WSBen: A Web Services Discovery and Composition Benchmark

Seog-Chan Oh[a], Hyunyoung Kil[b], Dongwon Lee[c], and Soundar R. T. Kumara[a]

[a] *Department of Industrial and Manufacturing Engineering*
[b] *Department of Computer Science and Engineering*
[c] *College of Information Sciences and Technology*
Penn State University, PA 16802, USA
{seogchan, hkil, dongwon, skumara}@psu.edu

Abstract

A novel benchmark, WSBen, for testing web services discovery and composition is presented. WSBen includes: (1) A collection of synthetic web services (WSDL) files with diverse characteristics and sizes; (2) Test discovery and composition queries and solutions; and (3) External files for statistical analysis and AI planners. Users can fine-tune the generated WSDL files using various parameters such as skewness or matching type. It is our hope that WSBen will provide useful insights for researchers evaluating the performance of web services discovery and composition algorithms and softwares.

1. Introduction

A web service, *w*, specified in WSDL files, is similar to an API file having a list of functions with two sets of parameters: $w_{in} = \{I_1, I_2, ...\}$ for SOAP request (as input) and $w_{out} = \{O_1, O_2, ...\}$ for SOAP response (as output). When one has a request r that has initial input parameters r_{in} and desired output parameters r_{out}, one needs to find a web service w that can fulfill r such that (1) $r_{in} \supseteq w_{in}$ and (2) $r_{out} \subseteq w_{out}$. Finding a web service that can fulfill *r* alone is referred to as *Web Service Discovery (WSD)* problem. When it is impossible for one web service to fully satisfy *r*, one the other hand, one has to compose multiple web services $\{w^1, w^2, \ldots, w^n\}$, such that (1) for all $w^i \in \{w^1, w^2, \ldots, w^n\}$, w^i_{in} can be grounded when w^i_{out} is required at a particular stage in composition, and (2) $(r_{in} \cup w^1_{out} \cup \ldots \cup w^n_{out}) \supseteq r_{out}$. This problem is often called as *Web Service Composition (WSC)* problem.

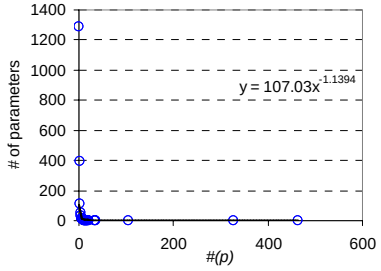
As a growing number of web services are available on the Web and in organizations, finding and composing the right set of web services become ever more important. As a result, in recent years, a plethora of research work and products on WSD and WSC problems have appeared[1]. In addition, the web service research community has hosted competition programs (e.g., EEE05[2], ICEBE05[3]) to solicit algorithms and softwares to discover pertinent web services and compose them to make value-added functionality. Despite all this attention, however, there have been very few test environments available for evaluating such algorithms and softwares.

Therefore, the need for a benchmark naturally arise to evaluate and compare algorithms and softwares for the WSD and WSC problems. In particular, the benchmark must have WSDL files and test queries that can represent diverse scenarios. Often, however, test environments used in research and evaluation have only skewed test cases that do not necessarily capture real scenarios. To demonstrate our claim, let us consider the following observation.

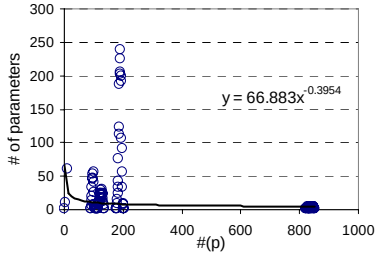
Observation. We first downloaded 1,544 raw WSDL files[4] that Fan et al. [5] gathered from real-world web services registries, such as xMethod or BindingPoint. After weeding out invalid WSDL files, we had 670

[1] At the time this paper was written, there were about 900 and 80 scholarly articles mentioning "web services composition" at Google Scholar and CiteSeer, respectively.
[2] http://www.comp.hkbu.edu.hk/~eee05/contest/
[3] http://www.comp.hkbu.edu.hk/~ctr/wschallenge/
[4] http://rakaposhi.eas.asu.edu/PublicWebServices.zip

valid WSDL files left. Then, we measured how many distinct parameters each WSDL file contained. Suppose that given a parameter, p, we denote the number of occurrences of p as #(p). That is, #("pwd") indicates the number of occurrences of the parameter "pwd". Figure 1(a) illustrates this distribution, where the x-axis is #(p) and the y-axis is the number of parameters with the same #(p) value. The distribution has no humps. We also plotted a power-function, $\alpha(1/k)^{\beta}$ over the #(p) distribution and found that the exponent β is 1.1394. Although 1.1394 does not suffice the requirement to be the power law [13], the distribution is skewed enough to be seen as the Zipf-like distribution. Indeed, the parameters such as "license key", "start date", "end date," or "password" have a large #(p) value while most parameters appear just once. This observation also implies the existence of hub parameters, which appear in web services frequently and serve important roles on the inter-connections between web services.



(a) Real web services



(b) Synthetic web services from ICEBE05

Figure 1. #(p) distributions.

Similarly, we had plotted a graph using synthetic WSDL files that were used in ICEBE05. Among the 18 test beds used in ICEBE05, in the interest of space, we show a graph for 100-32 of Composition-2 (other cases show a similar pattern). All ICEBE05 test beds have $\beta \leq 0.5$ and four humps equally. Figure 1(b) shows four humps at around 1, 100, 200, and 800 with the highest value at third hump. This distribution shape differs considerably from real public web services of Figure 1(a). That is, the test environments of ICEBE05 do not necessarily capture characteristics of real web services. Since diverse scenarios need to be evaluated in testing, any benchmark for WSD and WSC problems must be able to generate both cases shown in Figure 1, in addition to many other cases (to be shown in Sections 2 and 3).

To address these needs and shortcomings, we have built *WSBen* [5], a web service discovery and composition benchmark. The WSBen has two main characteristics: (1) flexible web services matching framework, and (2) diverse web services network models. In the following two sections, we describe each characteristic in detail. Table 1 summarizes important notations used in this paper.

Table 1. Summary of notations

Notation	Meaning
P	A set of parameters
W	A set of web services
r	A query (by users or s/w agents), $r =< r_{in}, r_{out} >$ where, $r_{in} \subseteq P$ is initial input parameters and $r_{out} \subseteq P$ is goal parameters.
$\#(p)$	The parameter usage of $p \in P$. It is the total occurrence number of the parameter in the corresponding web service repository.
$\#In(A)$	The total number of occurrences of the parameter set A in all w_{in}, $w \in W$, $A \subseteq P$
$\#Out(A)$	The total number of occurrences of the parameter set A in all w_{out}, $w \in W$, $A \subseteq P$
$G^*(\{p\})$	The minimum sum of costs to achieve $p \in P$ starting from r_{in} ,where costs means the number of arcs taken to reach p from r_{in} in a parameter node network

2. Flexible Matching Framework

The first feature of WSBen is its flexible matching framework. To determine when an operation Op1 in a web service can invoke another operation Op2 in another web service, it should be considered if their corresponding input and output parameters are "matching." When all web services agree to use a single ontology set, the matching can be easily solved – if two parameters have the same spellings, then they are matched. However, since individual web services are often created in isolation, matching their vocabularies is problematic because of different formats, abbreviation, typo-graphical error, and/or homonyms. For instance, different terms may be used to refer to the same meaning (e.g., "cost" and "price"). Therefore, using literal equality as the only way to

[5] *WSBen* is pronounced like "wee-s-ben."

determine "matching" is too rigid to handle current web services environments. Possible matching schemes are [10]:

- *Exact matching*: two parameters are lexicographically the same.
- *Approximate matching*: two parameters match if their similarity determined from a distance function is beyond some threshold. For instance, parameters "password" and "passwd" are considered to be a match if Edit distance function is used with the threshold of 2, or parameters "license-Fee" and "FeeForLicense" can be a match if, after proper token segmentation, *cosine* similarity function between {licence, fee} and {Fee, For, License} is used with TF/IDF weight.
- *Semantic matching*: two parameters match if their "semantic" meaning is interchangeable. For instance, "cost" and "fee" are synonyms according to WordNet [18] or "4 WHL DR" and "AX56" may refer to the same meaning according to the provided RDF/OWL documents.

In addition, one can also consider another matching scheme from the operation perspective – "partial" and "full" matching. In general, given w^1 and w^2, if w^1 can be invoked at the current information state and $w^1_{out} \supseteq w^2_{in}$, then w^1 can "fully" match w^2. On the other hand, if w^1 cannot fully match w^2 but w^1_{out} can match a subset of w^2_{in}, that is $(w^1_{out} \cap w^2_{in} \neq \phi) \wedge \neg(w^2_{in} \subseteq w^1_{out})$, then w^1 can "partially" match w^2.

When only full matching is considered in the WSC problem, it can be seen as a single-source shortest path problem whose computational complexity is known as polynomial [1]. On the other hand, when both full and partial matching must be considered concurrently, the problem becomes a decision problem to determine the existence of a solution of k operators or less for propositional STRIPS planning, with restrictions on negation in pre- and post-conditions [6]. Its computational complexity is proved to be NP-complete [14]. Therefore, when the number of web services to search is not small, finding an optimal solution to the WSC problem (i.e., a chain of web services to invoke) is prohibitively expensive, leading to approximate algorithms instead.

3. Diverse Web Services Network Models

A set of web services form a network (or graph). Depending on the policy to determine nodes and edges of the network, there are varieties: web service level (i.e., coarse granularity), operation level, and parameter level (i.e., fine granularity) models.

The graph at the middle of Figure 2 has a bipartite graph structure and consists of three distinct kinds of vertices (i.e., parameter, operation, and web-service node) and directed arcs between bipartite nodes (i.e., operation nodes and parameter nodes). An edge incident from a parameter node to an operation node means that the parameter is one of the inputs of the corresponding operation. Reversely, an edge incident from an operation node to a parameter means that the parameter is one of the outputs of the corresponding operation. The graph has three web services, labeled WS1 to WS3. WS1 has two operations Op11 and Op12, and WS2 and WS3 have one operation, Op21 and Op31, respectively. It also has eleven parameters, labeled A to K. According to the node granularity, we can project the upper graph into three different web service networks.

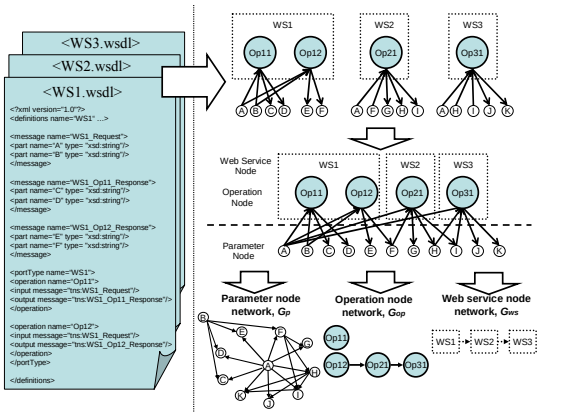
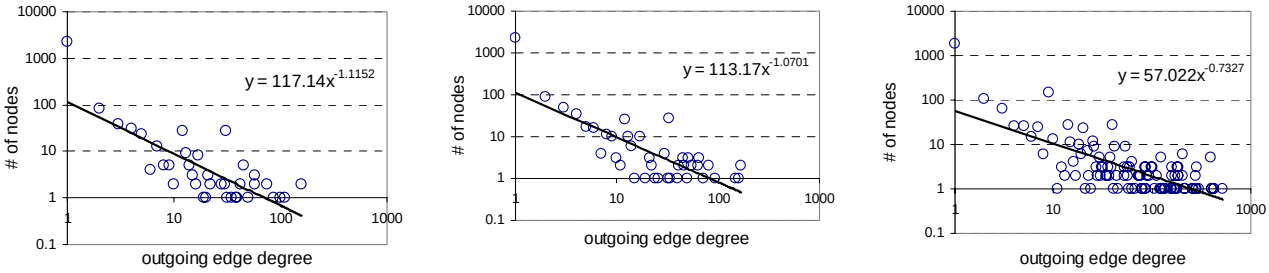


Figure 2. Web service networks

- *Parameter-Node Network*: A graph $G_p(V_p, E_p)$, where V_p is a set of all parameter nodes and E_p is a set of edges. An edge (p_i, p_j) is directly incident from input parameters $p_i \in V_p$ to output parameters $p_j \in V_p$, where there is an operation that has an input parameter matching p_i and an output parameter matching p_j. For example, A→Op11→C in the upper graph is projected into A→C in the parameter node network.
- *Operation-Node Network*: A graph $G_{op}(V_{op}, E_{op})$, where V_{op} is a set of all operation nodes, and E_{op} is a set of edges. An edge (op_i, op_j) is incident from operation $op_i \in V_{op}$ to operation $op_j \in V_{op}$, where op_i can fully or partially match op_j. For example, Op12 partially matches Op21 which, in turn, partially matches Op31 in the upper graph. Therefore, Op12→Op21→Op31 can be shown in the operation node network.



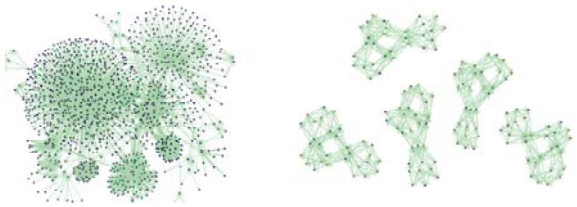
(a) exact matching (b) approximate matching using TF/IDF (c) semantic matching using WordNet

Figure 4. Outgoing edge distribution of operation node network

- *Web-service Node Network*: A graph $G_{ws}(V_{ws}, E_{ws})$, where V_{ws} is a set of all web-service nodes, and E_{ws} is a set of edges. An edge (ws_i, ws_j) is incident from web-service node $ws_i \in V_{ws}$, to $ws_j \in V_{ws}$, where there is at least one edge between any operation in ws_i and any operation in ws_j. For example, since WS1 possesses Op12 and WS2 possesses Op21 in the upper graph, WS1→WS2 appears in the web service node network.

1. **Parameter node network.** Figure 3(a) shows the parameter node network for the public web services where 1,612 nodes and 9,509 edges are discovered. Its network diameter is 8 and average out- and in-degree are 4.22 and 19.1, respectively. Note that the outgoing edge distribution of the network has a similar result to the $\#(p)$ distribution of Figure 1(a).

The study of parameter node network gives us an implication that a WSC problem can be relaxed and approximated by a search problem defined in a parameter space.



(a) public web services (b) workflow web services

Figure 3. Diverse parameter networks

When it comes to workflow web service domains (e.g., business, scientific, medical workflow), we can conjecture that their parameter node networks are different from public web services because the parameters used in workflows tend to be domain-specific or professional terms. Thus, $\#In(\{p\})$ and $\#Out(\{p\})$ are likely to be uniformly distributed. In other words, a web service is likely to be connected to a few number of neighboring web services in the succeeding stage of a workflow. As a result, a parameter node network like Figure 3(b) can be expected.

2. **Operation node network.** For the public web services, the outgoing edge distribution of the operation node network is shown in Figure 4(a), where the x-axis represents the number of outgoing edges of a node and the y-axis represents the number of the node with the same outgoing edges. In this context, the number of outgoing edges means how many times an operation gets involved to invoke other operations. Note that we allow an edge between operations whenever they match partially. When we consider a power function, the exponent β has 1.1152 which is enough to say that it follows a Zipf-like distribution. From this observation, we can imply that the public web services have hub operations that can play a critical role in finding intermediate operations in a WSC problem.

It is possible to draw different versions of operation node networks if one uses the diverse parameter matching schemes suggested in Section 2. Figure 4(b) is drawn using TF/IDF, which is an approximate parameter matching scheme, and Figure 4(c) is drawn using WordNet which enables semantic matching between parameters. Both graph follows Zipf-like distribution (i.e., β of Figures 4(b) and 4(c) are 1.07 and 0.7327, respectively). Note that in general, approximate and semantic matching relax the parameter matching condition so the number of edges in the operation node network increases.

3. **Web services node network.** The outgoing edge distribution of the web service node network for the public web services is shown in Figure 5, where both x and y axis have the same meaning as the operation node network. In this context, the number of outgoing edges means how many times a web service has a relationship with other web services through their operation matching. This network represents the relationship between web services by (1) unifying several edges between operations in two

different web services, and (2) wiping out all edges between internal operations in the same web service. By fitting it with a power function, we obtain 1.3073 as β, and again the distribution forms Zipf-like shape.

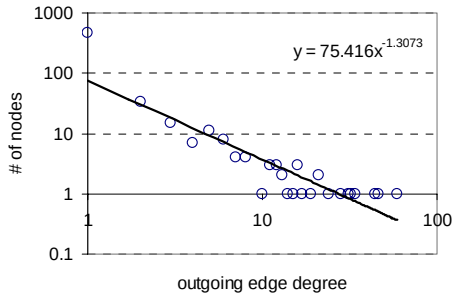


Figure 5. Outgoing edge distribution of the web service node network using exact matching

4. **WSBen: Web Services Benchmark**

The WSBen provides a set of functions to simplify the generation of test environments for WSD and WSC problems. Figure 6 shows the overview of WSBen. In detail, WSBen consists of the following functionalities:

- Input Parameters: users specify five parameters to control the generated synthetic WSDL files and their characteristics. The 5-tuple input parameters are named as *xTB*, which internally generate a cluster network. *xTB* are discussed in more detail below.
- Cluster network, $G_{cl}(V_{cl}, E_{cl})$**:** Based on the given *xTB* that users provide, clusters with atomic parameters are created as many as $|J|$ and then the incidence matrix M_j, to specify the directed edges between clusters is obtained. Based on J and M_j, a *Cluster Network*, $G_{cl}(V_{cl}, E_{cl})$ is defined. Each node, $cl_i \in V_{cl}$ corresponds to $j \in J$, and each edge, $\langle i,j \rangle \in E_{cl}$ is defined in M_j.
- Test-bed and sample queries**:** By randomly picking a web service template implemented in arcs of the cluster network, WSDL files are generated. The user can generate sample test queries, $r = < r_{in}, r_{out} >$ in this step. Note that r_{in} is a set of atomic parameters contained in the first cluster and r_{out} consists of the first five largest parameters with $G(\{p\})$, which is the number of arcs needed to reach p from r_{in} in the parameter network. Simply, parameters in r_{out} are farthest away from r_{in}.
- Test and evaluation: it is possible to export both the web service WSDL files and test queries into files in PDDL [17] and STRIPS format, enabling concurrent comparison with state-of-the-art AI planners. Additionally, it can export $\#(p)$ of all parameters into external files in comma separate file format (i.e., CSV), enabling users to analyze the repository data statistically.

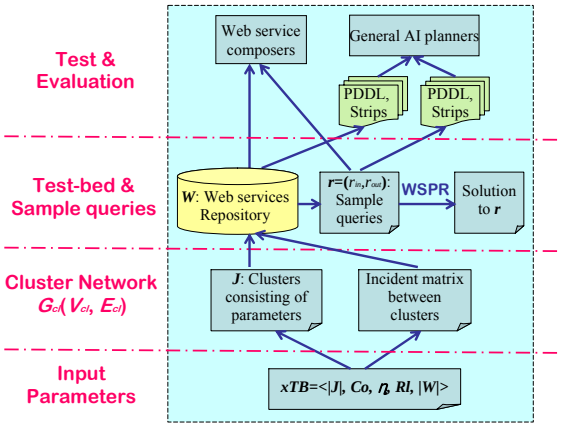


Figure 6. Overview of WSBen

Input Parameters. The 5-tuple input parameters consist of: $xTB = < J, Co, \eta, Rl, |W| >$. Provided that the first four tuple is grounded, one can build a cluster network like Figure 7 where clusters are nodes and web service templates are directed edges. At a higher level, web services can be assumed to be built on their application domains (e.g., travel, reservation, entertainment, and look-up services for diverse areas). One application domain can be projected into a parameter cluster, which contains a set of atomic parameters that occur together with similar frequency [4]. More precisely, two parameters $p_1, p_2 \in P$ in one cluster have the same co-occurrence probability such that $\Pr(p_2 \mid p_1) = \Pr(p_1 \mid p_2)$ where,

$$\Pr(p_2 \mid p_1) = \frac{[\#In(\{p_1, p_2\}) + \#Out(\{p_1, p_2\})]}{[\#In(\{p_1\}) + \#Out(\{p_1\})]}$$

In this context, web services are defined as transformations between two different clusters. That is, $\langle i,j \rangle \in E_{cl}$ in $G_{cl}(V_{cl}, E_{cl})$ becomes web service templates. In detail, each tuple of *xTB* is explained as follows:

(1) J is the set of clusters and $|J|$ denotes the total number of clusters of a test bed. $j \in J$ and $j = 1,2,\ldots,|J|$

(2) $Co(j)$ is the co-occurrence probability of parameters in a cluster $j \in J$. $Co(j)$ is grounded with one distribution selected from following alternatives:

- *skewed* : $skew(j; \alpha, \beta) = \beta e^{-\alpha j}$

- *bell*: $bell(j;\alpha,\beta) = \beta \exp^{(-\alpha \times ABSOLUTE(j-|j|/2))}$
- *uniform*: $uni(j;\alpha) = \alpha$

Where, α and β are constants given by users such that the value of the distribution must be between 0 and 1.

(3) η is the parameter condense rate. This value is used to specify the size of a cluster. That is, the total number of parameters in j cluster is $\eta / Co(j)$.

(4) $Rl(j)$ is the association distribution of a cluster $j \in J$ and is used to represent the outgoing edge degree of j. $Rl(j)$ shares the alternative distributions of $Co(j)$. For example, $Rl(j) = Uni(j,0.2)$ means that each cluster in a cluster network has outgoing edges incident to 20% of other clusters.

(5) $|W|$ denotes the total number of web services of a test bed.

For example, Figure 7 shows a cluster network specified by $<100, uni(0.6), 3, uni(0.1), |W|>$. Based on the network, we can generate $|W|$ size of web services by (1) randomly choosing $<i,j> \in E_{cl}$; and (2) generating input parameters from cluster i according to $Co(i)$, and output parameters from cluster j according to $Co(j)$. In the case that no parameter is generated, dummy parameters 'S' and 'T' are filled in the input and output parameters, respectively. Note that each parameter in one cluster can map into either an input parameter or an output parameter.

Both Co and Rl are paramount factors to determine the topology of a cluster network. By specifying both $Co(j)$ and $Rl(j)$ with *skewed* distributions, we can obtain a test bed which has a scale-free shaped parameter node network as shown Figure 8(a). We name this kind of test bed a *skew-skew* test bed. Similarly, a test bed characterized by a random-network shaped network can be obtained by specifying both $Co(j)$ and $Rl(j)$ with *uniform* distributions as shown in Figure 8(c). We name this type of test bed as *uni-uni* test bed. It is possible that the *skew-skew* and *uni-uni* test bed can approximate "public web services" and "workflow web services" respectively, based on the arguments in Section 3.

5. Illustrative Example

In this section, we demonstrate how to generate the benchmark using WSBen and how to run WSC products over the generated benchmark. In this paper, we use general AI planners in place of WSC products.

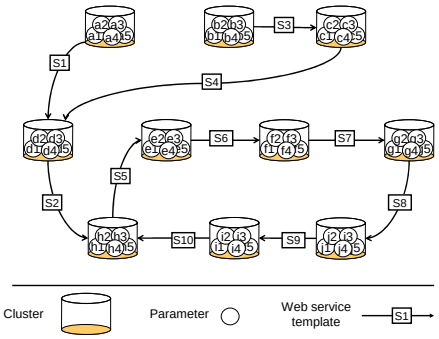


Figure 7. A cluster network example

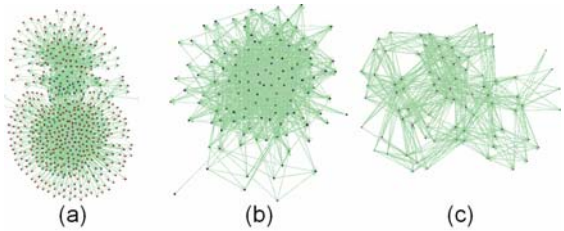
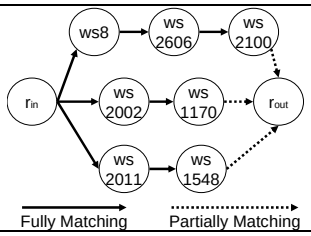


Figure 8. Synthetic parameter node networks

First, we defined two benchmark input parameters: (1) *aTB*: $<100, uni(0.2), 1, uni(0.2), |W|>$; and (2) *cTB*: $<100, skew(0.5,0.5), 1, skew(0.5,0.5), |W|>$. *aTB* is a kind of *uni-uni* test bed while *cTB* is classified as a *skew-skew* test bed.

Second, we created a cluster network for each *aTB* and *cTB*. Since the networks were too complex due to their large number of clusters and atomic parameters, we omit drawing the networks such as Figure 7 and their topologies such as Figure 8.

Table 2. A sample query and its solution

r_{in}	r_{out}	Solution
{par1, par2, par3, par4, par5}	{par483, par485, par134, par137, par 136}	

Third, for each *aTB* and *cTB*, 6 test beds are prepared by varying $|W|$ as 1,000, 3,000, 5,000, 7,000, 9,000, and 11,000, respectively. For each of the 12 test beds, we prepared one test query by using the test query generation option of WSBen. The solutions to the test queries are obtained by WSPR [11]. Table 2

shows an example query and its solution for $<100, uni(0.2), 1, uni(0.2), 3000>$.

Figure 9(a) shows #(*p*) distribution of *aTB* with |*W*|=11,000, which forms a bell curve. It coincides with the expected distribution since both *Co* and *Rl* have the same uniform distribution and therefore, theoretically all parameters have the same chance of occurring. Furthermore, since |*W*| is 11,000, the test-bed obeys the law of large numbers statistically, and the shape of the distribution converges to a bell or normal curve.

Meanwhile, Figure 9(b) shows #(*p*) distribution of *cTB*, which follows a Zipf-like distribution. It is also an expected result because both *Co* and *Rl* have the same skewed distribution and therefore, most parameters have a very low chance of occurring, but a very small number of parameters contained in hub clusters have high possibility of occurring. Note that in a cluster network, if a cluster has very large number of outgoing or incoming edges compared with other clusters, it is called a hub-cluster. When the power function is fitted, the exponent β is 1.5442, meaning that #(*p*) distribution is close to the power-law distribution.

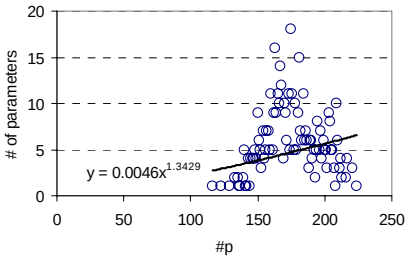
Last, in the experiments described below, we evaluated the performance of three prominent AI planners – Graphplan [2], Blackbox [7], and IPP. Blackbox and IPP are extended planning systems that originated from Graphplan. In particular, Blackbox is extended to be able to map a plan graph into a set of clauses for checking the satisfiability problem. Consequently, it can run even in large number of operators. For comparing the performance of three planners, we used the following two evaluation metrics:

- *#W*: the number of web services in a solution
- *Time*: the time (in millisecond or minute) taken to compose a solution

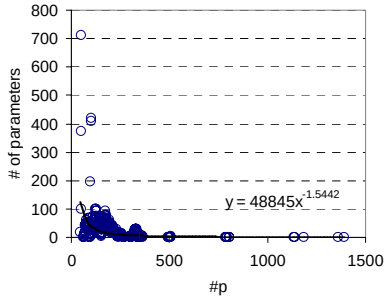
Note that all planners are an optimal parallel planner that minimizes the number of time steps but not necessarily the number of actions. All planners run with their default options, except that the maximum number of nodes for Blackbox and Graphplan was set to 32,768 and 10,000, respectively. Commonly, the time to read operator and fact files is not included in their *Time* measure. The experiments were performed on Linux with three Intel® Xeon™ CPU running at 2.4GHz with 8Gb RAM. Note that Blackbox and IPP (resp. Graphplan) accept the PDDL format (resp. STRIPS). Those files were generated by the auxiliary file generation option of WSBen.

The results of the first experiment dealing with *aTB* are shown in Table 3. As featured with the satisfiability solver, Blackbox generated all solutions, while IPP and Graphplan failed in many cases due to their internally confined array size problem.

As shown in Table 4, the second experiment using *cTB* shows that IPP and Graphplan failed to find solutions for all queries due to the complex network topology of *cTB*. Blackbox also failed in some cases. The complete solutions to *aTB* and *cTB* by WSPR are shown in Table 5.



(a) *aTB*



(b) *cTB*

Figure 9. #(*p*) distributions

Table 3. *aTB*: *Time(seconds)* and *#W*

Size (1,000)	Blackbox		IPP		Graphplan	
	Time	*#W*	*Time*	*#W*	*Time*	*#W*
1	0.26	16	1.5	16	0.04	17
3	0.32	13	6.75	13	0.03	9
5	0.44	8	15.16	8	0.04	4
7	0.55	4	27.46	4	-	-
9	0.211	7	2.86	7	-	-
11	3.215	17	-	-	-	-

Table 4. *cTB*: *Time(seconds)* and *#W* of Blackbox

Size (1,000)	1	3	5	7	9	11
Time	170	407	805	672	-	1,541
#W	17	16	11	12	-	10

Summary: From the experiments using WSBen, it is found that *cTB* is much harder to solve than *aTB* (e.g, in Table 4, the minimum *Time* exceeded two minutes). This implies that the queries in the *uni-uni* test bed can

be solved mainly using the full matching operation. However, the service composition in the skew-skew test bed requires both matching operations, resulting in planners having to spend more computational resource.

Table 5. #*W* of WSPR

Size(1,000)	1	3	5	7	9	11
aTB	19	**7**	6	**3**	**4**	**12**
cTB	**14**	**9**	**11**	**9**	**23**	12

6. Related Work

Constantinescu et al. [3] proposed a scalable syntactic test bed. Compared to [3], our WSBen is inspired by extensive studies on real web services and therefore, is designed to support various web service network topologies and distributions (unlike [3]). In addition, WSBen provides benchmarks that consist of various supports to simplify the testing process. We used XMark [12], an XML benchmark suite, as a reference model that can help identify the list of functions which an ideal benchmark should support.

As an independent research branch, *web service quality testing* has been established where quality and trustworthiness of web services are considered [8]. It is very unlikely that a business organization will dynamically select a partner from the Internet merely based upon the information found from some public registry without being highly confident. Our WSBen can be used to evaluate the quality and trustworthiness of web services in question.

There are two main approaches for WSC depending on the use of domain knowledge. First, the template-workflow based approach is to use software programs and domain experts to bind manually-generated workflows to the corresponding concrete web services. METEOR-S [15] is an example of this approach. Second, various AI planning techniques have been applied to the WSC problem, ranging from simple classical STRIPS-style planning to an extended estimated regression planning [16]. We believe that our WSBen is complementary for METEOR-S or AI Planning based tools for the WSC problem.

7. Conclusion

A novel web service benchmark, WSBen [6], is presented. Based on the snapshots of real-world web services and the assumption of workflow applications, WSBen is designed to test web service discovery and composition problems for diverse characteristics and sizes. Further research is needed to extend WSBen to support approximate and semantic matching among web services. WSBen is available for download at: **http://www2.ie.psu.edu/Kumara/Research/lisq/index_files/wsben/WSBen.htm**

[6] Current implementation of WSBen is limited as follows: (1) it supports only the exact matching without type compatibility check, and (2) each web service contains only one operation so that a web service can be viewed as equivalent to an operation. We are currently working to extend it to fully support features described in Sections 2 and 3.

8. References

[1] D. Bertsekas, "Dynamic Programming and Optimal Control", Vol 1, 2nd edn. Athena Scientific, 2000.

[2] A. Blum, and M. Furst, "Fast planning through planning graph analysis", Proceeding of IJCAI, 1995.

[3] I. Constantinescu, B. Faltings, and W. Binder, "Large scale testbed for type compatible service composition", Proceeding of ICAPS, 2004.

[4] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services", Proceeding of VLDB, 2004.

[5] J. Fan and S. Kambhampati, "A snapshot of public web services", SIGMOD Record, 34(1), 2005.

[6] R. E. Fikes, and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence, 5(2), 1971.

[7] H. Kautz, and B. Selman, "Unifying SAT-based and Graph-based planning", Proceeding of IJCAI, 1999.

[8] J. Zhang, and L.-J. Zhang, "Web services quality testing", Int'l J. of Web Services Research, 2(2), 2005

[9] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications", Phys. Rev. E 64, 026118, 2001.

[10] S.-C. Oh, D. Lee, and S. Kumara, "A comparative illustration of AI planning-based web services composition", ACM SIGecom Exchanges, 5(5), 2005.

[11] S.-C. Oh, D. Lee, and S. Kumara, "WSPR: An effective web service composition algorithm", Submitted to Int'l J. of Web Services Research, Special issue on web service discovery and composition, 2006.

[12] Xmark XML Benchmark, http://monetdb.cwi.nl/xml/

[13] J. P. Denning, "Network Laws", Communications of the ACM, 47(11), 2004.

[14] T. Bylander, "The computational complexity of propositional STRIPS planning", Artificial Intelligence, 69(1-2), 1994.

[15] K. Sivashanmugam, J. A. Miller, A. Sheth, and K. Verma, "Framework for semantic web process composition", Int'l J. of Electronic Commerce, 9(2), 2004.

[16] D. McDermott, "Estimated-regression planning for interactions with web services", Proceeding of AIPS, 2002.

[17] D. McDermott, "A heuristic estimator for means-ends analysis in planning". Proceeding of AIPS, 1996.

[18] Cognitive Science Laboratory in Princeton University, http://wordnet.princeton.edu/