2008

A Matching Algorithm for Selecting Web Services Based on Non-Functional Features

Said Elnaffar
UAE University, elnaffar@uaeu.ac.ae

A Matching Algorithm for Selecting Web Services Based on Non-Functional Features

Said Elnaffar
UAE University
elnaffar@uaeu.ac.ae

ABSTRACT

Searching for a Web service that meets the user requirements can be a complex task especially when the system starts to scale up by increasing the number of Web services, *w*, in the UDDI registry and by enlarging the number of QoS features (*f*) by which each Web service is described. This can be perceived as the commonly known nearest neighbor search problem, which typically imposes a time or storage complexity that is exponential in *f*. In this work, we present a new algorithm (wsSVD) that is founded on the algebraic matrix operation called Singular Value Decomposition (SVD). The basic idea is to encode the features of each Web service by a single value using the SVD. When a user seeks a Web service based on some specific requirements, these requirements get encoded by a single value using the same algorithm, and the matching process takes place in order to find the closest Web service that fulfills the user requirements. Our experiments show that the wsSVD algorithm performs and scales up well in comparison with other matching algorithms.

Keywords

UDDI, nearest neighbor search, Singular Value Decomposition, SVD, QoS, search algorithm

INTRODUCTION

In the Service Oriented Architecture (SOA), the W3C's Web services are deemed new and standard way of communication between software applications. To that end, many XML-based standards came into play in order to ensure that Web services support interoperable computer-to-computer interaction over a network such as the Internet. Among these standards is the UDDI (Universal Description, Discovery, and Integration) standard, which allows providers to advertise and publish their Web services at UDDI registries so that Web services can be searched by prospective users (or applications).

We view any user inquiry about an appropriate Web service going through two stages. In the first stage, *functional-based search*, the system finds a set of Web services that functionally satisfy the user's need. For example, if a user seeks a suitable Web service that can assist her to book a hotel, then the system should search the registry for all Web services that handle HotelBooking. In the second stage, *non-functional based search*, which is the focus of this work, the system seeks the HotelBooking Web services that are the closest to some minimum QoS requirements (e.g., throughput, computing power, etc.) specified by the user. Usually, user inquiries are handled by an agent who searches the UDDI registry using some matching algorithms.

This paper is concerned with Web services search and inquiry optimization based on non-functional features. Formally, if we assume that the Web service agent maintains a list of *w* services, where each Web service *i* (WS_i) is described by a vector of *f* features ($a_{i1}, a_{i2},\ldots, a_{if}$), then the goal is to find the WS_x that is the nearest to some minimum user requirements represented by the vector ($r_{i1}, r_{i2},\ldots, r_{if}$). WS_x must satisfy the feasibility condition

$$a_{xy} \geq r_{xy}\ ,\ \text{where}\ \ 1 \leq y \leq f \qquad (1)$$

Examples of Web service features are memory size, CPU cycles, disk capacity, network bandwidth, and I/O bandwidth. The values of these features are deemed the minimum requirements that the user can accept and are assumed to be proportional contributors to the quality of the Web service; the more powerful the CPU is, for example, the better the Web service is. Therefore, if a feature is inversely proportional to the quality of the Web service, as it is the case with response time, the reciprocal of this feature should be considered. The feasibility condition of Eq. (1) stresses the fact that it is not good enough

for the search algorithm to find the nearest match to the user requirements but also none of its features should be less than the corresponding feature value in the user requirements. This feasibility condition adds extra layer of complexity as we will explain soon.

This problem is a threefold challenge. First, the search complexity can scale up easily given that published Web services are incessantly on the rise and the number of features by which a Web service can be described can be considerably large. Second, it has the typical challenges associated with any nearest neighbor search problem. Third, it has the extra difficulty that the value of each of the Web service feature must be no smaller than the value of the corresponding feature in the user requirement. Furthermore, an effective solution is essentially needed when the user is on the go using her mobile or PDA in order to access a Web service at real time. Intuitively, the Web services agent has two obvious search algorithms:

Random. In this algorithm, the agent keeps picking a random Web service until it finds one that satisfies the feasibility condition (Eq. 1). This approach might be superb with respect to efficiency, measured by the number of probes the agent need to conduct, but it pays no attention to the quality of the chosen Web service.

Brute Force. In this algorithm, the agent exhaustively searchers the entire list of Web services looking for the nearest match, however, it is a very costly search technique as the complexity is O(wf), where w can be thousands of Web services, and f can plausibly be tens of features especially if, for example, the various kinds of hardware and software packages that exist nowadays are considered among the features.

In this work, we present an approach in which we encode the entire vector of features of each Web service by a single value using the Singular Value Decomposition technique (SVD) (Golub and Van Loan 1989; Skillicorn 2007). Similarly, the desired vector features specified by the user is encoded and subsequently matched by searching the list of encoded Web services. The overall complexity of the matching algorithm is $O(f \log w)$, where $\log w$ is caused by the binary search of the list. There is an obvious geometric interpretation of the problem in which each Web service and each user request are points in an f-dimensional space. The goal is to find the nearest neighbor of the user request – but with the extra complication that the value of each of the Web service features must be no smaller than the value of the corresponding feature in the user's request. For comparison purposes, we compare our algorithm (wsSVD) with two other algorithms: the wsSUM algorithm that encodes features by summing them up, and the wsRANDOM that randomly selects any feasible Web service.

The rest of this paper is organized as follows. The following section reviews some research body pertaining to Web services searching and matching. Section 3 gives a high level introduction to the SVD. Section 4 explains the wsSVD matching algorithm that is based on the SVD technique. Section 5 details our experimental setup and summarizes the results. In the last section, we conclude the paper and sketch the directions of our future work.

RELATED WORK

The literature is rich in recommender and matching systems germane to Web services that are mostly based on profiles, keyword searching, and functional descriptions (Aimeur, Benslimane, Daboue, Maamar and Onana 2007; Shaban and Haarslev 2005; Sajjanhar, Hou and Zhang 2004; Dong, Halevy, Madhavan, Nemes and Zhang 2004; Kawamura, Blasio, Hasegawa, Paolucci and Sycara 2004; Martin, Paolucci, McIlraith, Burstein, McDermott, McGuinness, Parsia, Payne, Sabou, Solanki, Srinivasan and Sycara 2004; Schade, Sahlmann, Lutz, Probst and Kuhn 2004; Wang and Stroulia 2003; Limthanmaphon and Zhang 2003). For example, (Aimeur et al. 2007) propose a recommender system that attempts to match a user profile with its counterpart from Web services. The user profile summarizes the user's interests, preferences, usage records, purchase records, browsing behavior, etc. This recommender system uses demographic, collaborative, and content-based filtering techniques.

(Dong et al. 2004) contend that traditional keyword search is insufficient because the very small text fragments in web services are unsuitable for keyword search, and the underlying structure and semantics of the web services are not exploited. So they introduce the Woogle search engine whose algorithms are primarily based on finding similarities among Web services using clustering techniques. (Kawamura et al. 2004) present another system with four filters to refine the Web services search results.

More improvements are proposed by (Shaban and Haarslev 2005) where they describe a prototype of an agent that uses OWL-S (Ontology Web Language for Services) (Martin et al. 2004) ontologies for reasoning in order to produce better results of similarities between Web services. In almost the same direction, several other researchers try to exploit semantic information that lies in the infrastructure of the WSDL/UDDI files. For example, (Limthanmaphon and Zhang 2003) use CBR (Case-Based Reasoning) to search for Web services. (Wang and Stroulia 2003) propose the "query by example" process, in which partial description of the Web service is provided as an input to the system. From that partial description, keywords are extracted and compared with textual information of other Web services. The system comes out with Web

services having similarity values higher than a certain threshold. The resulting set of Web services is then refined using the structure-matching techniques applied to the WSDL documents.

The above research works share one common goal which is searching for Web services based on their functionality (e.g., HotelBooking) description. Our research objective, however, is different in the sense that we try to search for Web services based on their non-functional (Second Stage described earlier), QoS-based properties such as availability. We assume that our search space already agglomerates Web services that share the same functionality (First Stage's outcome).

Finding the nearest Web service that matches a minimum set of requirements stipulated by the user can be a challenging problem. If we let the number of the candidate Web services listed in the UDDI registry be w, and the number of features (properties) that describe each Web service be f, it is not unrealistic for w to be in the order of thousands and for f to be in the order of tens. The intuitive geometric interpretation of this problem is to view the Web services and the user requirements as points modeled in an f-dimensional space. Our goal then is to find the nearest neighbor candidate Web service point to a given user requirements point, but with the additional challenge of satisfying the feasibility condition that entails the value of each of Web service feature must be no smaller than the value of the corresponding feature specified by the user requirements.

For a small number of features inexpensive solutions exist. For example, if there is only one feature (f=1), then a binary search with complexity O(log w) would suffice. If the Web service is described by two features (f=2), we can construct a Voronoi diagram for the set of points and using any fast planar point location algorithm we can locate the cell that contains the user requirements point. However, complexity starts to grow exponentially once f starts to be larger (e.g., f>10). An obvious algorithm to find the best positive nearest neighbor is to use brute-force exhaustive search. This is expensive given that the complexity is O(fw) for potentially large f and w, and given that the Web service inquiry can be issued from a mobile device or PDA where responsiveness is a priority.

As shown in (Dobkin and Lipton1976), the upper bound for the time needed to finding the exact positive nearest neighbor is O(2f log w) and $O(w^{2^{f+1}})$ for storage. Subsequent improvements (e.g., Matousek 1991; Agarwal and Matousek 1992) appeared later on to lower the search time to $\Omega(g(f)\log w)$,where $g(f)$ usually hides an exponential function in f. Another effort to find the positive nearest neighbor was based on the *k-d* tree (Samet 1990), which splits the search space into smaller regions that contain a small number of points (Web services). To find the best match for a given user requirements, the k coordinates of the requirements are used to locate the region that should contain the sought Web service. Afterwards, an exhaustive search is conducted in order to find the closest Web service. The *k-d* trees are efficient with low dimensions, however, the search time increases exponentially upon increasing dimensionality. Further improvements were achieved by shifting the focus from finding the exact positive nearest neighbor to seeking an approximated solution (e.g., Clarkson 1994; Kleinberg 1997; Arya, Mount, Netanyahu, Silverman and Wu1998). However, these approximation algorithms demanded non-trivial storage space and performed poorly when f>log w.

In this work we use the Singular Value Decomposition (SVD) (Golub and Van Loan 1989) aiming to make the complexity as low as O(f log w) by using it as a dimensionality reduction technique (Skillicorn 2007). (Sajjanhar et al. 2004) used SVD to select Web services based on search keywords. The authors argue that searching UDDI registries using traditional keyword search techniques is not effective as these techniques overlook the relationships between Web services. So they used SVD to reveal semantic relationships among Web services in order to retrieve functionally-related ones. Our work is different from Sajjanhar et al.'s work as we try to find the appropriate matches at the non-functional level assuming that all inquired Web services already have similar functionality but with various non-functional properties. A similar work to ours was done by (Roumani and Skillicorn 2004) where they used SVD in the Grid computing context. Through our experiments, we have found that their reported results were occasionally disappointing due to improper data preprocessing. Practically, we observed that normalizing the original dataset is indispensible in order to avert the shortcomings in (Roumani and Skillicorn 2004). Normalizing the dataset is crucial before applying SVD because SVD is principally based on the variation of data around the origin. Without normalization, features with larger values will have larger variations and subsequently undesirable, false influence over the remaining features.

Next, we give a brief description of SVD and shed the light on the geometric interpretation that we exploit to rank Web services based on the specified user requirements.

SINGULAR VALUE DECOMPOSITION (SVD): OVERVIEW

Let A be a matrix corresponding to a dataset. The *singular value decomposition* is a decomposition that rotates the original space in such a way that the variance is maximized along the first axis; the remaining variance is maximized along the second axis and so on. So the SVD of matrix A is formally defined as

$$A = B\mathrm{W}N' \qquad (2)$$

where

A is w x f

B is w x v,

W is a v x v diagonal matrix with non-increasing entries $\beta_1 \geq \beta_2 \geq \ldots \beta_{v-1} \geq \beta_v > 0$ (called the singular values)

N' is a v x f matrix representing the transpose of N.

In addition, both B and N are orthogonal, i.e., BB'=I and NN'=I. In practice, v is usually chosen to be equal to f.

For the purpose of our problem we use the more obvious geometric interpretation of the SVD, but the *component* and *factor* interpretations can be useful too (Skillicorn 2007). If we model the features of w Web services in an f dimensional space, and represent them by the (w x f) matrix, A, then the SVD can be seen as a transformation of the original data represented in the original geometric space, A, into a new set of data, B, represented in a new geometric space that has the following property: the maximum variation in the original data is explained by the Web services points that lie along the first dimension of the new space; the maximal variation remaining is explained by the Web services points that lie along the second dimension, and so on. Therefore, and in the context of our problem the matrices shown in Eq. 2 can be interpreted as follows:

A: is the w x f matrix that describes a list of w Web services in the original space of f dimensions (features).

B: is the w x v matrix that describes the same list of w Web services in the new space of v dimensions. That is, each row of B gives the coordinates of the corresponding row of A in the coordinate system of the new axes (which are defined by N').

N': is the v x f matrix that describes the v axes of the new space in terms of the original f axes of the original space.

W: is a v x v positive diagonal matrix whose singular values are non-increasing, and they represent the weight and significance of each of the new axes.

In general, the storage complexity required to store the data structures during the computation of the SVD of a matrix is $O(fv + v^2 + vw)$. The time complexity is $O(w^2f + f^3)$, however, since the number of features of a Web service, f, is much smaller than the number of the Web services (w), the complexity in practice approaches $O(w^2f)$. This is not a concern because computing the SVD of the Web services dataset is part of the preprocessing phase that takes place offline. Furthermore, one of the most useful properties of an SVD is that the matrices on the right hand side can be truncated at the k^{th} largest singular values of W, and the corresponding k columns of B and rows of N'. In fact, this technique is deemed one of the other utilities of using the SVD method in order to map the data model from a space of high dimensionality to another one with lower dimensionality, yet the latter still accounts for most of the variations in the data. Therefore, we can re-write Eq. (2) as follows:

$$A_k = B_k\, W_k\, N_k' \qquad (\mathbf{3})$$

Where the matrix A_k denotes the best rank-k approximation to the original matrix. In the next section, we describe our Web services matching algorithm, wsSVD, that is based on the SVD.

1. *Phase I (Off-line): Preprocessing*
    a. *Normalize A using z-scores*
    b. *Compute SVD of $A_{k=1}$ (i.e., truncated to the first dimension using Eq. 3)*
    c. *Compute column matrix $B_{k=1}$ using Eq. 4*
    d. *List L = $B_{k=1}$*
    e. *Sort L that has w values of B_1*
2. *Phase II (On-line): Matching*
    a. *Receive some user requirements vector $R=(r_{i1}, r_{i2}, \ldots, r_{if})$*
    b. *Compute the SVD for the row matrix R.*
    c. *Search for the single value encoding of B_1 in the sorted list (L) using binary search:*
        i. *If the found WS_x satisfies all f features, then return WS_x*
        ii. *Else, do zig-zag sequential searching on both sides of WS_x until* (i) *is satisfied, else Stop.*

Figure 1. The *wsSVD* Algorithm

THE wsSVD ALGORITHM

Based on the SVD method, we introduce the wsSVD algorithm for solving the positive nearest-neighbor matching problem. As sketched in Figure 1**Error! Reference source not found.**, the algorithm has two phases: the *off-line preprocessing phase*, and the *on-line matching phase*. In the data preprocessing phase, we normalize the matrix A, that represents the original dataset of Web services, and compute its SVD by truncating the results to one dimension. By re-arranging the SVD decomposition equation we get

$$B_k=A_kN_kW_k^{-1} \qquad (4)$$

That is, the Web services that were represented by rows in A can be mapped into the transformed space (B) by multiplying them by NW^{-1}. Since we truncated the SVD to one dimension (k=1), this mapping requires only the first column of N and the first singular value, resulting in $O(f)$ time complexity.

The resulting list (L) is sorted by the increasing values, b's, of the first column of B. Each b_x in this list corresponds to Web service number x in the original matrix A after encoding its features in a manner that maximizes the variation among all Web services; i.e., it spreads them as far apart as possible along this new, first dimension in the new space. We practically observed that there is a large drop in the singular values (β's) of the diagonal matrix W beyond the first dimension (k=1), which makes one not concerned about losing major component[1] in data as a result of truncation.

In the second phase, *matching*, a request for a service arrives with some user specified requirements, represented by the vector R, which gets mapped (using the same SVD-based encoding explained above) into the corresponding space of B, and subsequently encoded into a single value. This single value is compared to the ranked list (L) of encoded Web services using binary search to find a Web service with the closest value. However, the nearest Web service (WS_x) may not be feasible, that is, at least one of values of its features is smaller than the corresponding user requirement. In this case, the ranked list is sequentially searched, to the left and the right of WS_x, to find the closest Web service point that is feasible.

[1] Due to this property, the SVD is one of the tools that can be used for dimensionality reduction.

Property	wsSVD	wsSUM	wsRANDOM
Storage Complexity	O(*nm*)	O(*nm*)	O(*nm*)
Time Complexity	O(*f* log *w*)	O(*f* log *w*)	O(*nm*)
Feasibility Check	O(*m*)	O(*m*)	O(*m*)
Preprocessing Cost	High	Medium	Low

Table 1. The Theoretical Properties of All Algorithms

EXPERIMNETS

Setup and Configuration

Due to the present lack of a standard way for storing QoS features of Web services, it is very challenging to obtain real datasets. Therefore, in order to implement the wsSVD algorithm and compare it with others, we generated artificial dataset of Web services with real-valued features. To have reasonably realistic values for the WS features, each feature is generated using the Poisson distribution based on different means. We also took care of limiting the percentage of feasible Web services that can be good matches to user requests, as we will explain shortly. With the careful procedure of data generation, we conjecture that the wsSVD should exhibit the same performance especially that we tested several randomly generated data set and got consistent concluding results.

For comparison purposes, we compare our algorithm with other two algorithms: wsSUM and wsRANDOM. The ultimate difference between the wsSUM and the wsSVD is in the way we encode and summarize the Web service features. In the wsSUM, the features of a Web service are encoded by linearly summing them up. Similarly, the requirements vector supplied by the user is encoded in the same manner, simple summation. The wsRANDOM finds a matching Web service by randomly selecting a Web service until it finds a feasible one, i.e., features are not encoded by any means. Note that this algorithm is not concerned with how near that Web service to the desired user requirements.

One of the advantages of using the wsSUM is its ability to promptly exclude the Web services whose encoded vector is less than the encoded value of the user requirements because these Web services cannot be feasible anyway. We should also note that the preprocessing time of the wsSVD is more expensive than the wsSUM's, however, this cost is not a concern at real time.

Through several experimentations, we found that normalizing the original dataset is an indispensable preprocessing procedure in order to cancel the undesirable effect of some features on the others. In our experiments we normalized the values of each feature using the z-score normalization. We trust that any range normalization (e.g., [0.0, 1.0]) would work too. Table 1 summarizes the theoretical properties of all algorithms. Notice that all algorithms require an O(*fw*) storage because the complete set of features must be always checked for feasibility. In order to experimentally compare the performance of our algorithms, two metrics are reported:

Overhead. It reflects the cost of the matching algorithm measured by the number of probes needed to find a feasible Web service.

Unfitness. It is a ratio that reflects how poor the quality of the match found by a given algorithm with respect to the best match resulting from the exhaustive search. The similarity between two Web services is determined by calculating the difference in the Euclidean distance between the two points that represent these Web services. Therefore,

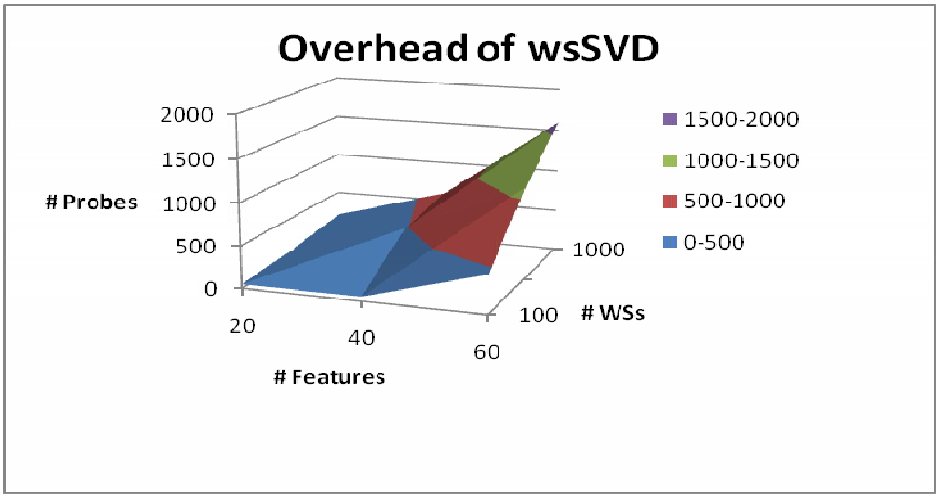
$$Unfitness = \frac{|\, d(u,\psi) - d(u,\alpha)\,|}{d(u,\alpha)} \qquad (5)$$

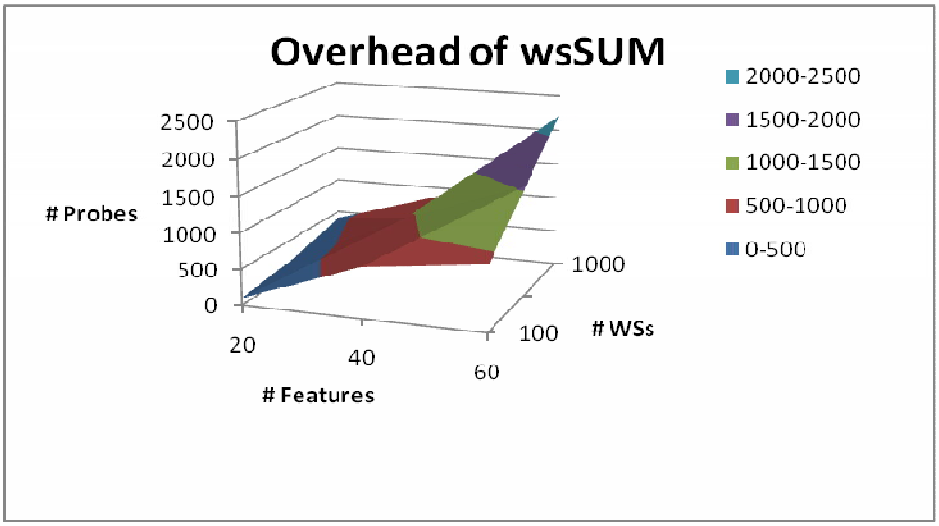
Where

$d(u, x)$: the function that returns the Euclidian distance between the user sought Web service, u, and the candidate Web service x.

ψ: a Web service point found by the wsSVD, wsSUM, or wsRANDOM.

α: the Web service point found by the exhaustive search and it is deemed a reference point.





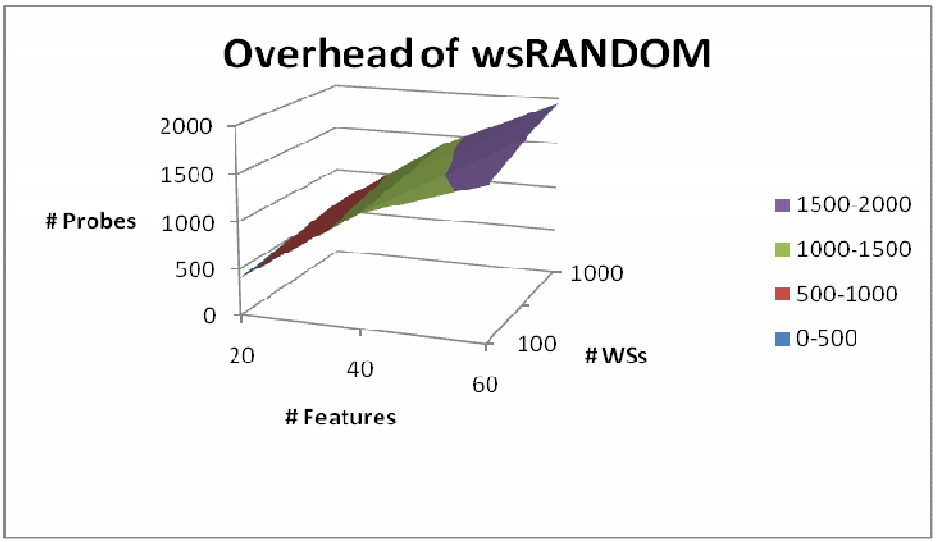


Figure 2. Search Overhead of Each Matching Algorithm (Lower is Better)

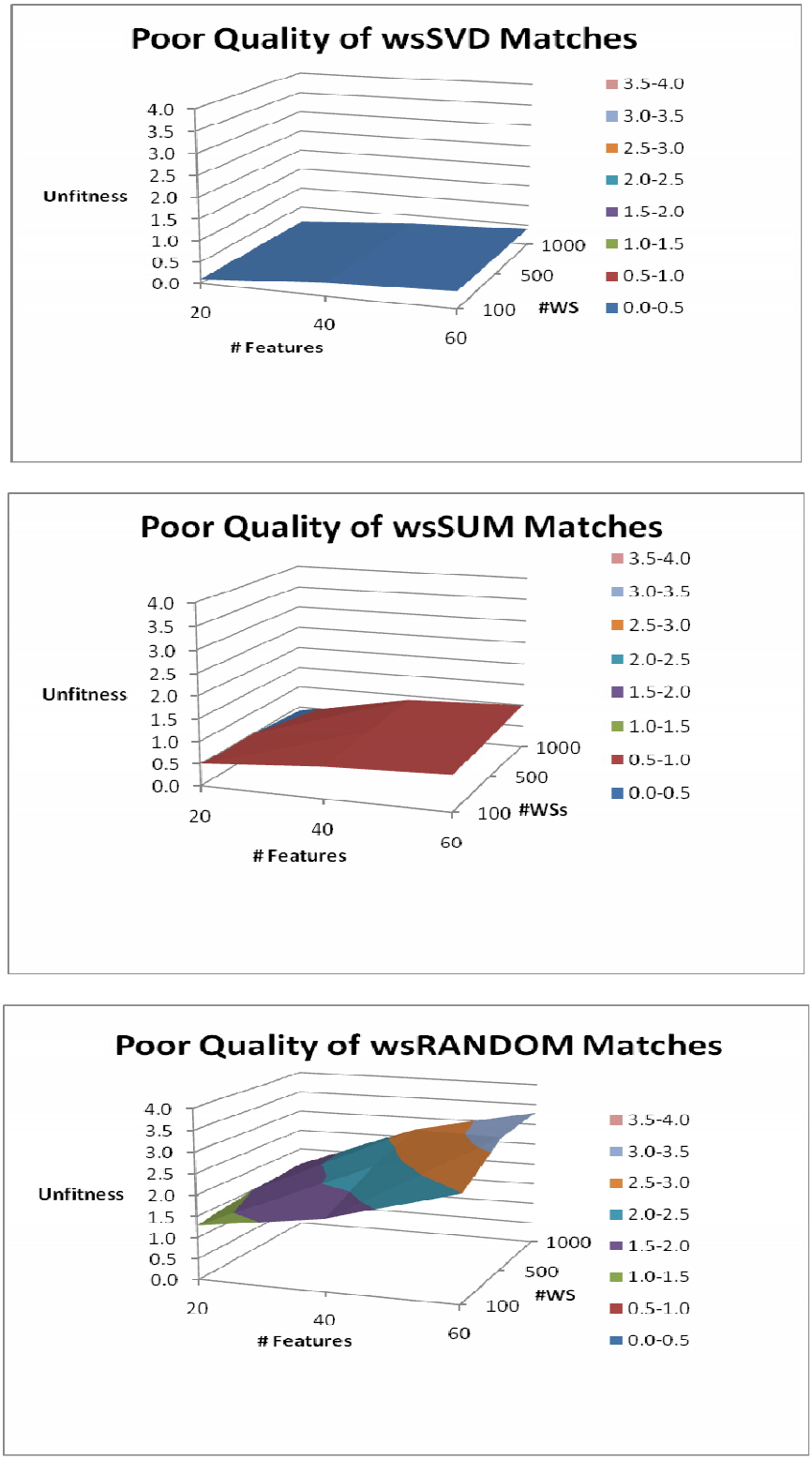


Figure 3. Poor Quality of Matches for Each Search Algorithm (Lower is Better)

Parameter	Range
Number of Web services (*w*)	100-1000
Number of features of each Web service (*f*)	20-60
% of feasible Web services	3-7%

Table 2. Simulation Parameters

Therefore, the higher the unfitness is for a given algorithm the poorer quality of matches that algorithm returns. We used Java and Jama library (JAMA 2008) to implement our experiments. The results of experiments are summarized over 100 runs. Table 2 lists the main parameters that control our experiments. In order to make our dataset more realistic and challenging to our algorithm, we limited the percentage of Web services that are deemed feasible in the range of 3-7%. If we made the percentage of feasible Web services low, then the exhaustive sequential algorithm would be the best choice. On the other hand, if there is a profusion of feasible Web services, which is unlikely, then the wsRANDOM would be the one to use.

The percentage of feasible Web services of 3-7% is ensured as follows. We randomly sample 3-7% of the Web services and generate their counterpart of user requests (requirements vectors). This leads to having a list of Web services that contain at least 3-7% feasible solutions for user requests. However, these user requests could be satisfied by more than 3-7% of Web services as these requests represent the minimum user requirements. So and in order to ensure an upper bound to the percentage of feasible Web services, we repeatedly check the feasibility percentage using these requests. If it is too high, we keep dropping some user request from the set until we get down to the desired feasibility percentage.

Experimental Results

The goal of our experimentation is to observe the performance of the three matching algorithms with respect to the search overhead (Figure 2**Error! Reference source not found.**) and the quality of the matching results (Figure 3**Error! Reference source not found.**). To that end, we examine the performance of each algorithm under different parameter values: number of Web services under investigation, and the number of features by which we describe each Web service. Figure 2**Error! Reference source not found.** illustrates the overhead incurred by each algorithm in order to find the nearest positive neighbor Web service that satisfies the minimum requirements specified by the user request. Fewer probes mean a better algorithm. Figure 3**Error! Reference source not found.** shows the unfitness ratio of the returned search result in reference to the result spawned from the exhaustive search; a lower unfitness ratio means a better matching algorithm.

As seen in the figures, wsSVD incurred lower number of probes and exhibited better matching quality than wsSUM and wsRANDOM do especially when the number of features and the number of Web services increase. It is not surprising to observe that the overhead of the search using wsRANDOM is, at some runs, less than the wsSUM because we should remember that wsRANDOM randomly searchers for the first Web service point that is feasible, not necessarily be the nearest to the user requirements vector. For that reason, the quality of wsRANDOM is the worst among all algorithms.

CONCLUSION

Searching the UDDI registry or a large Web services community for the nearest Web service that fulfills some minimum user requirements has a scalability challenge especially for real-time requests, e.g. from a PDA, where responsiveness is a priority. UDDI registries nowadays contain thousands of Web services. Each Web service can easily be described by tens of non-functional properties especially if we keep in mind the increasing number of different combinations of hardware and software that are rolled out in the computing market nowadays. In this work, we discussed how we used the SVD matrix decomposition technique to devise a search algorithm that can be less costly and return better quality matches especially when the search space scales up. We exploited the most important properties of the SVD: dimensionality reduction and the ability to summarize the majority of data variation by truncating the computation of the SVD to one dimension. The wsSVD algorithm shows good scalability in comparison with the wsSUM and wsRANDOM algorithms with respect to the overhead of the search and the quality of the returned results. The overall time and storage complexities of the wsSVD algorithm are $O(f \log w)$ and $O(fw)$, respectively.

As a future work, we wish to try our algorithm on real dataset of Web services. We also look forward to investigate the possibility of using the SVD technique to build the other type of Web services search, which is the functional-based search. To that end, we intend to scan the UDDI registry in order to build a community of homogenous Web services that share the same functionality. In other words, the SVD will be used as a clustering technique. This kind of search uses keywords and ontologies and is dependent on the semantics embedded in the WSDL files and in the associated web sites. In the subsequent phase of the search process, users can narrow down their search scope by specifying their non-functional requirements, which has been the focus of this paper. We plan to build a prototype in which we integrate the two search phases in one framework.

ACKNOWLEDGEMENT

We thank the Research Affairs at the UAE University for supporting this project financially under contract no. 03-06-9-11/07. We also thank Dr. Ali Roumani and Dr. David Skillicorn for their advices at the early stages of this research.

REFERENCES

1. Agarwal, P. and Matousek, J. (1992) Ray shooting and parametric search, *Proceedings of the 24th ACM Symposium on Theory of Computing*, 517–526.
2. Aimeur, E., Benslimane, D., Daboue, D., Maamar, Z. and Onana, F. S. M. (2007) WSRS: A Web Service Recommender system, *Proceedings of the 3rd International Conference on Web Information Systems and Technologies: Internet Technology / Web Interface and Applications*, Barcelona, Spain.
3. Arya, S., Mount, D., Netanyahu, N., Silverman, R. and Wu, A. (1998) An *optimal algorithm for approximate nearest neighbor searching in f*ixed dimensions, *Journal of the ACM*, 45, 6, 891–923.
4. Clarkson, K. (1994) An algorithm for approximate closest-point queries, *Proceedings of the 10th Annual Symposium on Computational Geometry*, 160–164.
5. Dobkin, D. and Lipton, R. (1976) Multidimensional search problems. *SIAM Journal on Computing*, 5, 181–186.
6. Golub, G. and Van Loan , C. (1989) Matrix Computations, The Johns Hopkins University Press, Baltimore, MD, USA, second edition.
7. Dong, X., Halevy, A., Madhavan, J., Nemes, E., and Zhang, J. (2004) Similarity Search for Web Services, *Proceedings. of the 30th VLDB (Very Large Databases) Conference*, 372–383, Toronto, Canada.
8. JAMA (2008) A Java Matrix Package, URL: http://math.nist.gov/javanumerics/jama/.
9. Kawamura, T., Blasio, J., Hasegawa, T., Paolucci, M., and Sycara, K. (2004) Public Deployment of Semantic Service Matchmaker with UDDI Business Registry, Proceedings of the 3rd Intern. Semantic Web Conference (ISWC 2004), LNCS 3298, 752–766, Hiroshima, Japan.
10. Kleinberg, J. (1997) Two algorithms for nearest-neighbour search in high dimensions*, Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 599–608.
11. Limthanmaphon, B. and Zhang Y. (2003) Web service composition with case-based reasoning, *Proceedings of the 4th Australasian Database Conf. on Database Technologies*, 201–208, Adelaide, Australia.
12. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N. and Sycara, K. (2004) Bringing semantics to Web Services: The OWL-S approach, *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA.
13. Matousek, J. (1991) Reporting points in half-spaces, Proceedings of 35th Symposium on Foundations of Computer Science, 207–215.
14. Paunovic, M. and Schlesinger, M. (1998) Fundamentals of Electrochemical Deposition, John Wiley & Sons, Inc.
15. Roumani, A. and Skillicorn, D. (2004) Large-Scale Resource Selection in Grids, *Proceedings of On the Move Federated Conferences (OTM): GADA*, Springer Press, vol. 3292, 154-164, Agia Napa, Cyprus.
16. Sajjanhar, A., Hou, J. and Zhang, Y. (2004) Algorithm for Web Services Ma*tching, Advanced Web Technologies and Applications/Lecture Notes in Computer Science. Springer Berlin* / Heidelberg. ISSN 0302-9743 (Print) 1611-3349 (Online), Volume 3007/2004, ISBN 978-3-540-21371-0*, 665-670.*
17. Samet, H. (1990) The Design and Analysis of Spatial Data Structures, Addison-Wesley, Longman Publishing Co., Inc., Boston, MA, USA.

18. Schade, S., Sahlmann, A., Lutz, M., Probst, F. and Kuhn, W. (2004) Comparing approaches for semantic service description and matchmaking, *Proceedings of the 3rd International Conference on Web service descriptions, Databases, and Applications of Semantics for Large Scale Information Systems,* 1062–1079, Larnaca, Cyprus.
19. Shaban, A. and Haarslev, V. (2005) Applying semantic Web technologies to matchmaking and Web service descriptions, *Proceedings of the Montreal Conference. on eTechnologies*, 97–104, Montreal, Canada.
20. Skillicorn, D. (2007) Understanding Complex Datasets: Data Mining with Matrix Decompositions, Chapman & Hall/Crc Data Mining and Knowledge Discovery Series, ISBN 1584888326.
21. Wang, Y. and Stroulia, E. (2003) Semantic Structure Matching for Assessing Web-Service Similarity, *Proceedings of the First International Conference on Service Oriented Computing*, Trento, Italy.