WSSM：A WordNet-Based Web Service Similarity Mining Mechanism

Xianyang Qu，Hailong Sun，Xiang Li，Xudong Liu，Wei Lin
School of Computer Science and Engineering
Beihang University
Beijing 100191, China
{quxy, sunhl, lixiang, liuxd, linwei}@act.buaa.edu.cn

***Abstract*—One of the challenging issues in web service research is to efficiently find appropriate services from large volumes of existing services. In this paper, we propose a WordNet-based web service similarity mining mechanism called WSSM to address this issue. WSSM can effectively cluster services into different categories so as to reduce the searching space. As a result, it improves service discovery efficiency greatly. Finally, based on the web services collected from online service repositories, we conduct extensive experiments to study the performance of our implementation.**

Keywords-web service; service discovery; data mining; similarity

I. Introduction

With the rapid development of web services, Service Oriented Architecture (SOA) [1] has been widely adopted by both industry and research communities. Large enterprises are increasingly relying on SOA as the methodology for large-scale software development and sharing services within an organization.

As the amount of available web services grows continually, efficient discovery of the needed services from a mass of candidate services becomes an important issue in web service research.

Although there has been a lot of research work on service discovery [2, 3, 4], this issue is still far from being well addressed. The syntax-based service discovery, which is mostly based on string matching of service names, can be easily implemented, but the precision of service locating cannot be guaranteed. Although some methods in information retrieval area have been adopted to increase the precision of syntax-based service discovery, i.e., TF/IDF [4], they do not fit for service discovery due to the need for a great deal of information so as to guarantee accuracy. The semantic-based service discovery, which relies on ontologies to describe the semantics of service functions and behaviors, uses logic deduction and reasoning to implement the corresponding matching algorithms. In this regard, several representative systems have been developed, including Augment UDDI Registry [2] developed by CMU and SpeedR [3] from University of Georgia. Although the semantic approach can achieve a high matching precision, the efficiency is usually not satisfactory due to the complexity of the algorithms. Additionally, the service discovery systems based on semantic approach are difficult to implement. In a word, the existing approaches have limitations either on implementation feasibility, system efficiency or matching precision.

In this work, we propose a WordNet [16] based web service similarity mining scheme called WSSM, focusing on reducing searching space to improve the efficiency of service discovery with acceptable complexity. In particular, our contributions are as follows:

1. We propose a new description model for web service, which is a premise for the matchmaking between user requirements and candidate web services.
2. We design a web service similarity computing algorithm base on the proposed model. Then, a clustering algorithm is proposed to cluster similar web services into different categories so as to reduce the size of searching space and thus improve the efficiency of service discovery.
3. We implement a service clustering system based on WSSM. And we show the efficiency of our implementation through experiments based on real services collected from Internet.

The rest of the paper is organized as follows. Section II describes the definitions of web service function model and web service similarity. In Section III, we propose the design and implementation, including overview of WSSM, core processes and algorithms. In Section IV, we perform comprehensive evaluations of WSSM based on real services collected from Internet. We discuss some related work in Section V. Finally, in Section VI, conclusion is drawn.

II. Web service Description Model and Similarity Computing

The web service similarity in this paper refers to the functional similarity of web services. For example, service *A* provides a function to reserve self-service apartments in Beijing, while service *B* provides a function to reserve three-star hotels in Beijing. Then we can say service *A* and *B* are similar in function since both of them provide a hotel reservation function. Before giving the similarity computing method, we propose a web service description model, which defines the meta-data format of a web service.

***Definition 1**:* A word collection *c* is a sequence of words $w_1w_2...w_{|c|}$, where w_i is a word from a fixed vocabulary, thus we present a word collection with a bag of words, i.e. $c = \{ w_1, w_2, ..., w_{|c|}\}$.

Definition 2: The meta-data of a web service function can be represented as a triple, $W = \{T, B, A\}$, where T denotes

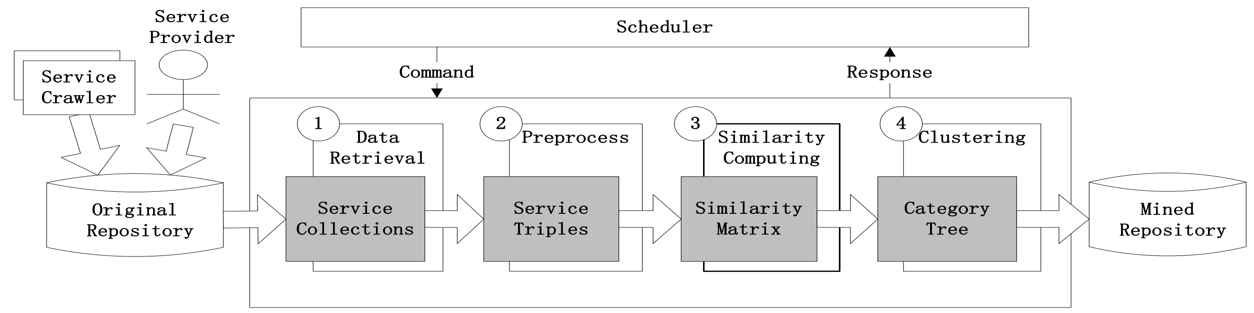


Figure 1. Overview of WSSM

the title information including service names and service comments, *B* denotes body information including every operation name, operation comment, input message name and output message name and *A* denotes additional information including description from web pages and service containers where a service is deployed. In fact, each element in the meta-data is a word collection: $T = \{ w_1, w_2, ..., w_{|T|}\}$, $B = \{ w_1, w_2, ..., w_{|B|}\}$, $A = \{ w_1, w_2, ..., w_{|A|}\}$.

Given the above web service meta-data, we provide the following similarity computing methods.

Definition 3: *SimWord*(W_1, W_2) denotes the similarity of lexical meaning between two words W_1 and W_2. As a word collection is composed of different words, the similarity of lexical meaning between two word collections *SimSet* (Set_1, Set_2) can be converted into similarity between words. We will explain this in detail later in Section III.

Definition 4: Given two web services S_1 and S_2, the functional similarity value can be calculated as follows.

$$\begin{aligned} Sim(S_1, S_2) = & \\ & \alpha * SimSet(S_1.T, S_2.T) \\ & + \beta * SimSet(S_1.B, S_2.B) \\ & + \gamma * SimSet(S_1.A, S_2.A) \end{aligned} \quad (1)$$

where *S.T* denotes the word collection of service title information; *S.B* denotes the word collection of service body information, *S.A* denotes the word collection of service additional information; *T, B, A* represent the weights in similarity computation for title, body and additional information respectively; α, β, γ are coefficients of different parts and they sum to 1.

III. DESIGN AND IMPLEMENTATION OF WSSM

In this section, we will introduce WSSM in detail, including the whole process of WSSM, core idea of service similarity and algorithm of clustering.

A. *Overview of WSSM*

With the introduction of web service similarity, we can first cluster all the services with similar functions as a function class, and service discovery will be implemented by two steps. Firstly, locating the target function class and then searching the target service inside a function class. With this method, the size of the searching space will be dramatically reduced and the service discovery efficiency will be significantly improved.

Fig. 1 illustrates the overview of WSSM. There are two repositories in WSSM. Original repository contains meta-data about services, including WSDL files and other documents published by service providers and service crawler, while mined repository consists of results generated by WSSM. The four modules with serial number (1, 2, 3 and 4) comprise the core processes of WSSM. Firstly, information retrieval module gets meta-data from original repository and organizes them as services collections. Secondly, preprocess module extracts keywords from meta-information. The keywords are then used to construct the triples, which are the meta-data of web service functions described in *Definition 2*. Thirdly, similarity values between any two services are computed by similarity computing module with the equation proposed in *Definition 4*. Suppose there are *N* web services totally, the similarity values among them are represented as an $N \times N$ matrix. Finally, a clustering algorithm handles the matrix produced in the last step to cluster services into categories. As a result, a category tree can be obtained and stored in the mined repository, which provides the information of all the categories of the existing web services, and to which category a web services belongs. Schedule module executes every module automatically in the same manner, so it can reduce coupling coefficient.

The following section will introduce core modules in detail.

B. *Meta-data retrieval and preprocessing*

The description information of a web service function mainly comes from its WSDL document and partly from web pages and information about the service container onto which the service is deployed. Therefore, we adopt two methods to retrieve the meta-data of a web service.

- Retrieve the title information *T* and the body information *B* from WSDL documents by a WSDL parser.
- Retrieve the additional information *A* from service provider and web pages by the service crawler.

The retrieved information may contain punctuations, non-standard form words, misspelled words, stop words, compact words, repeated words, and so on. Thus we need to preprocess the meta-data to compute the similarities between two web services accurately.

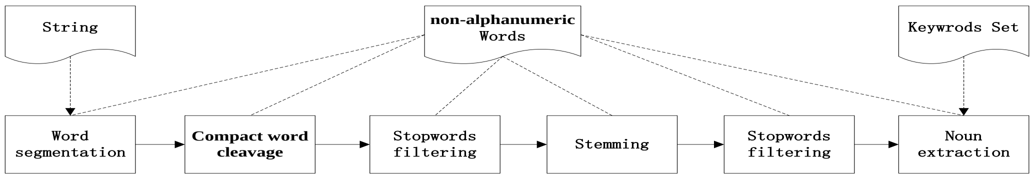


Figure 2. Procedure of the preprocessing of meta-information of web services

As shown in Fig. 2, the preprocessing procedure consists of the following steps:

- Word segmentation. As English words are separated by white spaces, this step just replaces non-alphanumeric characters with white spaces.
- Compact word cleavage. The names of services, functions and parameters that are always named in Pascal or Camel form contain important functional information of services. Thus, they need to be cleaved. For example, service name "RealTimeMarketData" need to be split into "real time market data".
- Stop words removal. Stop words like "a" and "an" should be removed to improve precision.
- Stemming [5]. Stemming can improve recall by removing term suffixes and reducing all forms of a term to a single stemmed form.

C. *Web service similarity computing*

A word collection will be generated after preprocess. As mentioned in *Definition 2*, the obtained word collection can be further divided into three sub-collections: title keyword collection *T*, body keyword collection *B* and additional information keyword collection *A*. According to *Definition 4*, the similarity computing can be decomposed into the following two steps.

- Calculate *SimSet*($W_1.T$, $W_2.T$), *SimSet*($W_1.B$, $W_2.B$) and *SimSet*($W_1.A$, $W_2.A$) .
- Determine the appropriate values for coefficients.

1) Similarity computing between word collections: There are many methods of similarity computing between two word collections, such as edit distance-based approach [6], rules-based approach, vector-based approach, word overlap-based approach [7], and TF/IDF-based approach. However, these methods do not apply to the context in this work due to their specific requirements. Rule-based approach relies on a rules repository which is hardly to build. Besides, the other four methods are based on statistics of words, but not semantic information.

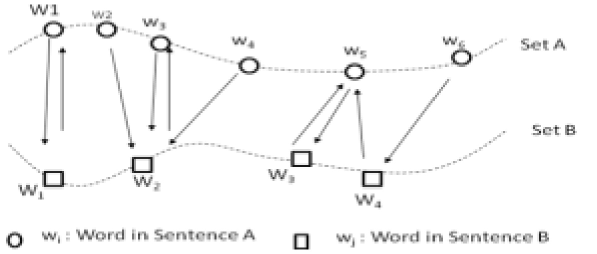


Figure 3. Similarity computing between two collections

Here we adopt the Part-of-Speech Similarity [8] computing method. In this method, given two word collections S_1 and S_2, the similarity between S_1 and S_2 is calculated by *Equation* (*2*).

$$Sim_{ps}(S_1,S_2)=\frac{\sum_{w\in S_1} Sim_m(w,S_2)+\sum_{w\in S_2} Sim_m(w,S_1)}{|S_1|+|S_2|} \quad (2)$$

where $Sim_m(w,S)$ is the reciprocal of distance between word *W* and collection *S*. This distance is defined as the minimum distance between word *W* and word W_i in collection *S*, which has the same part of speech with *W*.

Fig. 3 exemplifies the similarity computing between two word collections. The connection line starting from $A.W_3$ to $B.W_2$ represents W_2 is the most similar word to *W3* in the collection *B*.

With *Equation* (*2*), the step one in Section III.C can be reduced to similarity computing between two words.

2) Similarity computing between two words: Existing methods for similarity computing between two words mainly fall into two categories. One is the edit distance-based approache with less accuracy. The other is the dictionary-based approache. For the latter approach, usually the WordNet-based method is adopted, which can be divided into three types: edge-based [9], node-based [10] and hybrid-based [11] approaches.

The node-based and hybrid-based approaches are more complicated in calculation. These two approaches need to calculate the value of the information content of nodes. And this process involves a huge amount of computation and requires a rather complete dictionary. In addition, Alexander Budanitsky and Graeme Hirst made several experiments for five WordNet-based algorithms in 2001[12]. Although the edge-based approach ranked second only slightly behind the accuracy of the hybrid-based approach, it provides much higher efficiency. Under this consideration, we choose the edge-based approach proposed by Leacock and Chodorow in 1998. With this approach, the similarity between two words can be calculated using the following equation and words cannot be found in WordNet will be ignored.

$$Sim(w_1,w_2)=-\log\frac{\left[\min\limits_{c_1\in sen(w_1),c_2\in sen(w_2)} len(c_1,c_2)\right]}{2d_{\max}} \quad (3)$$

where *sen(w)* denotes the set of possible senses of word *w*, d_{max} is the maximum depth of the taxonomy in WordNet. The *len*(c_1, c_2) function is the calculation of shortest path's length between c_1 and c_2 in the taxonomy. Fig. 4 shows a segment of the hyponymy taxonomy in WordNet. Assuming c_1 is Car and c_2 is Bicycle, the *len*(c_1, c_2) then equals 5.

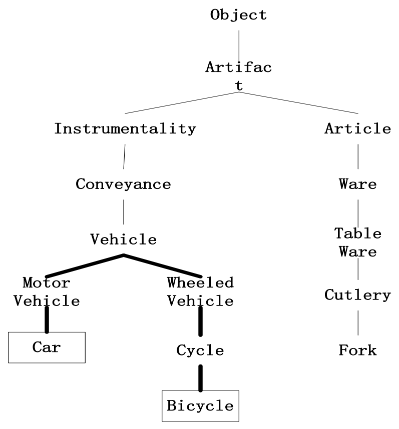


Figure 4. A segment of the hyponymy taxonomy in WordNet

D. Clustering

With the methods introduced above, the similarity value of any two services can be calculated in *Equation* (*1*). We use an $N\times N$ matrix to represent the similarity value between any two of N services. In this section, we introduce a clustering algorithm [13] to cluster similar web services based on the similarity matrix.

A clustering algorithm usually needs to consider the following three factors:

- Number of categories. This usually can be set as a parameter of clustering algorithm and be determined by the algorithm itself.
- Data processing method. This involves whether to load all the data at one time or in an incremental manner.
- Evaluation of category. This is about the process of evaluating whether an obtained category will be accepted or not.

Considering the large number of web services and the categories are unknown in advance, we present a top-down autonomous incremental clustering algorithm and our algorithm uses a category utility function to evaluate the obtained categories. It clusters data while reading it orderly. When the first data has been read, it is added to a category. The following data will be added into appropriate category by the algorithm which will decide whether to split or merge corresponding categories according to category diameter. This procedure will be repeated until all the data have been processed. Finally, a category tree can be generated to represent cluster result.

The pseudo code of the algorithm is shown in Fig. 5. In these pseudo code, "getNearestService($W_{i,}$, *W[]*, *n*)" denotes getting service which is nearest to W_i from service array *W[]*. And "getNearerService(W_i, *W[]*, *n*)" denotes getting service which is secondly nearest to W_i from service array *W[]*. These two methods rely on the distance between services which is calculated according to service similarity computing defined in Section III.C. "getCluster(W_i)" denotes getting cluster which service W_i belongs to. "*T*.add(W_i)" denotes adding service W_i to cluster *T*. "update(T_j)" denotes updating according to diameter, core and verge of *T* and deciding whether to split

```
Clustering Algorithm
Input:  service array: W[ ], array length: n,
        root of category tree: T,
        similarity matrix S[][]
Cluster (W[ ], n, T, S[][])
1.  for i ← 1 to n do
2.      $W_i$ ← getNearestService ($W_i$, W[ ], n, S[][]);
3.      $T_j$ ← getCluster ($W_j$);
4.      $W_k$ ← getNearestService ($W_i$, W[ ], n, S[][]);
5.      if getDistance ($W_k$, Wi) > $D_{max}$
6.          $W_k$← NULL;
7.      endif
8.      $T_k$ ← getCluster ($W_k$);
9.      if !exist ($T_j$)
10.         $T_j$ ← new Cluster ();
11.         $T_j$.add ($W_i$);
12.         addClusterToTree($T_j$, T);
13.     else if !exist ($T_k$) || $T_k$ == $T_j$
14.         $T_j$.add ($W_i$);
15.         update ($T_j$);
16.     else if  $T_k$.size () == 1
17.         $T_j$.add ($W_i$);
18.             if $T_j$.core == $W_i$
19.                 $T_j$← merge ($T_j$, $T_k$);
20.             endif
21.             update ($T_j$);
22.     else
23.         $T_j$.add ($W_i$);
24.         if $T_j$.core == $W_i$ && $T_k$.core != $W_k$
25.             $T_j$.add ($W_k$);
26.             update ($T_k$);
27.         endif
28.         update ($T_j$);
29.     endif
30. endfor
```

Figure 5. The pseudo code of the clustering algorithm

In this clustering algorithm, there are three issues to be addressed.

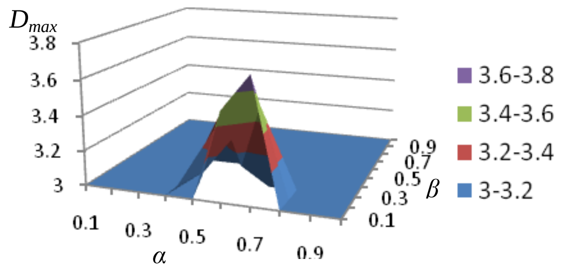
- Disturbance caused by the sequence of data.
- Calculation of the category utility.
- Over-fitting avoidance.

The corresponding solutions in this algorithm are:

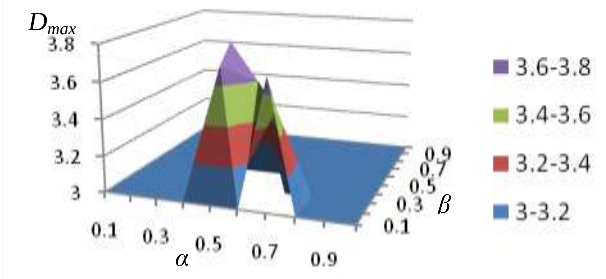
- Appropriate splitting and merging algorithm are used to eliminate disturbance from data sequence.
- Categories are treated as a sphere and the diameter is considered as category utility [14]. It means that, the smaller diameter, the better category utility it is.
- An upper bound for diameter is introduced to make category is not split unless its diameter is bigger than this bound.

The following definitions interpret three basic concepts used in this clustering algorithm.

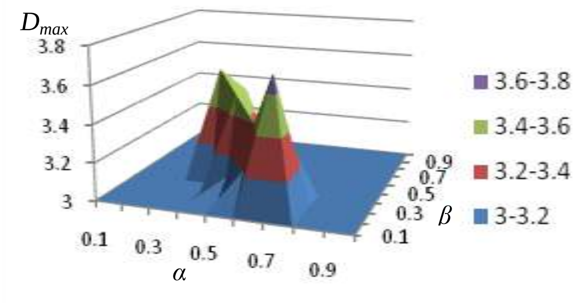
Definition 5*:* Diameter and core of a category: $\{n_1,n_2,..n_{|n|}\}$ are points in category *A*. Suppose when n_i is seemed as the



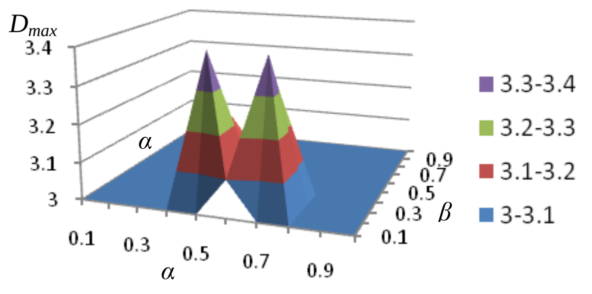
a. Precision in the scope of 0.7 to 0.71



b. Precision in the scope of 0.71 to 0.73



c. Precision in the scope of 0.73 to 0.75



d. Precision above 0.75

Figure 6. Combinations of α, β and D_{max}

core of this sphere, d_i is corresponding diameter. If d_i is smaller than any other diameter of spheres whose core is any other point in $\{n_1,n_2,..n_{|n|}\}$, then n_i is defined as the core of category A and d_i is defined as the diameter of the category A.

Definition 6: Maximum allowable diameter of a category: Let D_{max} denotes the maximum allowable diameter of a category. D denotes the diameter of a category. If $D > D_{max}$, The algorithm chooses two farthest points as the cores of two new categories, adds the remaining points into the two new categories and calculates diameters for two new categories. If diameter is bigger than D_{max}, then continue splitting on the category until diameter is smaller than D.

IV. EXPERIMENTAL EVALUATION

We design two experiments to evaluate the effectiveness and precision of our clustering and similarity computing algorithms. In the first experiment, we use the clustering algorithm proposed in Section III to cluster the candidate services through varying α, β, γ and D_{max} that determine the clustering results. In the second experiment, we apply our clustering algorithm to 1,566 services with appropriate α, β and γ that are specified according to the evaluation result of the first experiment.

The experiments are performed on a computer with Intel Pentium D 3.4GHz CPU, 2GB memory and the Windows XP SP2 OS. The algorithms are implemented in Java programming language.

A. Precision evaluation

1) In order to evaluate the precision of our system, we collect 69 web services and corresponding description information from Xmethods[15]. We extract 603 keywords from these services and only 5 of them cannot be found in the WordNet. All the candidate 69 services have been clustered manually by Xmethods as shown in Table I. Therefore the precision of our methods can be observed by comparing the clustering results with the results obtained manually, and we can speculate the appropriate scope of α, β, γ and D_{max}.

The precision we use in this experiment is defined as follows:

Definition 7: Assume there are M services, and these services are clustered into N categories (i.e. $C_1, C_2\ldots C_N$) by

TABLE I. CATEGORIES OF THE 69 WEB SERVICES

Category Name	Num. of services
Calendars&Economy	4
Corporate & Earnings	8
Currency	5
Date&Time	3
Email	2
Fax	3
Graphics & Multimedia	4
Instant messaging	2
Market data	8
Medical	6
SMS	5
Sports Info	4
Stock Info	11
ZipCode	4

a clustering algorithm, the clustering precision p can be calculated by the following equation:

$$p = \sum_{i=1}^{N} \frac{CorrectNum(C_i)}{M} \quad (4)$$

where $CorrectNum(C_i)$ denotes the number of services that really belong to C_i . Note there might be some services that are mistakenly clustered into C_i by the clustering algorithm.

In practice, our clustering algorithm is meaningful only when the precision determined by α, β, γ and D_{max} reaches a threshold T_p. According to this, we will only analyze combinations of α, β, γ and D_{max} that make precision reach T_p. Through experiments of all optional combinations, the best precision we can get is 0.821, so we set T_p. to be 0.7.

2) Precision evaluation: We tune α, β from 0 to 1 and vary D_{max} from 0.1 to 8, combinations make the precision greater than T_p are shown in Fig. 6.

From the result, we can obtain that, the precision is particularly high when α is around 0.6, which means title makes more contribution to the resulting similarity. This is accorded with the feature of WSDL which is a structured file. However, we do not find similar tendency of β and γ.

It can be observed that when D_{max} is in the scope 3.1 to 3.7, the clustering precision reaches a high level. In these experiments, there are three typical values of D_{max}: the lower bound (D^l), the optimum value (D^*), the upper bound (D^u).When $0 < D_{max} < D^l$, as D_{max} is less than the practical category diameter, each service is classified into one category. We call it over-fitting, and in this case the precision is 0. When $D^l < D_{max} < D^*$, as D_{max} moves closer to the practical category diameter, similar services are classified into one category. And the precision increases rapidly and then reaches the maximum. When $D^* < D_{max} < D^u$, as D_{max} deviates the practical category diameter, non-similar services are also classified into one category, the precision declines rapidly. When $D_{max} > D^u$, all services are classified into one category, the precision keeps a very small value.

3) Experiments comparision: We compare our result to that in paper [19]. It also focuses on reducing the searching space of service discovery through service clustering. It downloads 8 services from Internet, clusters them using approach methioned in paper [19] and gets a accuracy of 0.7. In our experiment, we get a precision of 0.721 on average. The highest precision we can get is 0.82.

B. *Effectiveness Evaluation*

1) In order to evaluate effectiveness of WSSM, we collected 1566 web services from the internet, including WSDL files and additional information. And only 73 of 11,372 keywords are missing in the WordNet. As clustering so many services manually is a time-consuming work, this experiment will only focus on effectiveness of reducing search space in service discovery.

Suppose a user want to find a service that is most similar to service W from M services and there is no clustering in advance, the average search complexity is $O(M)$. However, if there is a clustering processing (assuming that there are N categories, each category has Q_i services and the clustering precision is p), the search algorithm will first locate the appropriate category by comparing distance between service W and the category core, then search each service in the target category. As a result, the average search complexity will be $O(K)$, where K is:

$$K = N + \left[\sum_{i=1}^{N} \frac{(Q_i - 1)^2}{M} \right] \quad (5)$$

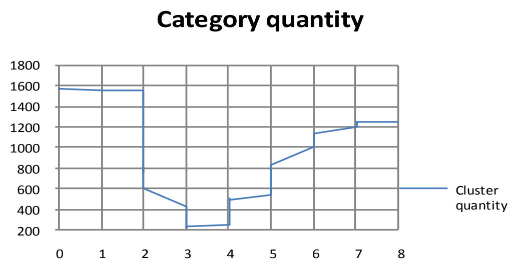


Figure 7. Cluster quantity vs D_{max}

2) Effectiveness evaluation: From the first experiment, we can know that the title is more important for similarity computing. So let $\alpha = 0.6$, $\beta = 0.2$, $\gamma = 0.2$, we tune Dmax from 0.1 to 8 to evaluate the effectiveness of the algorithm. The category quantity result is shown in Fig. 7.

We get the smallest category quantity 218, when $D_{max} = 3.5$. According to *Equation (5)*, the search times can be reduced to 225.2, which is a significant improvement from 1566. In addition, if we know a user is interested in service A, we can “push” the most similar service to the user by searching just in the same category.

V. RELATED WORK

There are some related works about WordNet-based service similarity computing and clustering. The paper [17] clusters tags offered by users in a music sharing website through an algorithm based on co-occurrence of words. This approach cannot deal with synonym, hypernym and hyponym effectively, because these words have few overlapped character. In paper [18], Eleni Stroulia and Yiqiao Wang combine WordNet-powered vector space model with semantic structure matching. Although this work contributes to find candidates accurately according to the query, it neglects the searching space which can be rather huge with the rapid growth of available services. In addition, this work just focuses on WSDL but drops other information including description and tags which are becoming a fashion in many websites. The paper [19] also only considers operations, parameters in WSDL and ignores weight of different part in WordNet-based service similarity computing.

VI. CONCLUSIONS

In this paper, we propose a web service similarity mining scheme called WSSM to support efficient service discovery.

We first define a service description model based on a triple consisting of three word collections. On this basis, we put forward a WordNet-based similarity mining algorithm to perform autonomously and incrementally clustering on services. Firstly, we extract service functional information from WSDL documents, service containers and web pages. Based on this information, similarity values between services are computed and a similarity matrix is generated as a result. Then a clustering algorithm is designed to cluster all the services based on this similarity matrix. Moreover, we perform extensive experiments based on real web services collected from Internet to evaluate the performance of WSSM, and the results show the efficiency of WSSM on service discovery.

Future research work will lead into the following three directions. Firstly, in this paper we only consider the nouns in computing the similarity values and we will consider verbs in future to further improved the clustering precision. Secondly, we will consider merging some categories with fewer services during clustering and this is expected to further reduce the average search times. Thirdly, we will develop a whole service discovery system based on WSSM including semantic structure service matching and so on.

ACKNOWLEDGMENT

Supported by the National High Technology Research and Development Program of China (863 program) under Grant 2007AA010301, 2006AA01A106 and 2009AA01Z419.

REFERENCES

[1] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pal Krogdahl, Min Luo, and Tony Newling, “Patterns: Service Oriented Architecture and Web Services”, IBM, 2004.

[2] Massimo Paolucci , Takahiro Kawamura , Terry R. Payne, and Katia Sycara, "Importing the Semantic Web in UDDI", “Web Services, E-Business, and the Semantic Web”, Springer, 2002:815-821.

[3] Kaarthik Sivashanmugan, Kunal Verma, Ranjit Mulye, and Zhenyu Zhong, “SpeedR: Semantic Peer-to-Peer Environment for Diverse Web Service Registries”, CSCI, 2002.

[4] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang, “Similarity Search for Web Services”, 30th VLDB, 2004

[5] M.F Porter, “An algorithm for suffix stripping”, Program, 14(3):130-137, 1980.

[6] Vladimir I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals”. Soviet Physics Doklady, 1966:707–710.

[7] Yuhua Li, David Mclean, Zuhair A. Bander, James D. O’Shea, and Keeley Crockett, “Sentence Similarity Based on Semantic Nets and Corpus Statistics”. IEEE Transactions on Knowledge and Data Engineering 18, 2006:1138-1150.

[8] Rahul Malik, L. Venkata Subramaniam, and Saroj Kaushik. “Automatically Selecting Answer Templates to Respond to Customer Emails”. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 2007:1659-1664.

[9] Claudia Leacock and Martin Chodorow, “Combining local context and WordNet similarity for word sense identification”, Fellbaum, 1998:265–283.

[10] Philip Resnik, “Using Information Content to Evaluate Semantic Similarity in a Taxonomy”, Proceedings of the 14th International Joint Conference on Artificial Intelligence, Vol. 1, 448-453, Montreal, August 1995.

[11] Jay J. Jiang and David W. Conrath, “Semantic similarity based on corpus statistics and lexical taxonomy”, Proceedings of International Conference on Research in Computational Linguistics, Taiwan, 1997.

[12] Alexander Budanitsky and Graeme Hirst, “Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures”, Computational Linguistics, 2006:13--47.

[13] Leonard Kaufman and Peter J. Rousseeuw. “Finding Groups in Data: An Introduction to Cluster Analysis”. John Wiley& Sons, New York, 1990.

[14] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, “BIRCH: an efficient data clustering method for very large databases”, Proceedings of the 1996 ACM SIGMOD international conference.

[15] Xmethods, http://www.xmethods.net, [August 30, 2009].

[16] Wordnet, http://wordnet.princeton.edu/, [August 20, 2009].

[17] Ya-Xi Chen, Rodrigo Santamaria, Andreas Butz, and Roberto Theron. “TagClusters: Semantic Aggregation of Collaborative Tags beyond TagClouds”, 9th International Symposium on Smart Graphics, SG 2009, Salamanca, Spain, May 28-30, 2009. Proceedings.

[18] Eleni Stroulia and Yiqiao Wang. “Structural and semantic matching for assessing web-service similarity”, International Journal of Cooperative Information Systems, 2005: 407-437.

[19] Aparna Kondurib and Chien-Chung Chana, “Clustering of Web Services Based on WordNet Semantic Similarity”, University of Akron, 2008.