Web Services Discovery Based on Schema Matching

Yanan Hao **Yanchun Zhang**

School of Computer Science and Mathematics
Victoria University
Melbourne, VIC, Australia
haoyn@csm.vu.edu.au
yzhang@csm.vu.edu.au

Abstract

A web service is programmatically available application logic exposed over Internet. With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can dynamically combine individual services to provide new value-added services. A main problem that remains is how to discover desired web services. In this paper, we propose a novel web services discovery strategy given a textual description of services. In particularly, we propose a new schema matching algorithm for supporting web-service operations matching. The matching algorithm catches not only structures, but also semantic information of schemas. We also propose a ranking strategy to satisfy a user's top-k requirements. Experimental evaluation shows that our approach can achieve high precision and recall ratio.

Keywords: Web service, XML Schema, Matching

1 Introduction

A web service is programmatically available application logic exposed over Internet. It has a set of operations and data types. The current set of web service specifications defines how to specify reusable operations through the Web-Service Description Language(WSDL) (Christensen, Curbera, Meredith & Weerawarana 2001), how these operations can be discovered and reused through the Universal Description, Discovery, and Integration(UDDI)API (Clement, Hately, Riegen & Rogers 2004), and how the requests to and responses from web-service operations can be transmitted through the Simple Object Access Protocol API(SOAP) (Gudgin, Hadley, Mendelsohn, Moreau & Nielsen 2003).

With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web (see, e.g., (Wang, Zhang, Cao & Varadharajan 2003, Bhiri, Perrin & Godart 2005, Wang, Cao & Zhang 2005, Limthanmaphon & Zhang 2004, Limthanmaphon & Zhang 2003)). Then they can combine individual services into more complex, orchestrated services.

A main problem that remains is how to discover desired web services. To find a service in UDDI, a user needs to input some keywords about the required service and then to browse the relevant UDDI category to locate relevant web services. Considering a large amount of service entries, this process is time consuming and frustrating. Furthermore, this method does not provide a mechanism assisting users in selection among similar web services. For example, consider the examples shown in Figure 1. A user searching for a *CreateOrder* service may also be interested in an *OrderGeneration* service. These two services are similar because they have the same function. But if the cost of *CreateOrder* is higher than that of *OrderGeneration*, the user would choose the latter one. This form of similarity potentially involves more web services. It is particularly useful and challenging in service composition.

This paper is devoted to address the problems above in web service search. The contribution of the work reported here is summarized as follows:

1. We propose algorithms for supporting web-service operations matching. The key part of our algorithms is a schema tree matching algorithm, which employs a new cost model to compute tree edit distances. Our new schema tree matching algorithm can not only catch structures, but also the semantic information of schemas.

2. Based on operations matching, we use the agglomeration algorithm to cluster similar web-service operations.

3. We also introduce a ranking strategy to satisfy a user's top-k requirements. Experimental evaluation shows that our approach can achieve acceptable result with high performance.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 gives an overview of our web service search approach. Section 4 describes a web-service operation matching algorithm, in which a new cost model and some XML schema transformation rules are defined. In section 5 we present how to cluster web-service operations. Section 6 describes our experimental evaluation. Section 7 gives some concluding remarks.

2 Related Works

Recently, several approaches have been proposed to find similar web services for a given web service. The earlier technique tModel presents an abstract interface to enhance service matching process. But the tModel needs to be defined while authors publishing in UDDI (Booth, Haas, McCab, Newcomer, Champion, Ferris & Orchard 2004). In (Sajjanhar, Hou & Zhang 2004), the authors propose a SVD-Based algorithm to locate matched services for a given service. This algorithm uses characteristics of singular

value decomposition to find relationships among services. But it only considers textual descriptions and can not reveal the semantic relationship between web services. Wang etc.(Wang & Stroulia 2003)proposed a method based on information retrieval and structure matching. Given a potentially partial specification of the desired service, all textual elements of the specification are extracted and are compared against the textual elements of the available services, to identify the most similar service description files and to order them according to their similarity. Next, given this set of likely candidates, a structure-matching method further refines the candidate set and assesses its quality. The drawback is that simple structural matching may be invalid when two web-service operations have many similar substructures on data types. Our approach is similar to this work, but we focus on the semantic simiarity not the structural similarity. Woogle (Dong, Halevy, Madhavan, Nemes & Zhang 2004) develops a clustering algorithm to group names of parameters of web-service operations into semantically meaningful concepts. Then these concepts are used to measure similarity of web-service operations. It relies too much on names of parameters and does not deal with composition problem however. (Shen & Su 2005) formally defines a behaviour model for web service by automata and logic formalisms. However, the behaviour signature and query statements need to be constructed manually, which can be very hard for common users.

3 An Overview of Web Services Search

The goal of our web-service search method is to find relevant web-service operations given a natural language description of desired web services and WSDL specifications of all available services published through UDDI. The WSDL files consist of textual description of web-service operations. Thus, firstly we use traditional IR technique TF (*term frequency*) and IDF (*inverse document frequency*) to find service operations that are most similar to the given description. We call these operations *candidate operations*. To do this, we extract words from web-service operation descriptions in WSDL. These words are pre-processed and assigned weight based on IDF. According to these weights, the similarity between the given description and a web-service operation description can be measured. A higher score indicates a closer similarity. For more details on measuring similarity among documents interested readers are referred to see (Salton, Wong & Yang 1975). After obtaining candidate operations, we employ a schema-match based method to measure similarity among them. Based on operations matching, the candidate operations are clustered into some *operation sets*. For each operation set the operation with the minimum cost in it is output as a search result. Since each candidate operation has a score, we can rank search results simply by the score of operations. Now we turn to the main focus of this paper, which is measuring similarity between web-service operations based on schema matching.

4 Web-service Operation Matching

4.1 Web-service Operation Modelling

Definition 1 *A web service is a triple* $ws = (TpSet, MsgSet, OpSet)$*, where TpSet is a set of data types; MsgSet is a set of messages conforming to the data types defined in TpSet;* $OpSet = \{op_i(input_i, output_i) | i = 1, 2, ..., n\}$ *is a set of operations, where* $input_i$ *and outputi are parameters(messages) for exchanging data between web-service operations.*

WS1: Web Service: CreateOrderService
Operation: *OrderBuilder*
Input: UserID DataType: *int*
Output: ProductsList DataType: *Order*

WS2: Web Service: OrderGeneration
Operation: *GetOrder*
Input: UserName DataType: *String*
Output: MyProducts DataType: *PurchaseOrder*

Figure 1: Sample Web-service Operations

Figure1 gives two web-service operations used as examples in this paper. According to definition 1, a web service can be briefly described as a set of operations.

Definition 2 *Each web-service operation is a multi-input-multi-output function of the form* $f : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$*, where* s_i *and* t_j *are data types in according with XML schema specification. We call* f *a* dependency *and* s_i/t_j *a* dependency attribute.

A dependency attribute can be a complex data type or a primitive data type. Complex data types, for example in *Order* and *PurchaseOrder* in Figure 1, define the structure, content, and semantics of parameters, whereas primitive data types, like *int* and *string*, are typically too coarse to reflect semantic information. We can convert primitive data types to complex data types by replacing them with their corresponding parameters. For example, in figure 1, *string* is converted into *UserName* type while *int* is converted into *UserID* type. Both *UserName* and *UserID* are considered as complex data types with semantics. Thus, each data type defined in a web-service operation carries semantic meaning.

An XML schema can be modelled as a tree of labelled nodes. We categorize a node n by its label:

1. **Tag node**: Each tag node n is associated with an element type T. T is also the tag name of node n.
2. **Constraint node**:
   - **Sequence node**: A sequence node indicates its children are an ordered set of element types. We use [“,”] to denote a sequence node.
   - **Union node**: A union node represents a choice complex-type, that is, the instance of which can only be one of the children types in accordance with the XML Schema specification. We use [“|”] to denote a union node.
   - **Multiplicity node**: Each node may optionally have a multiplicity modifier [m, n] indicating that in the instance, its occurrence is between m and n. This corresponds to the *minOccurs* and *maxOccurs* constraints in XML Schema. We use [m, n] to denote a multiplicity node.

As an example, the schema tree of data type *Order* is shown in Figure2.

As we can see, data types defined in web-service operations carry semantic information. Intuitively,

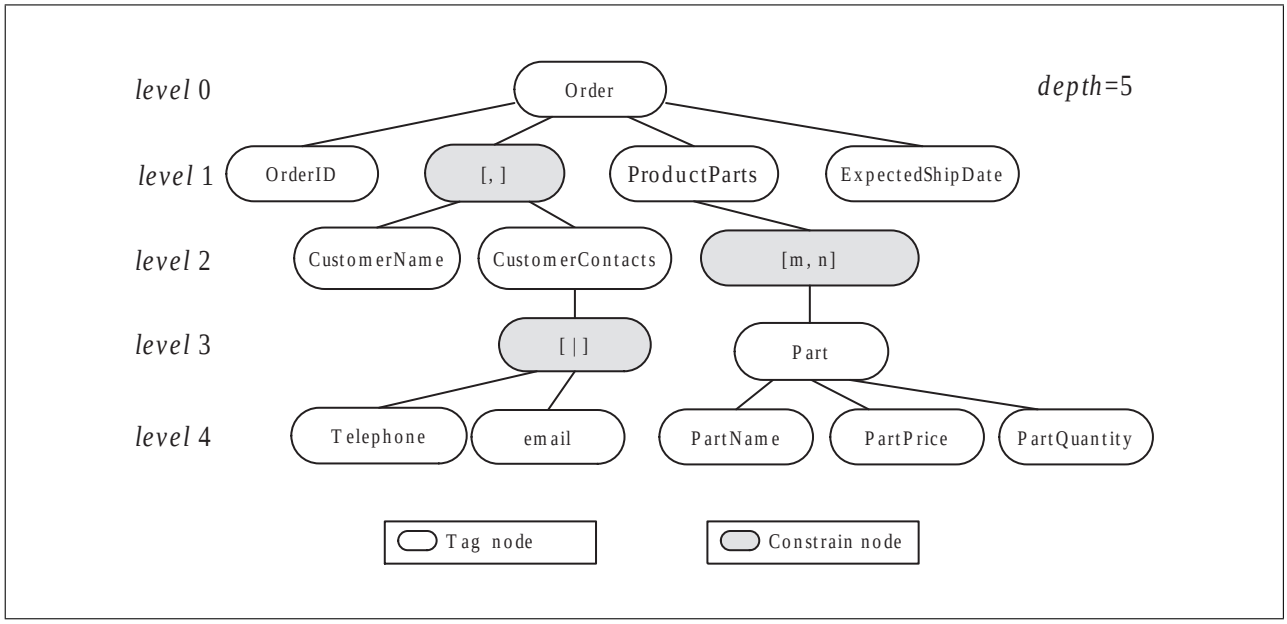


Figure 2: XML schema tree of *Order* type

we consider two web-service operations similar if they have similar input/output data types. Thus the problem of web-service operation matching is converted to the problem of schema tree matching.

4.2 Tree Edit Distance

Many works have been done on the similarity computation on trees. Among them *tree edit distance* is one of the efficient approaches to describe difference between two trees. We introduce tree edit operations first. Generally, the tree edit distance operations include: (a) *node removal*, (b) *node insertion*, and (c) *node relabelling*. Such a set of operations can be represented by a mapping with minimum cost between the two trees. The concept of mapping is formally defined as follows (Reis, Golgher, d. Silva & Laender 2004):

Definition 3 *Let T_x be a tree and let $T_x[i]$ be the* i*th node of tree T_x in a preorder traverse of the tree. A mapping between a tree T_1 and a tree T_2 is a set* M *of ordered pairs (i, j), satisfying the following conditions for all (i_1, j_1),$(i_2, j_2) \in M$*

1. *$i_1 = i_2$ iff $j_1 = j_2$;*
2. *$T_1[i_1]$ is on the left of $T_1[i_2]$ iff $T_2[j_1]$ is on the left of $T_2[j_2]$;*
3. *$T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$*

Figure 3 gives an example of tree mapping. This mapping also shows the way of transforming the left tree to the right one. A dotted line from a node of T_1 to a node of T_2 indicates that the node of T_1 should be changed if the corresponding nodes are different, remaining unchanged otherwise. Nodes of T_1 not connected by dotted lines are deleted, and nodes of T_2 not connected are inserted.

Each of these operations is assigned a cost. The tree edit distance between two trees is defined as the minimal set of operations to transform one tree into the other.

Our schema matching algorithm is based on tree edit distance. However, the problem in our case is more complex than the traditional tree edit distance for the following reasons:

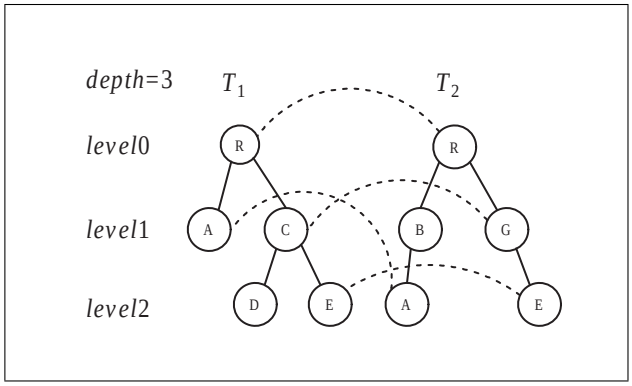


Figure 3: Example of tree mapping

1. The labels of an XML Schema tree can carry complex type information (e.g., union, multiplicity) which makes simple relabelling operations inapplicable. For instance, let T_1 and T_2 be the schema trees of *Order* and *PurchaseOrder* respectively. Let us imagine there exits a mapping M between T_1 and T_2, and there are two node-mapping pairs (i_1, j_1),$(i_2, j_2) \in M$, where $T_1[i_1]$ =[*telephone* |*email*], $T_2[j_1]$ =*email*, $T_1[i_2]$=*price*, and $T_2[j_2]$=*quantity*. The edit operation of (i_1, j_1) should have less cost than that of (i_2, j_2).But in the previous work, all tree edit operations are considered to have same unit distance.
2. The labels of nodes carry semantic information. So a relabelling from one node to another unrelated node will have more cost than to a semantic related node. For example, relabelling *part* to *item* is less costing than relabelling *price* to *email*.
3. We argue that tree edit operations on low-level nodes of a tree should have more influence than operations on high-level nodes. So, for example, if a *part* node on the third level of the first tree is mapped into a *part* node on the fifth level of the second tree, the edit operation cost should not be zero. But the traditional works on tree edit distance do not consider the difference and assign each edit operation unit cost.

In the next section, we present a new cost model to compute the cost of tree edit operation, as a consequence, the tree edit distance of two schema trees.

4.3 Cost Model

Measuring similarity between two XML schema trees equals to finding a mapping with minimum cost. So, the cost of each edit operation involved in the mapping needs to be computed first. In this section we introduce a new cost mode based on tree edit distance presented in (Zhang & Shasha 1989) (Xie, Sha, Wang & Zhou 2006). The new cost model integrates weights of nodes and semantic connections between nodes. Let T_1, T_2 be two schema trees and let n, $node_1$ and $node_2$ be tree nodes. Formally, the cost model is defined as

$$cost(\rho) = \begin{cases} weight(n)/W(T_1,T_2), & \text{if} \rho = insert(n) \\ weight(n)/W(T_1,T_2), & \text{if} \rho = delete(n) \\ \alpha \times wd(node_1, node_2) & \text{if} \rho \text{ relabels} \\ +\beta \times sd(node_1, node_2) & node_1 \text{to } node_2 \end{cases}$$

where ρ indicates a tree edit operation. $weight(n)$ shows the weight of node n. $wd(node_1, node_2)$ and $sd(node_1, node_2)$ give the weight and semantic difference of $node_1$ and $node_2$, respectively. α and β are weights of wd and sd, satisfying $\alpha + \beta = 1$. $W(T_1, T_2)$ is defined as $W(T_1, T_2) = weight(T_1) + weight(T_2)$, where $weight(T_i)$ is the sum of all node weights of tree $T_i (i = 1, 2)$. $wd(node_1, node_2)$ is defined as

$$wd(node_1, node_2) = \frac{\|weight(node_1) - weight(node_2)\|}{W(T_1, T_2)}$$

where $node_1 \in T_1$ and $node_2 \in T_2$.

In the next two sections, we propose a set of schema-tree transformation rules and a semantic similarity measure to compute wd and sd, i.e. the weight and semantic difference of nodes.

4.4 XML Schema Tree Transformation

Definition 4 *The tag name of a node is typically a sequence of concatenated words, with the first letter of every word capitalized (e.g., ExpectedShipDate). Such a set of words is referred to as a* word bag. *We use $\pi(n)$ to denote the word bag of node n.*

Definition 5 *Two word bags $\pi(n_1)$ and $\pi(n_2)$ are said to be equal, only if they have same words.*

Two nodes are considered different if they have different word bags. The word bag reflects semantic meaning of a node. As we shall see later, using word bags we can measure the semantic similarity between two schema-tree nodes.

Definition 6 *Let $level(n)$ denote the level of node n in schema tree T. The weight of node n is defined by a weight function:*

$$weight(n) = 2^{depth(T) - level(n)} (\forall n \in T)$$

The weights of all nodes fall in the range of $[2, 2^{depth(T)}]$. Each weight reflects the importance of a node in schema tree T.

From section 4.2, it can be seen that traditional tree-edit-distance algorithm is not suitable for XML schema trees. It does not deal with constraint nodes. We propose three transformation rules to solve this problem. These rules are used to transform constraint nodes, specifically, sequence nodes, union nodes and multiplicity nodes to tag nodes. At the same time, the weights of nodes are reassigned.

1. *split*: This rule is applied to sequence nodes. A sequence node $l = [l_1, l_2, ..., l_s]$ is split into an ordered list of nodes $l_1, l_2, ..., l_s$, where $l_i (i = 1, 2, ..., s)$ is a child node of the sequence node l. After the split process, each sequence node is replaced by its child nodes. Each child node l_i inherits the weight of its parent node l as a new weight. Figure 4(a) gives an example of the split rule.

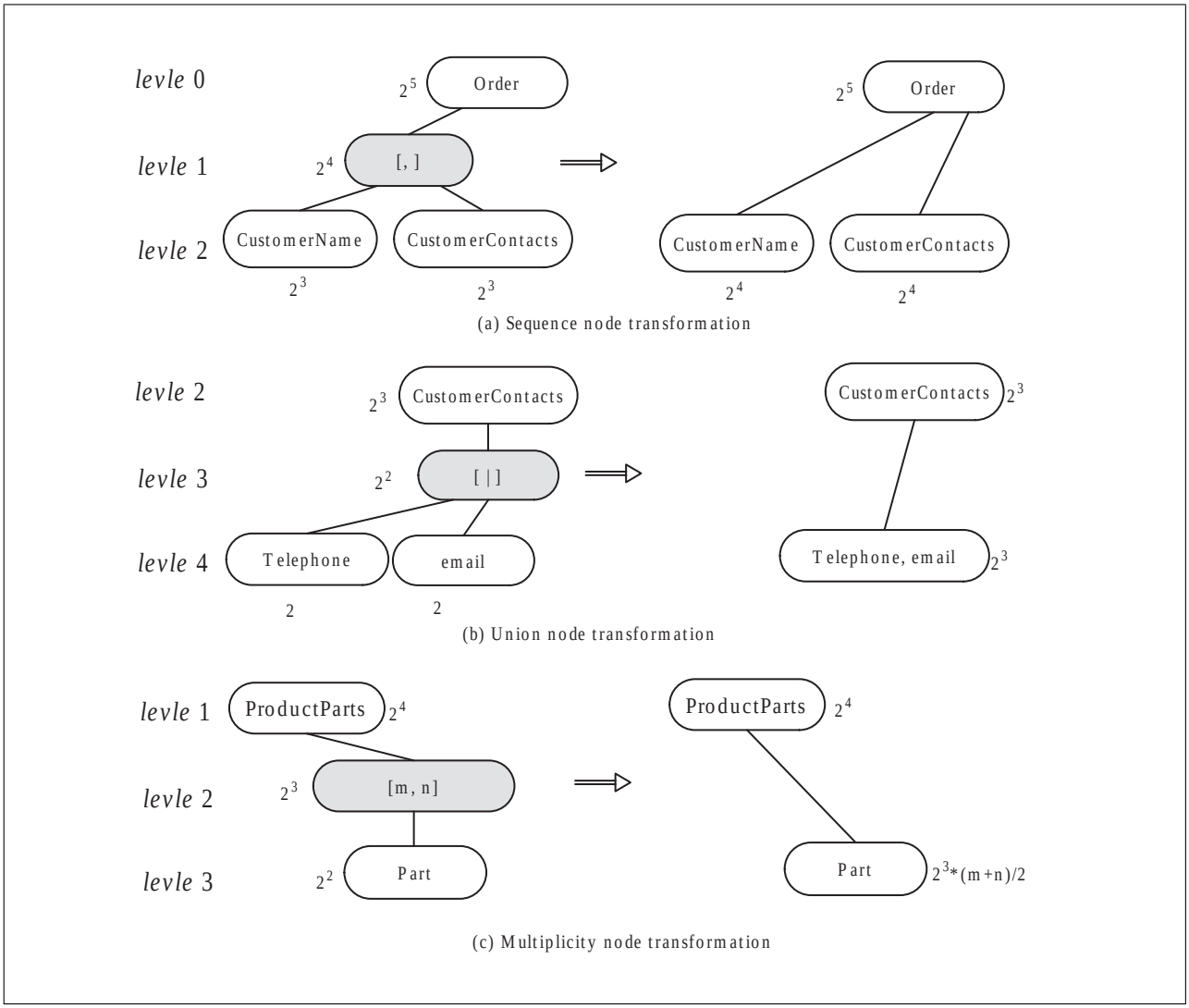
2. *merge*: This rule is applied to union nodes. After the mergence process, each union node is replaced by all its option nodes, i.e. all its child nodes. All child nodes of the union node $l = [l_1|l_2|...|l_s]$ are merged into a new node l^*, while the union node l is deleted. The weight of node l^* is s times the weight of l. Each l_i's$(i = 1, 2, ..., s)$ word bag is also merged into a new word bag. Formally, we have $weight(l^*) = weight(l) \times s$. Figure 4(b) gives an example of the merge rule.

3. *delete*: This rule is applied to multiplicity nodes. We delete a multiplicity node $l = [m, n] (m, n \in N)$ and scale up the weight of each of its child nodes l_i. After the deletion process, each multiplicity node is replaced by its child nodes. We have $weight(l_i) = weight(l) \times (m + n)/2$. Figure 4(c) gives an example of the delete rule.

Note that the definition of complex types can be nested according to XML schema specification. Thus, given a schema tree, we apply the three transformation rules to its nodes level by level, from bottom to top. This process is formally described as *bottom-up-transformation* algorithm (see Algorithm 1). The time complexity of Bottom-up-transformation is $O(n)$, where n is the number of nodes in the XML schema tree.

```
   input  : schema tree T
   output: transformed schema tree T*
 1 d = GetDepth(T);
 2 for i ← d to 0 do
 3     foreach node p ∈ level_i do
 4         if p is a sequence node then
 5             weight(each of p's child
               nodes)=weight(p);
 6             add p's child nodes to p's parent's
               child list;
 7             delete p;
 8         end
 9         if p is a union node with s options
           {l_i|i = 1, ..s} then
10             merge p's child nodes into a new
               node q;
11             add q to p's parent's child list;
12             weight(q) = weight(p) × s;
               π(q) = ⋃_{i=1}^{s} π(l_i) ;
13
14             delete p;
15         end
16         if p is a multiplicity node [m, n] then
17             add p's child node to p's parent's
               child list;
18             weight(p's child
               node)=weight(p) × (m + n)/2;
19             delete p;
20         end
21     end
22 end
```

Algorithm 1: Bottom-up-transformation



(a) Sequence node transformation

(b) Union node transformation

(c) Multiplicity node transformation

Figure 4: Examples of XML schema tree transformation

4.5 Semantic Measurement between Schema-tree Nodes

After the bottom-up transformation, schema tree T is converted into a new schema tree T^*. Each node n of T^* is a tag node, whose word bag may come from two or more word tags because of nodes mergence by the merge rule. Formally, node n can be regarded as a vector (W, B), where W is the weight of node n and B is the word bag of node n. As we can see, after transformation the weight difference between two nodes can be computed by the new cost model. In this section, we present a strategy to determine the semantic similarity of two schema-tree nodes, i.e. the semantic distance between two word bags.

Our approach relies on a hypothesis that two co-occurrence words in a WSDL description tend to have same semantics. We exploit the co-occurrence of words in word bags to cluster them into meaningful concepts. To improve accuracy of semantic measurement, a pre-processing step is carried out first before words clustering. Pre-processing includes word stemming, removing stop words and expanding abbreviations and acronyms into the original forms.

Let $I = \{w_1, w_2, ..., w_m\}$ be a set of words. These words come from word bags of all schema-tree nodes to which similarity measurement is applied. Let D be a set of candidate web-service operation descriptions available in WSDL files. We introduce association rules to reflect the notion of word co-occurrence. An *association rule* is an implication of the form $w_i \rightarrow w_j$, where $w_i, w_j \in I$. The rule $w_i \rightarrow w_j$ holds in the descriptions set D with *support* s and *confidence* c, where s is the probability that w_i occurs in an web-service operation description; c is the probability that w_j occurs in an operation description, given w_i is known to occur in it. All association rules can be found by the A-Priori algorithm (Kaufman & Rousseeuw 1990). We are only interested in rules that have confidence above a certain threshold t.

We use the agglomeration algorithm (Kaufman & Rousseeuw 1990) to cluster words set $I = \{w_1, w_2, ..., w_m\}$ into concept set $C = \{C_1, C_2, ...\}$. There are three steps in the clustering process. It begins with each word forming its own cluster and gradually merges similar clusters.

1. Set up a confidence matrix $M_{m \times m}$. M_{ij} is a two-dimensional vector (s_{ij}, c_{ij}), where s_{ij} and c_{ij} are the support and confidence of association rule $w_i \rightarrow w_j$, respectively.

2. Find M_{ij} with the largest c_{ij} in the confidence matrix M. If $c_{ij} > t$ and $s_{ij} > t$ then merge these two clusters and update M by replacing the two rows with a new row that describes the association between the merged cluster and the remaining clusters. The distance between two clusters is given by the distance between their closest members. There are now $m - 1$ clusters and $m - 1$ rows in M.

3. Repeat the merge step until no more clusters can be merged.

Finally, we get a set of concepts C. Each concept C_i consists a set of words $\{w_1, w_2, ...\}$. To compute semantic similarity between schema-tree nodes, we replace each word in word bags with its corresponding concept, and then use the TF/IDF measure.

After schema-tree transformation and semantic similarity measure, the tree edit distance can be applied to match two XML schema trees by the new cost model.

4.6 Identifying Similar Web-service Operations

As it has been mentioned before, we use tree edit distance to match two schema trees. It is equivalent to finding the minimum cost mapping. Let M be a mapping between schema tree T_1 and T_2, let S be a subset of pairs $(i, j) \in M$ with distinct word bags, let $D(I)$ be the set of nodes in $T_1(T_2)$ that are not mapped by M. The mapping cost is given by $C = Sp + Iq + Dr$, where p, q and r are the costs assigned to the relabel, insertion, and removal operations according to the cost model proposed in section 4.3. We call C the *match distance* between T_1 and T_2, denoted as $C = ED(T_1, T_2)$. Match distance reflects semantic similarity of two schema trees.

Now let us see the algorithm for matching web-service operations. Given two web-service operations $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$ and $op_2 : x_1, x_2, ..., x_l \rightarrow y_1, y_2, ..., y_k$, we identify all possible matches between two lists of schema trees, and return the source-target correspondence that minimizes the overall match distance between the two lists. See Figure 5. We formally describe this process in algorithm 2.

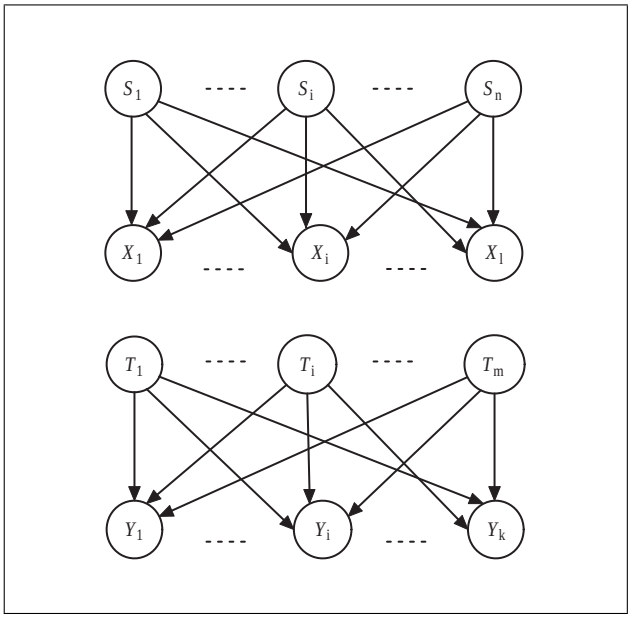


Figure 5: Matching Web-service Operations

5 Clustering Web-service Operations

Suppose $OP = \{op_1, op_2, ..., op_q\}$ is a set of web-service operations and each pair of operations op_i and op_j $(i, j = 1, 2, ..., q)$ match with the distance of z_{ij}. We classify OP into a set of clusters $\{op_{c1}, op_{c2}, ...\}$. The clustering algorithm is described as below. It begins with each operation forming its own cluster and gradually merges similar clusters.

input : $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$
$op_2 : x_1, x_2, ..., x_l \rightarrow y_1, y_2, ..., y_k$
output: The match distance Z between op_1 and op_2

for $i \leftarrow 1$ **to** n **do**
  $S_i = min\{ED(s_i, x_j)|j = 1, 2, ..., l\}$;
end
for $i \leftarrow 1$ **to** m **do**
  $T_i = min\{ED(t_i, y_j)|j = 1, 2, ..., k\}$;
end
$Z = \sum_{i=1}^{n} S_i + \sum_{i=1}^{m} T_i$

Algorithm 2: Algorithm for matching web-service operations

1. Set up a match matrix $M_{q \times q}$. M_{ij} is the match distance of operation op_i and op_j.
2. Find the smallest M_{ij} in the match matrix M. If $M_{ij} <$ threshhold δ then merge these two clusters and update M by replacing the two rows with a new row that describes the association between the merged cluster and the remaining clusters. The distance between two clusters is given by the distance between their closest members. There are now $q - 1$ clusters and $q - 1$ rows in M.
3. Repeat the merge step until no more clusters can be merged.

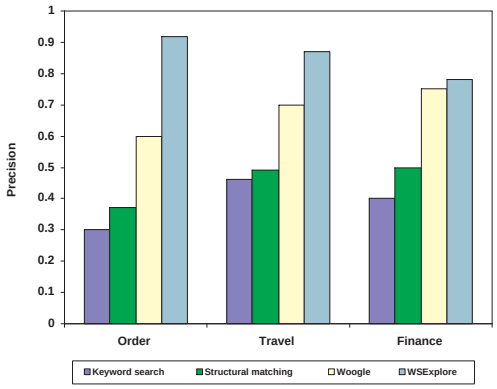
Finally, a set of clusters $\{OPC_1, OPC_2, ...\}$ is obtained. For example, Figure 1 shows a sample cluster of two web-service operations: *GetOrder* and *OrderBuilder*. Given a cluster OPC_i and an operation $OPC_{ik} \in OPC_i$, OPC_{ik} is called the *pattern* of OPC_i if it has the minimum cost among OPC_i. We output all the patterns as search results.

6 Experiments and Evaluations

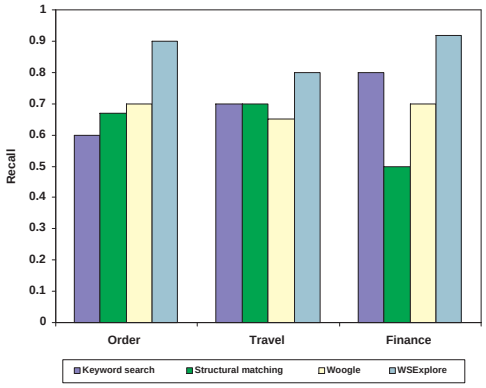
We have implemented a prototype system and conducted some experiments to evaluate the effectiveness and efficiency. We measured the efficiency of our web-service operation matching method by comparing it with keyword search, Woogle and structure matching. The experiments were conducted on a P4 Windows machine with a 2GHz Pentium IV and 512M main memory. The data set used in our tests is a group of web-service operations whose WSDL specifications are available, so we can obtain their textual descriptions and XML schemas of input/output data types. The data contains 223 web services including 930 web-service operations. We chose 7 web-service operations from three domains: *order*(3), *travel*(2) and *finance*(2). Each operation description was used as the basis for desired operations.

We use recall and precision ratio to evaluate the effectiveness of our approach. The precision(p) and recall(r) are defined as $p = \frac{A}{A+B}, r = \frac{A}{A+C}$ where A stands for the number of returned relevant operations, B stands for the number of returned irrelevant operations, C stands for the number of missing relevant operations, $A + C$ stands for the total number of relevant operations, and $A + B$ stands for the total number of returned operations. Specially, the top 100 search results are considered in our experiments for each web-service operation search.

We evaluated the efficiency of our approach by comparing the recall and precision of operation search with three other methods: keyword searching method, structure matching (Wang & Stroulia 2003) and Woogle (Dong et al. 2004). The results obtained



(a)



(b)

Figure 6: Precision and recall comparisons

are shown in Figure 6. As can be seen, the precisions of our approach are 92%, 87% and 78% respectively, almost always outperforming that of keyword, structrure and Woogle. The precision is higher on *order* operations but lower in *finance* operations because *order* operations have more complex structures and richer semantics in input/output data types. This indicates that, by combining structural and semantic information, the precision of our approach improves significantly, compared to the results obtained with structural or semantic information only. It is also can be seen that by keyword method the precision is rather low whereas the recall is rather high. This demonstrates textual description of operations contain much useful information but also much noise at the same time.

7 Conclusions

In this paper we have presented a novel approach to retrieve desired web-service operations of a given textual description. The concept of tree edit distance is employed to match web-service operations. Meanwhile, some algorithms are proposed for measuring and grouping similar operations. Our approach can be used for web-service searching tasks with top-k requirements.

As part of on-going work, we are interested in improving performance of the web-service operation matching algorithm, as well as integrating more semantic information to our system in order to improve the search precision.

References

Bhiri, S., Perrin, O. & Godart, C. (2005), Ensuring required failure atomicity of composite web services, *in* 'WWW', pp. 138–147.

Booth, D., Haas, H., McCab, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. (2004), Web services architecture. http://www.w3.org/tr/ws-arch/.

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001), Web services description language (wsdl) 1.1. http://www.w3.org/tr/wsdl.

Clement, L., Hately, A., Riegen, C. V. & Rogers, T. (2004), Universal description discovery and integration. http://uddi.org.

Dong, X., Halevy, A. Y., Madhavan, J., Nemes, E. & Zhang, J. (2004), Simlarity search for web services, *in* 'VLDB', pp. 372–383.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. J. & Nielsen, H. F. (2003), Simple object access protocol (soap) version 1.2. http://www.w3.org/tr/soap/.

Kaufman, L. & Rousseeuw, P. J. (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley, New York. ID: 58.

Limthanmaphon, B. & Zhang, Y. (2003), Web service composition with case-based reasoning., *in* 'ADC', pp. 201–208.

Limthanmaphon, B. & Zhang, Y. (2004), Web service composition transaction management., *in* 'ADC', pp. 171–179.

Reis, D. D. C., Golgher, P. B., d. Silva, A. S. & Laender, A. H. F. (2004), Automatic web news extraction using tree edit distance, *in* 'WWW', pp. 502–511.

Sajjanhar, A., Hou, J. & Zhang, Y. (2004), Algorithm for web services matching, *in* 'APWeb', Vol. 3007, pp. 665–670.

Salton, G., Wong, A. & Yang, C. S. (1975), 'A vector space model for automatic indexing', *Commun.ACM* **18**(11), 613–620.

Shen, Z. & Su, J. (2005), Web service discovery based on behavior signatures, *in* 'SCC', Vol. 1, pp. 279–286 vol.1.

Wang, H., Cao, J. & Zhang, Y. (2005), 'A flexible payment scheme and its role-based access control', *IEEE Trans. Knowl. Data Eng.* **17**(3), 425–436.

Wang, H., Zhang, Y., Cao, J. & Varadharajan, V. (2003), 'Achieving secure and flexible m-services through tickets', *IEEE Transactions on Systems, Man, and Cybernetics, Part A* **33**(6), 697–708.

Wang, Y. & Stroulia, E. (2003), Flexible interface matching for web-service discovery, *in* 'WISE'.

Xie, T., Sha, C., Wang, X. & Zhou, A. (2006), Approximate top-k structural similarity search over xml documents, *in* 'APWeb', Vol. 3841, pp. 319–330.

Zhang, K. & Shasha, D. (1989), 'Simple fast algorithms for the editing distance between trees and related problems', *SIAM J.Comput.* **18**(6), 1245–1262.