He had been Associate Professor of Department of Computer Science and Engineering, The University of Texas at Arlington until 1999. He is Professor of College of Information and Communication Engineering and Director of Ubiquitous Computing Technology Research Institute, Sungkyunkwan University, Suwon, Korea, and he is presently visiting SW R&D Center, Device Solutions, Samsung Electronics. His research interests include cloud and ubiquitous computing, system software and middleware, and RFID/USN. He has published numerous papers and received Outstanding Paper Award from the 1988 IEEE International Conference on Distributed Computing Systems, 1992 Supercomputing, IEEE 2012 Int’l Conference on Computer, Information and Telecommunication Systems, and CyberC 2014. Prof. Youn has been General Co-Chair of IEEE PRDC 2001, International Conference on Ubiquitous Computing Systems (UCS) in 2006 and 2009, UbiComp 2008, CyberC 2010, Program Chair of PDCS 2003 and UCS 2007, respectively.