WSBen:
A Web Services Discovery and Composition Benchmark Toolkit[1]

Seog-Chan Oh, General Motors R&D Center, USA

Dongwon Lee, The Pennsylvania State University, USA

ABSTRACT

In this article, a novel benchmark toolkit, WSBen, for testing web services discovery and composition algorithms is presented. The WSBen includes: (1) a collection of synthetically generated web services files in WSDL format with diverse data and model characteristics; (2) queries for testing discovery and composition algorithms; (3) auxiliary files to do statistical analysis on the WSDL test sets; (4) converted WSDL test sets that conventional AI planners can read; and (5) a graphical interface to control all these behaviors. Users can fine-tune the generated WSDL test files by varying underlying network models. To illustrate the application of the WSBen, in addition, we present case studies from three domains: (1) web service composition; (2) AI planning; and (3) the laws of networks in Physics community. It is our hope that WSBen will provide useful insights in evaluating the performance of web services discovery and composition algorithms. The WSBen toolkit is available at: http://pike.psu.edu/sw/wsben/.

Keywords: *AI Planning; Complex Network; Benchmark; Web Service Composition; Web Service Network*

INTRODUCTION

A *Web Service* is a set of related functionalities that can be loosely coupled with other services programmatically through the Web. Examples of web applications using Web services include weather forecasting, credit check, and travel agency programs. 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 Web-service discovery and composition problems have appeared[2]. In addition, the Web service research community has hosted open competition programs (e.g., ICEBE05[3], EEE06[4]) to solicit algorithms and software 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 software. The lack of such a testing environment with flexible features hinders the development of

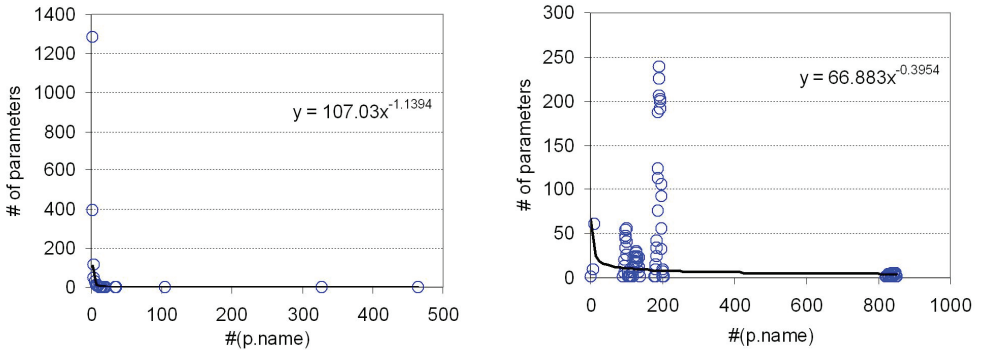
new composition algorithms and validation of the proposed ones. Therefore, the need for a benchmark arises naturally to evaluate and compare algorithms and software for the Web-service discovery and composition problems. As desiderata for such a benchmark, it must have (a large number of) web services in the standard-based WSDL files and test queries that can represent diverse scenarios and situations that emphasize different aspects of various Web-service application domains. Often, however, test environments used in research and evaluation have skewed test cases that do not necessarily capture real scenarios. Consider the following example.

Example 1 (Motivating) *Let us use the following notations: A Web service $w \in W$, specified in a WSDL file, can be viewed as a collection of operations, each of which in turn consists of input and output parameters. When an operation op has input parameters $op^i = \{p_1,...,p_n\}$ and output parameters $op^o = \{q_1,...,q_n\}$, we denote the operation by $op(op^i, op^o)$. Furthermore, each parameter is viewed as a pair of (name, type). We denote the name and type of a parameter p by p.name and p.type, respectively. For the motivating observation, we first downloaded 1,544 raw WSDL files that Fan and Kambhampati (2005) gathered from real-world Web services registries such as XMethods or BindingPoint. We refer to the data set as PUB06. For the purpose of preprocessing PUB06, first, we conducted WSDL validation according to WSDL standard, where 874 invalid WSDL files are removed and 670 files are left out. Second, we removed 101 duplicated WSDL files at operation level, yielding 569 valid WSDL files. Finally, we conducted type flattening and data cleaning processes subsequently. The type flattening process is to extract atomic types from user-defined complex types using type hierarchy of XML schema. This process helps find more compatible parameter faster. Details are found in (Kil, Oh, & Lee, 2006). The final step is the data cleansing to improve the quality of parameters. For instance, substantial number of output parameters (16%) was named "return", "result", or "response" which is too ambiguous for clients. However, often, their more precise underline meaning can be derived from contexts. For instance, if the output parameter named "result" belongs to the operation named "getAddress'", then the "result" is in fact "Address". In addition, often, naming follows apparent pattern such as getFooFromBar or searchFooByBar. Therefore, to replace names of parameters or operations by more meaningful ones, we removed spam tokens like "get" or "by" as much as we could.*

We measured how many distinct parameters each WSDL file contained. Suppose that given a parameter $p \in P$, we denote the number of occurrences of p.name as #(p.name). That is, #("pwd") indicates the number of occurrences of the parameter with name of "pwd". Figure 1 illustrates #(p.name) distributions of PUB06 and the ICEBE05 test set, where the x-axis is #(p.name) and the y-axis is the number of parameters with the same #(p.name) value. The distribution of PUB06 has no humps. We also plotted a power-function, over the #(p.name) distribution, and found that the exponent is 1.1394. Although 1.1394 does not suffice the requirement to be the power law (Denning, 2004), 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.name) 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. On the contrary, the distribution of ICEBE05 test set has four humps equally at around 1, 100, 200, and 800 with the highest value at third hump. This distribution shape differs considerably from PUB06, the real public Web services. This implies that the test environments of ICEBE05 do not necessarily capture characteristics of real Web services.

In conclusion, as demonstrated in the example, our claim is that *any Web-services discovery and composition solutions must be evaluated under diverse configurations of Web*

Figure 1. #(p.name) distributions. (left) PUB05. (right) ICEBE05.



services networks including two cases of Figure 1. However, to our best knowledge, there have been no publicly available benchmark tools. To address these needs and shortcomings, therefore, we developed the *WSBen* - a Web-Service discovery and composition Benchmark tool. The main contributions of WSBen is to provide diverse Web service test sets based on three network models such as "random", "small-world", and "scale-free" types. These three network models have been shown to model many real-world networks sufficiently (Albert & Barabasi, 2002). We also present three use cases in different communities to demonstrate the application of WSBen. In addition, we propose a flexible framework, by which we can study real Web service networks, and establish the design foundation of WSBen. As a whole, this article is based on two of our earlier works (Kil et al., 2006; Oh, Kil, Lee, & Kumara, 2006). Extended from the previous works, this article describes how WSBen is designed and works to generate test sets from the software architecture perspective, and furthermore introduces three use cases to highlight the practical benefits of WSBen. Table 1 summarizes important notations used in this article.

Table 1. Summary of notation

Notation	*Meaning*
w, W	Web service, set of Web services
p, P	Parameter, set of parameters
r, r^i, r^o	Request, initial and goal parameter sets of r
$G_p(V_p, E_p)$	Parameter node network
$G_{op}(V_{op}, E_{op})$	Operation node network
$G_{sw}(V_{sw}, E_{sw})$	Web service node network
$G^f_{op}(V^f_{op}, E^f_{op})$	Full-matching operation node network
$G_{c1}(V_{c1}, E_{c1})$	Parameter cluster network
$g_{r^i}(p)$	Minimum cost of achieving $p \in P$ from r^i in G_p
xTS	WSBen's 5-tuple input framework (e.g., *baTS*, *erTS* and *nwsTS* are instances)

This article is organized as follows. First, in the background section, we review concepts and techniques required for the WSBen development, especially focusing on the complex network theory. Second, in the related works section, we discuss related studies in the literature as well as surveying existing world-wide challenges with regard to Web services and Semantic Web. Third, in the overview of WSBen section, we present WSBen with its design concept, test set generation mechanism, key functions and characteristics. Fourth, in the use cases of WSBen section, we illustrate how WSBen can be exploited to obtain research benefits, especially by demonstrating three use cases. We expect three use cases enough to provide vigorous experiments and evaluation of our WSBen. Finally, conclusions are provided.

BACKGROUND

In this section, we review prerequisite techniques and concepts required to build WSBen. First, we revisit the definition and complexity of Web-service discovery and composition problems. Second, we introduce three complex network topologies based on which WSBen is designed to populate WSDL test files. Finally, we explain our conceptual methodology to project a bipartite Web-service network consisting of three distinct nodes (parameter, operation, and Web service) and heterogeneous arc types into three distinct Web-service networks, each of which consists of single node and uniform arc. The main benefit of projecting Web-service networks is that it can allow for straightforward analysis on referred network's characteristics. Throughout this article, we will use our conceptual Web-service network concept in order to analyze real public Web-service networks as well as WSDL test file sets generated by WSBen.

Web-Service Discovery and Composition

Suppose that a Web service w has one operation so that a Web service can be considered as an operation, and input and output parameter sets of w are denoted by w^i and w^o, respectively. When one has a request r that has initial input parameters r^i and desired output parameters r^o, one needs to find a Web service w that can fulfill such that (1) $r^i \supseteq w^i$ and (2) $r^o \subseteq w^o$. 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, on the other hand, one has to compose multiple Web services $\{w_1, w_2,...,w_n\}$, such that (1) for all $w_k \in \{w_1, w_2,...,w_n\}$, w_k^i can be applicable when w_k^i is required at a particular stage in composition, and (2) $(r^i \cup w_1^o \cup w_2^o \cup ... \cup w_n^o) \supseteq r^o$. This problem is often called as *Web-service composition* (WSC) problem. In addition, one can also consider different matching schemes 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 $w_1^o \supseteq w_2^i$, then w_1 can "fully" match w_2. On the other hand, if w_1 cannot fully match w_2, but w_1 can match a subset of w_2, 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 (Bertsekas, 2000). 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 (Nilsson, 2001). Its computational complexity is proved to be NP-complete (Bylander, 1994). 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.

Complex Network Models

There are many empirical systems to form complex networks such as the scale-free network and the small-world network, in which nodes signify the elements of the system and edges represent the interactions between them.

Definition 1 (Random network) *A network is defined as the random network on N nodes, if each pair of nodes is connected with probability p. As a result, edges are randomly placed among a fixed set of nodes. The random network can be constructed by means of the Erdos-Renyi's random-graph model (Erdos, Graham, & Nesetril, 1996).*

Definition 2 (Regular network) $Rg_{(N,\ k)}$ *is defined as the regular network on N nodes, if node i is adjacent to nodes* $[(i + j) \bmod N]$ *and* $[(i - j) \bmod N]$ *for* $1 \le j \le k$, *where k is the number of valid edge of each node. If* $k = N - 1$, $Rg_{(N,\ k)}$ *becomes the complete N-nodes graph, where every node is adjacent to all the other* $N - 1$ *nodes.*

We can define some metrics to quantify the characteristic properties of the complex networks as follows:

- L: The average shortest distance between reachable pairs of nodes, where the distance between nodes refers to the number of hops between the nodes. $L(p)$ is defined as L of the randomly rewired Watts-Strogatz graph (Watts & Strogatz, 1998) with probability p. L_{random} is identical to $L(1)$.
- C: The average clustering coefficient. Suppose that for a node i with v_i neighbor,

$$C_i = \frac{2E_i}{v_i(v_i - 1)},$$

where E_i is the number of edges between v_i neighbors of i. C is the average clustering coefficient C_i for a network. $C(p)$ is defined as C of the randomly rewired Watts-Strogatz graph with probability p. C_{random} is identical to $C(1)$.

Definition 3 (Small-world network) *Small-world networks are characterized by a highly clustered topology like regular lattices and the small network diameter, where the network diameter suggests the longest shortest distance between nodes. Specifically, small-world networks are* $C \succ C_{random}$ *and* $L \approx L_{random}$ *(Delgado, 2002).*

By using the Watts-Strogatz model (Watts, 1999; Watts & Strogatz, 1998), we can construct networks that have the small-world properties. The model depends on two parameters, connectivity (k) and randomness (p), given the desired size of the graph (N). The Watts-Strogatz model starts with a $Rg_{(N,k)}$ and then every edge is rewired at random with probability p; for every edge (i, j), we decide whether we change j node (the destination node of (i, j)) with probability p. The Watts-Strogatz model leads to different graphs according to the different p as follows:

- When $p = 0$, an $Rg_{(N,\ k)}$ is built.
- When $p = 1$, a completely random network is built.

Otherwise, with $0 < p < 1$, each edge (i, j) is reconnected with probability p to a new node k that is chosen at random (no self-links allowed). If the new edge (i, k) is added, the (i, j) is removed from the graph. The long-range connections (short-cuts) generated by this process decrease the distance between the nodes. For intermediate values of p, there is the "small-world" region, where the graph is highly clustered yet has a small average path length.

Definition 4 (Scale-free network) *Networks are called scale-free networks if the number of nodes that have v number of neighbor nodes is proportional to* $P_w(v) \propto v^{(-\gamma)}$, *where* γ *is typically greater than two with no humps.*

Barabasi and Albert provided several extended models (Albert, Jeong, & Barabasi, 1999; Delgado, 2002) to provide the scale-free proper-

ties. The extended model uses an algorithm to build graphs that depend on four parameters: m_0 (initial number of nodes), m(number of links added and/or rewired at every step of the algorithm), p (probability of adding links), q (probability of edge rewiring). The procedure starts with m_0 isolated nodes and performs one of the following three actions at every step:

- With the probability of p, $m(\leq m_0)$ new links are added. The two nodes are picked randomly. The starting point of the link is chosen uniformly, and the end point of the new link is chosen according to the following probability distribution:

$$\Pi_i = \frac{v_i + 1}{\sum_j (v_j + 1)} \qquad (1)$$

  where $\prod_i$ is the probability of selecting the i node, and v_i is the number of edges of node i.
  The process is repeated m times.
- With the probability of q, m edges are rewired. For this purpose, i node and its link l_{ij} are chosen at random. The link is deleted. Instead, another node z is selected according to the probabilities of Equation (1), and the new link l_{iz} is added.
- With the probability of $1 - p - q$, a new node with m links is added. These new links connect the new node to m other nodes chosen according to the probabilities of Equation (1).
- Once the desired number N nodes are obtained, the algorithm stops. The graphs generated by this algorithm are scale-free graphs, and the edges of the graphs are constructed such that the correlations among edges do not form. When $p = q$, the algorithm results in a graph, whose connectivity distribution can be approximated by

$$P(v) \propto (v+1)^{-\frac{2m(1-p)+1-2p}{m}+1} \qquad (2)$$

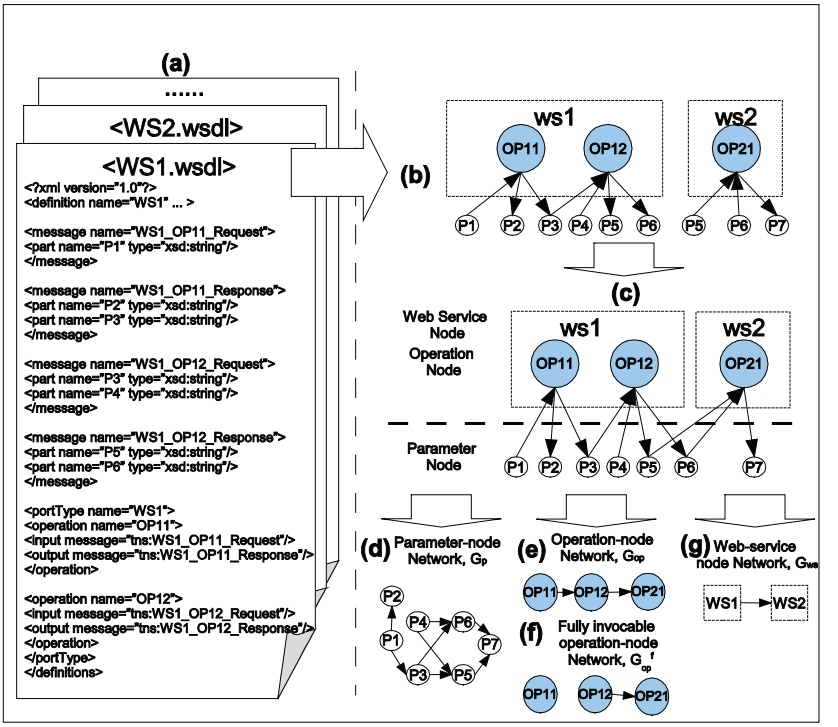
  where v is the number of edges.

Diverse Web Service 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 two Web services, labeled `WS1` and `WS2`. `WS1` has two operations `Op11` and `Op12`, and `WS2` has one operation, `Op21`, respectively. There are seven parameters, labeled `P1` to `P7`. According to the node granularity, we can project the upper graph into three different 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, `P1→Op11→P3` in the upper graph is projected into `P1→P3` in the parameter node network. Figure 3 shows the parameter node network for PUB06 and the ICEBE05 test set.
- 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 $op_i \in V_p$ to operation $op_j \in V_p$, here op_i can fully or partially match op_j. For example, `Op11` partially matches `Op12` which, in turn, fully matches `Op21` in the upper graph. Therefore, `Op11→Op12→Op21` can be shown

Figure 2. Web services networks: (a) WSDLs, (b) Conceptual Web service network, (c) Web service networks from diverse models, (d) Parameter node network, G_p, (e) Operation node network, G_{op}, (f) Fully invocable operation node network, G_{op}^f, and (g) Web service node network, G_{ws}



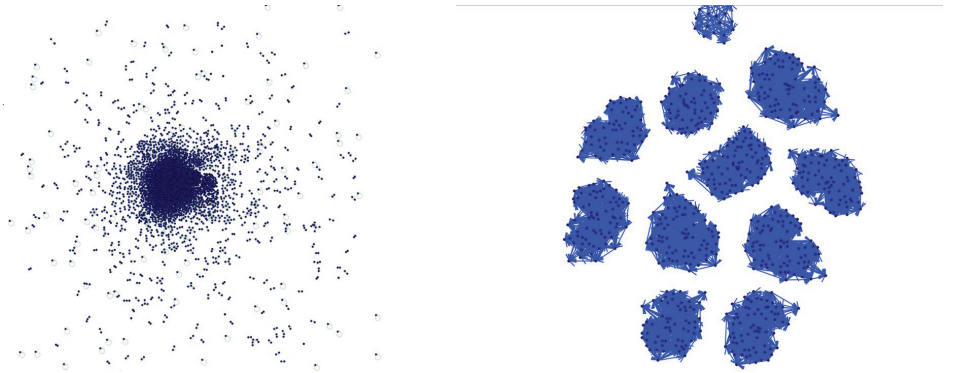
in the operation node network. In particular, the fully matching operation node network, G_{op}^f has only `Op12→Op21`.

- 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.

RELATED WORKS

Constantinescu, Faltings, and Binder (2004) proposed a scalable syntactic test bed where Web services are generated as transformation between sets of terms in two application domains. For doing that, they first defined parameter sets corresponding to application domains and then, connected those parameter sets randomly and constructed a service graph which structure (i.e., nodes and arcs) is similar to the parameter cluster network of WSBen. However, WSBen takes a significant different approach to construct its parameter cluster networks in that WSBen does not simply connect parameter sets at random but simulates topologies of real Web service networks. WSBen is inspired by extensive studies on real Web services, and

Figure 3. Diverse parameter networks. (left) PUB05. (right) ICEBE05



therefore is designed to support various Web service network topologies and distributions. As a result, WSBen can present more realistic testing situation for researchers who want to test their Web service discovery or composition algorithms than that of Constantinescu et al. (2004).

XMark (XMark, 2006) is an XML benchmark suite that can help identify the list of functions which an ideal benchmark should support. WSBen uses XMark as a reference model to identify necessary functions to simplify the testing process. One feature that is offered by XMark but not by WSBen is the provision of solutions to queries. In other words, XMark provides queries and their corresponding solutions but WSBen gives requests only because the optimal solution to a Web service composition problem may not be obtained in the reasonable time window due to the problem's inherently high complexity.

There are three unique challenges that have been established to investigate research issues with regard to Web services and Semantic Web. First is the Semantic Web Services Challenge[5]. This venue invites application submissions for demonstrating practical progress towards achieving the vision of the Semantic Web. According to the event, it has the overall goal to advance our understanding of how semantic technologies can be exploited to produce useful applications for the Web. Second is the Web Services Challenge[6]. This venue solicits approaches, methods, and algorithms in the domain of Web-service discovery and composition. This event evaluates participants' approaches based on their quantitative and qualitative performance results on discovery and composition problems. The Web Services Challenge is more driven by common problems, but the Semantic Web Challenge concentrates more on the environment. As such, the Semantic Web Challenge places more focus on semantics while the Web Services Challenges favors applied and short-term solutions (Brian, William, Michael, & Andreas, 2007). Third is the Service Oriented Architecture Contest[7] which asks participants to openly choose particular domain-specific problems and show their best approaches for them. There are unique characteristics for each venue so that they have undoubtedly contributed to advance the state-of-art technologies in Web services and Semantic Web. Among these challenges, WSBen can be exploited especially for the Web Services Challenge to provide various benchmark environments, discovery and composition problems by varying Web-service network topologies.

As for WSC, there are two main approaches, 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 (Sivashanmugam, Verma, Sheth, & Miller, 2003) 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 (McDermott, 2002). We believe that our WSBen is complementary for AI Planning based tools for the WSC problem. In fact, we demonstrate how WSBen can be used to compare the performance of AI planners for the WSC problems in the illustrative use-cases section. In this article, meanwhile, we do not propose how METEOR-S can make use of WSBen for a test case generation tool. It is because METEOR-S consists of three modules such as process designer, configuration module, and execution environment, where the execution environment requires executable Web services but WSBen can generate only WSDL files without real implementation.

Overview of WSBen

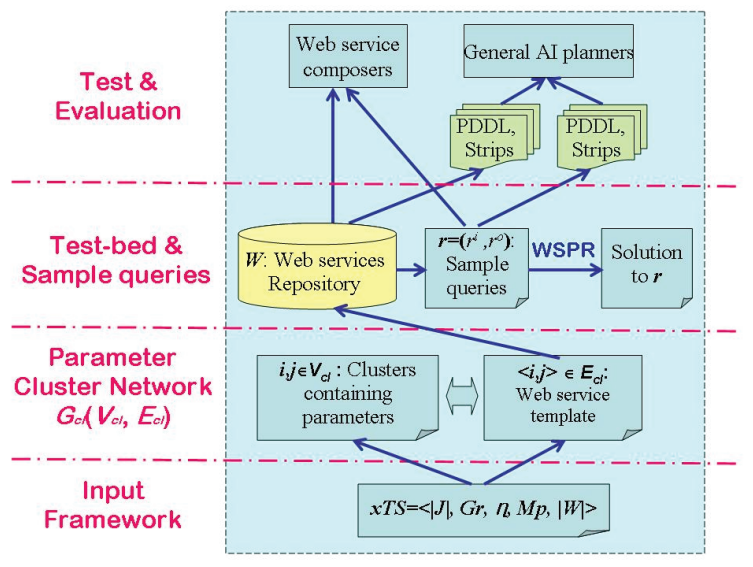
In this section, we present a novel benchmark tool titled WSBen, which provides a set of functions to simplify the generation of test environments for WSD and WSC problems.

Overview of WSBen

At a higher level, a Web service can be assumed as a transformation between two different application domains, and each can be represented by a cluster. This assumption was the basis in developing WSBen. From the perspective of graph theory, WSBen builds *Parameter Cluster Network*, which consists of clusters and directed edges connecting two different clusters. These directed edges become Web service templates from which WSBen generates Web services as users specify. Formally, the parameter cluster network is defined as follows:

Definition 5 (Parameter Cluster Network) *A directed graph $G_{cl}(V_{cl}, E_{cl})$, where V_{cl} is a set of clusters and E_{cl} is a set of directed edges that are incident from input clusters $i \in V_{cl}$ to output clusters $j \in V_{cl}$. Here, cluster i and j contain a set of non-overlapping parameters denoted by Pa_i and Pa_j, respectively, where $Pa_i \cap Pa_j = \phi$. Each directed edge is also called a Web*

Figure 4. Overview of WSBen



service template, from which WSDL files are generated.

Figure 4 shows the overview of WSBen. In detail, WSBen consists of the following functionalities:

- Input framework: Users can specify and control the generated synthetic WSDL files and their characteristics. For this purpose, WSBen provides an input framework $xTS = <|J|, G_r, \eta, G_p, |W|>$. *xTS* applies existing complex and random network models to specify G_r. Each element of *xTS* will be discussed in more detail below.
- Parameter cluster network, $G_{cl}(V_{cl}, E_{cl})$: If *xTS* is given by users, based on the first four elements, WSBen generates G_{cl}. Each cluster of G_{cl} is filled with some number of atomic parameters. In this network, Web services are defined as transformations between two different clusters. That is, $<i, j> \in E_{cl}$ becomes Web service templates. The role of Web service templates in the test set generation will be illustrated.
- Test set and sample requests: By randomly selecting the Web service templates (arcs of the parameter cluster network), WSDL files are generated. Once a test set is generated, users can generate sample test requests $r = <r^i, r^o>$. The generation process of test sets and test requests will be illustrated.
- Test and evaluation: WSBen can export both the Web service WSDL files and test requests into files in PDDL (McDermott, 1996) and STRIPS format, enabling concurrent comparison with state-of-the-art AI planners.

WSBen input framework: ***xTS***

WSBen input framework, *xTS* consists of five tuples, $<|J|, G_r, \eta, G_p, |W|>$. In detail:

1. $|J|$ is the total number of parameter clusters.
2. G_r denotes a graph model to specify the underlying topology of a parameter cluster network. G_r can be on of the following three models discussed in the Background section:
   - *Erdos-Renyi*($|J|$, p): This model has such a simple generation approach that it chooses each of the possible

     $$\frac{|J|(|J|-1)}{2}$$

     edges in the graph with $|J|$ nodes with probability p. The resulting graph becomes the same as the binomial graph.
   - *Newman-Watts-Strogatz*($|J|, k, p$): The initialization is a regular ring graph with k neighbors. During the generation process, new edges (shortcuts) are added randomly with probability p for each edge. Note that no edges are removed, differing from Watts-Strogatz model.
   - *Barabasi-Albert*($|J|$, m): This graph model is generated by adding new nodes with m edges that are preferentially attached to existing nodes with a high degree. The initialization is a graph with m nodes and no edges. Note that the current implementation of WSBen is limited because it can only generate the simplified version of the extended Barabai-Albert model, by setting $p = q = 0$ and $m_0 = m$, resulting in graphs with $\gamma = 2.0 \pm 0.1$, where γ is the exponent of a power function $P_w(v)$ defined over connectivity v range in the form of $P_w(v) \propto v^{-\gamma}$.
3. η denotes the parameter condense rate. With η, users can control the density of partial-matching cases in produced Web services.
4. M_p denotes the minimum number of parameters a cluster can contain. In other words, clusters may have a different number of parameters but all clusters must have at least M_p number of parameters.
5. $|W|$ denotes the total number of Web services of a test set.

With $|J|$ and G_r, the first two input elements of xTS, we can build G_{cl} with each empty cluster. Thus, we need a procedure to fill each empty cluster with parameters. For this purpose, WSBen uses the following procedure:

1. A parameter cluster network G_{cl} with empty clusters is built by specifying $|J|$ and G_r.
2. Co-occurrence probability of each cluster is measured by the following probability:

$$\Delta_j = \frac{k_j}{\max_{j \in V_{cl}} k_j} \eta \qquad (3)$$

   where Δ_j is the co-occurrence probability of cluster j, and k_j is the edge degree of cluster j.
   η is the parameter condense rate which is given by users.
3. $|Pa_j|$ is measured based on the following equation.

$$|Pa_j| = \frac{M_p}{\Delta_j} \qquad (4)$$

   where Pa_j is the set of parameters contained in cluster j.
4. For each j cluster, atomic parameters are generated up to $|Pa_j|$, with duplicated parameters forbidden (i.e., $\forall i, j \in V_{cl}$, $Pa_i \cap Pa_j = \phi$).

Once a complete parameter cluster network, $G_{cl}(V_{cl}, E_{cl})$ is built, WSBen repeats the following procedure until $|W|$ number of Web services are generated:

1. A Web service template $<i, j>$ is chosen at random from E_{cl}.
2. WSBen generates a WSDL file, in which each input parameter is selected from i cluster with probability Δ_i, and each output parameter is selected from j cluster with probability Δ_j.

Figure 5 illustrates how WSBen builds G_{cl} and generates WSDL files based on the G_{cl}. Suppose that xTS = <8, *Barabasi – Albert* (8, 2), 0.8, 1.5, 100> is given. Then, the generation steps are as follows:

1. WSBen generates a graph of *Barabasi-Albert*(8,2). The direction of each edge is determined at random.
2. Δ_j and $|Pa_j|$ are specified. For example, Cluster 5 has nine parameters as shown in Figure 5. That is, $|Pa_5| = 9$, as

$$\Delta_5 = \frac{k_j}{\max_{j \in V_{cl}} k_j} \eta = \frac{1}{5} \times 0.8 = 0.16,$$

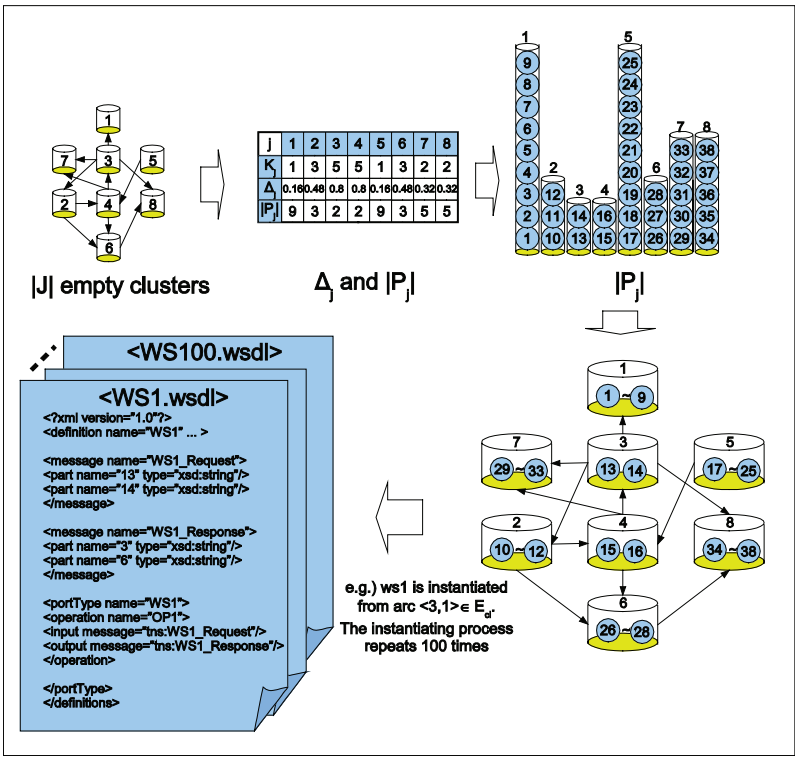
   resulting in

$$|Pa_5| = \frac{M_p}{\Delta_5} \approx 9.$$

3. Pa_j is specified. For example Pa_5={17,18, 19,20,21,22,23,24,25} as shown in Figure 5 because $|Pa_5| = 9$ and for $\forall i, j \in V_{cl}$, $Pa_i \cap Pa_j = \phi$. Note that the parameter names are automatically generated, and thus do not contain any semantics.
4. Finally, G_{cl} is built and WSBen generates $|W|$ Web services. For example, in Figure 5, `WS1` is instanced from a Web service template <3, 1> $\in E_{cl}$. Note that $\Delta_1 = 0.16$ and $\Delta_3 = 0.8$. $\Delta_1 = 0.16$ suggests that the occurrence probability of each parameter in Cluster 1 has 0.16. Due to the low probability, only "1" and "9" are selected from Cluster 1. Similarly, $\Delta_3 = 0.8$ means that the occurrence probability of each parameter in Cluster 3 has 0.8. Due to the high probability, all parameters in Cluster 3 that are "13" and "14" are selected. In the case where no parameter is generated, dummy parameters "S" and "T" are filled in the input and output parameters, respectively.

The state, $s \in S$ is a collection of parameters in $|P|$. Therefore, r^i and r^o are states. The test request r is constructed such that r^o is farthest away from r^i in a parameter space in terms of $g_{r^i}(p)$ - the cost of achieving $p \in P$ from a state r^i. To obtain $g_{r^i}(p)$, we propose following *Forward Searching* algorithm.

Figure 5. Test set generation with <8, Barabasi – Albert (8, 2), 0.8, 1.5, 100>



Algorithm 1. Forward searching algorithm of WSBen

Test Request Generation

Input : r^i
Output: $g_{r^i}(p)$ for all p reachable from r^i
1: $s = r^i$; $C = \phi$; $d = 1$;
2: **while** ($\delta \neq \phi$) **do**
3: $\delta = \{w \mid w \in \Omega(s), w \notin C\}$;
4: **for** p in $w^o (w \in \delta)$ **do**
5: **if** $g_{r^i}(p) = \infty$ **then**
6: $g_{r^i}(p) = d$; $s = s \cup \{p\}$;
7: $C = C \cup \delta$; d++;

Forward Searching: $g_{r^i}(p)$ can be characterized by the solution of a recursive equation as follows:

$$g_{r^i}(p) = \min_{w \in Ow(p)} [c(w) + \max_{p' \in w^i} g_{r^i}(p')] \quad (5)$$

where $c(w)$ is an invocation cost of a Web service, $w \in W$ and is assumed to be 1. $Ow(p)$ is a set of Web services: $Ow(p) = \{w \in W \mid p \in w^o\}$. At first, $g_{r^i}(p)$ is initialized to 0 if $p \in r^i$, and to ∞ otherwise. Then, the current information state s is set to r^i (Line 1 in Algorithm 1). We denote $\Omega(s)$ by a set of Web services $w \in W$ such that $w^i \subseteq s$. That is, w can be invoked or applicable in the state s.

Every time for $\forall w \in \Omega(s)$, each parameter $p \in w^o$ is added to s, and $g_{r^i}(p)$ is updated until $\Omega(s)$ stops to increase, meaning that this process ends with finding $g_{r^i}(p)$ for all parameters reachable from r^i (Lines 2-6 in Algorithm 1).

We can use Equation (5) to drive the lower bound of the optimal cost of WSC solutions. Note that the invocation cost of a Web service is assumed to be 1. Thus, the optimal cost of a WSC problem coincides with the minimum number of Web services required to solve the WSC problem. For a set of parameters A, we can regard the following cost function:

$$g_{r^i}^{\max}(A) = \max_{p \in A} g_{r^i}(p) \qquad (6)$$

The cost of achieving a set of parameters cannot be lower than the cost of achieving each of the parameters in the set. Thus, $g_{r^i}^{\max}(A)$ is the lower bound of the optimal cost of achieving r^o from r^i.

Based on the forward searching algorithm, WSBen create a test request r, as follows:

1. WSBen selects a Cluster $j \in G_{cl}$ at random.
2. WSBen copies all parameters in the Cluster j (i.e., Pa_j) into r^i, and then r^o is constructed so that it consists of the first five largest parameters in terms of $g_{r^i}(p)$. Consequently, parameters in r^o are farthest away from parameters in r^i in a parameter space.

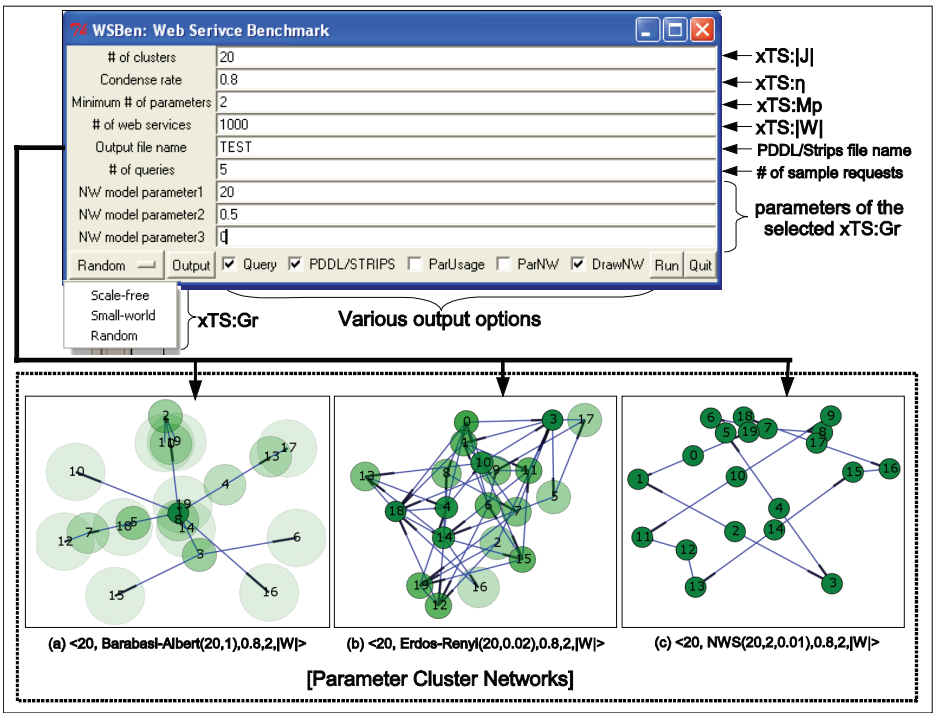
As a default, WSBen repeats the above procedure five times, generating five request sets for each test set.

Implementation

As shown in Figure 6, WSBen provides user interfaces to specify *xTS* and several parameters, which are required to form a parameter cluster network and generate WSDL files. WSBen is implemented in Python, and run on Python 2.3 or later. It runs on Unix and Windows. For the creation, manipulation, and functions of complex networks, we used a Python package called NetworkX[8]. 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. Therefore, w^i and w^o indicate the input and output parameter set of a Web service, w.

Figure 6 also shows three sample parameter cluster networks, where each circular node represents a cluster and edges with heads denote the Web service template, from which Web services

Figure 6. WSBen user interface



are instanced. The size of node is proportional to the number of parameters in the node, while the transparency level of a node's color is inversely proportional to the degree of the node. For example, in the left cluster network, Cluster 18 can be considered a hub cluster in that it has the high degree. Therefore, it is presented by a small circle with denser color.

Following the mechanism of WSBen explained so far, we can build three illustrative test set frameworks by specifying *xTS* as follows:

1. *baTS* = <100, *Barabasi – Albert* (100, 6), 0.8, 5, |*W*|>
2. *nwsTS* = <100, *Newman – Watts – Strogatz*(100, 6, 0.1), 0.8, 5, |*W*|>
3. *erTS* = <100, *Erdos – Renyi*(100, 0.006), 0.8, 5, |*W*|>

Figures 7 and 8 show that there are distinctive differences between *baTS*, *nwsTS*, and *erTS* in terms of G_p and outgoing edge degree distribution.

ILLUSTRATIVE USE CASES OF WSBEN

In this section, we present three use cases to demonstrate the application of WSBen: (1) evaluating Web-service composition algorithms; (2) comparing performance of AI planners; and (3) estimating the size of giant component. These use cases are prepared to provide vigorous experiments and evaluation for assessing the usage of WSBen. For each use case, we will provide three Web-service test sets by varying *xTS* with three parameter cluster networks such as "random", "small-world", and "scale-free" types. Note that these three network models have the expression power enough to model many real-world networks sufficiently (Albert & Barabasi, 2002). This implies that our generated test cases can be appropriate for representing diverse real-world Web-service networks. Furthermore, these three Web-service test sets are significantly distinctive from each other in terms of their Web-service network topologies and degree distributions as we have shown in the previous section. This indicates that we have sufficient reason to analyze how different network topologies can affect the performance of Web-service applications or environments.

Evaluating Web-Service Composition Algorithms

Recently, many WSC researches have been reported in the Web service community. As such, the EEE06 Web-service composition contest holds as many participants as 11. Among the 11 WSC algorithms, we choose a WSC algorithm named WSPR (Oh, Lee, & Kumara, 2007), which was proved effective and efficient in the contest, in order to demonstrate the application of WSBen.

Figure 7. G_p of baTS, erTS, and nwsTS when |W| = 1,000

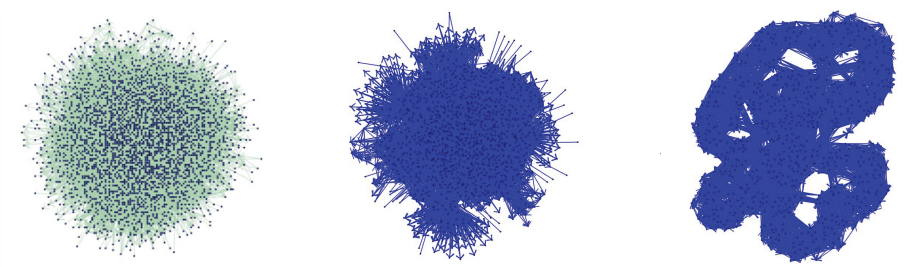


Figure 8. Outgoing edge degree of baTS, erTS, and nwsTS when |W| = 1,000

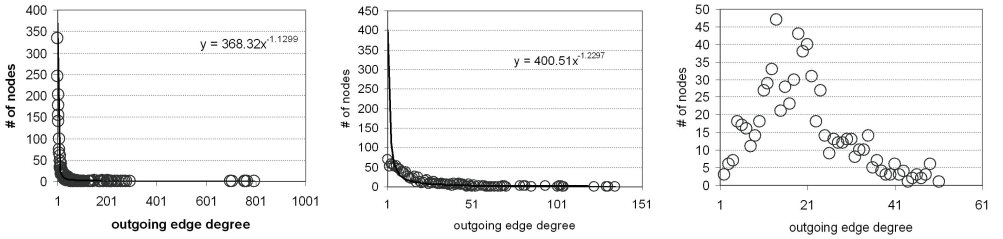
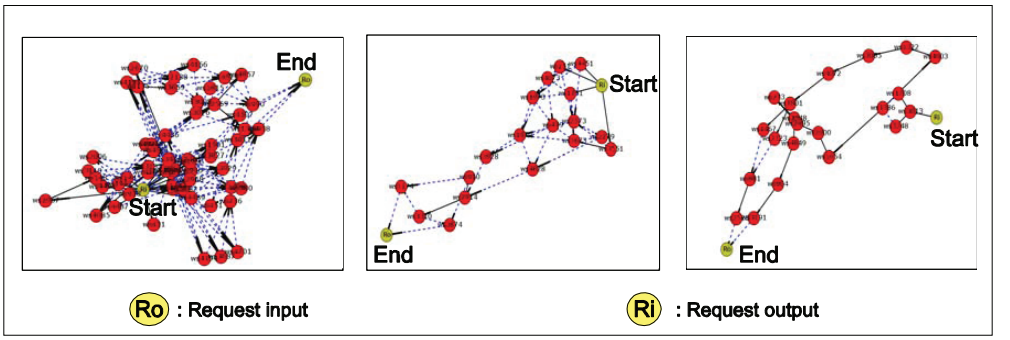


Figure 9. Composed services using WSPR for three test sets. (left) baTS with |W| = 5,000. (center) erTS with |W| = 5,000. (right) nwsTS with |W| = 5,000



In this case, we use three test sets generated by WSBen: (1) *baTS* with |*W*| = 5,000; (2) *erTS* with |*W*| = 5,000; and (3) *nwsTS* with |*W*| = 5,000. The resultant composed services generated by WSPR are shown in Figure 9, where WSPR addressed a request for each of the three test sets. Note that WSBen can automatically create sample requests for a given test set. In the graph, each composed solution has nodes such as "*Ri*" and "*Ro*", which represent the initial condition and goal state, respectively. Other nodes represent Web services. The directed arcs indicate the invocation flow, where a solid edge means full-matching invocation and a dotted edge represents partial-matching invocation. From the experiments based on diverse test sets such as *baTS*, *erTS*, and *nwsTS*, we can understand how different network models of G_{cl} influences the performance of WSC algorithms. In general, given the same number of clusters, the *Barabasi-Albert* model generates G_{cl} with a greater number of parameters, and a larger variance of the number of parameters between clusters than the *Newman-Watts-Strogatz* and *Erdos-Renyi* models do. Due to the greater number of parameters and larger variance, *baTS* needs more partial-matching Web services to fulfill the given requests than others. The increasing need for partial-matching Web services leads to increasing number of Web services in the composed service. This is the reason that the *baTS* case has more Web services to create a resultant composed service as shown in Figure 9 (left).

Comparing Performance of AI Planners

We demonstrate how WSBen can be used to compare the performance of AI planners. For this purpose, we choose three prominent AI planners – Graphplan (Blum & Furst, 1997), Blackbox (Kautz & Selman,1996), and IPP[9].

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 use two evaluation metrics as follows:

1. τ(Time): It measures how long an algorithm takes to find a right solution, in seconds. This is a measure of computational efficiency.
2. *#W*: The number of Web services in a right solution. This is a measure of effectiveness.

All AI 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 τ. Blackbox and IPP accept only the PDDL format, while Graphplan accepts only the STRIPS format for their operator and fact files. Note that an operator file corresponds to a test set, and a fact file corresponds to a test request file. Also note that WSBen provides a function to convert test sets and requests into PDDL and STRIPS files automatically. The experiments were performed on Linux with three Intel® Xeon™ CPU, running at 2.4GHz with 8GB RAM.

Tables 2, 3, and 4 shows the results of the five test requests for each of *baTS*, *erTS*, and *nwsTS* with $|W|$ = 3,000. Graphplan ran out of memory in many cases. IPP also failed to solve the some requests. As a whole, Blackbox showed better performance than others, meaning that it can solve more requests than others. It is because Blackbox uses the local-search SAT solver, Walksat, for the satisfiability problem, so that Blackbox can run relatively well even with a large number of operators.

We can estimate the size of giant component in a service network using random graph theory. Often it is believed to be important to have a large and dense giant component in a service network. Otherwise, the isolated services will never have a chance to provide any services to

Table 2. Results over baTS with $|W|$ = 3,000

Requests	BlackBox		GraphPlan		IPP	
	#W	τ	*#W*	τ	*#W*	τ
r_1	61	478.69	-	-	-	-
r_2	-	-	-	-	-	-
r_3	5	5	5	0.09	5	26.22
r_4	9	27.78	9	0.11	9	28.56
r_5	4	1.4	4	0.04	4	23.97

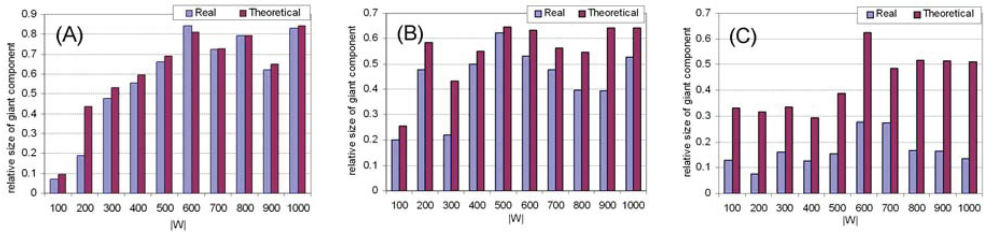
Table 3. Results over erTS with $|W|$ = 3,000

Requests	BlackBox		GraphPlan		IPP	
	#W	τ	*#W*	τ	*#W*	τ
r_1	75	38.09	-	-	-	-
r_2	50	16.02	-	-	-	-
r_3	22	18.68	-	-	22	24.78
r_4	23	4.38	-	-	23	21.06
r_5	38	4.01	-	-	38	21

Table 4. Results over nwsTS with |W| = 3,000

Requests	BlackBox		GraphPlan		IPP	
	#W	τ	*#W*	τ	*#W*	τ
r_1	48	571.63	-	-	48	29.52
r_2	35	114.67	-	-	35	28.57
r_3	24	192.99	-	-	24	30.19
r_4	26	11.88	-	-	26	28.39
r_5	31	111.21	-	-	-	-

Figure 10. Comparison of real and theoretical size of giant components: (A) random, (B) scale-free, and (C) NWS models.



Estimating the size of giant component

clients. Newman, Strogatz, and Watts (2001) suggested the theoretical framework in order to estimate the giant component size in networks by using random graph theory. In order to see if their theoretical framework works, we generated the g_{op}^{f} with different network size for each of following cases:

1. *Random model*: <50, *Erdos – Renyi*(100, 0.6), 0.8, 5, |*W*|>
2. *Scale-free model*: <50, *Barabasi – Albert*(100, 6), 0.8, 5, |*W*|>
3. *NWS model*: <50, *Newman – Watts – Strogatz*(100, 6, 0.1), 0.8, 5, |*W*|>

For each of these networks, we measured the real size of giant components. Then, we calculated the theoretical size of giant components according to the estimation model of Newman et al. (2001). The comparisons between real and theoretical sizes are summarized in Figure 10. For g_{op}^{f} based on the random parameter cluster network in Figure 10(A), the theoretical value of the giant component size is very close to the measured one for each synthetic network. This implies that even a simple random model may be very helpful to estimate the inter-operable portion of such networks with random topology without even analyzing the available network beyond its degree distribution. However, Figure 10(B) shows that the estimation model is not a good model for Scale-free type. There is a considerable gap between theory and real value for many of the synthetic networks in this type. The deviation between theory and actual networks becomes more dramatic for the *NWS* (small world phenomenon and highly clustered property) type shown in Figure 10(C). The results show that the random network model might be good generative model for such Web services networks if these networks are entirely random, which is also in accordance with the basic assumption by Newman et al. (2001).

CONCLUSION

A novel Web-service benchmark toolkit, WSBen, is presented with three use cases in different application domains. The WSBen development is inspired by the study on real-world Web services, and is designed to provide diverse scenarios and configurations which users can fine-tune easily. As a result, using WSBen, users can conduct extensive experimental validation on their Web-service discovery and composition algorithms. It is our hope that WSBen will provide useful insights to the design and development of Web-services discovery and composition solutions and software. The latest version of WSBen is available at: http://pike.psu.edu/sw/wsben/. Further research is needed to extend WSBen to support approximate and semantic matching among Web services. Also, we plan to discover additional applications where the WSBen can be used.

ACKNOWLEGMENT

Authors would like to thank Hyunyoung Kil and Ergin Elmacioglu for their help on the design and implementation of WSBen, and Professor Soundar R. T. Kumara at Penn State for his helpful comments on the earlier draft of this article.

REFERENCES

Albert, R. & Barabasi, A. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, *74*(1), pp. 47-95.

Albert, R. & Barabasi, A. (2000). Topology of evolving networks. *Phys. Rev.Lett.*, *85*, pp. 5234-5237.

Albert, R., Jeong, H. & Barabasi, A. (1999). The diameter of the world wide web. *Nature*, *401*, pp. 130-131.

Bertsekas, D. (2000). *Dynamic programming and optimal control*, volume 1, 2nd ed. Athena Scientific, Boston.

Blum, A. & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*, pp. 281-300.

Brian, B., William, C., Michael, J., & Andreas, W. (2007). WSC-07: Evolving the web services challenge. *In Proc. of 9th IEEE International Conference on E-Commerce Technology and 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC-EEE 2007)*, pp. 505-508, Tokyo, Japan.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*(1-2), pp. 165-204.

Constantinescu, I., Faltings, B. & Binder, W. (2004). Large scale testbed for type compatible service composition. In *Proc. of 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 23-28, Whistler, British Columbia, Canada.

Delgado, J. (2002). Emergence of social conventions in complex networks. *Artificial Intelligence*, *141*, pp. 171-185.

Denning, J. P. (2004). Network laws. Communications of the ACM, 47(11), pp. 15-20.

Erdos, P, Graham, R., & Nesetril, J. (1996). *The mathematics of Paul Erdos*. Springer-Verlag, Berlin, Germany.

Fan, J., & Kambhampati, S. (2005). A snapshot of public Web services. *SIGMOD Record*, *34*(1), pp. 24-32.

Kautz, H. & Selman, B. (1996). Unifying SAT-based and graph-based planning. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 318-325, Stockholm, Sweden.

Kil, H., Oh, S., & Lee, D. (2006). On the Topological Landscape of Semantic Web services Matchmaking. In *Proc. of 1st International Workshop on Semantic Matchmaking and Resource Retreival (SMR2006)*, pp. 19-34, Seoul, Korea.

McDermott, D. (1996). A heuristic estimator for means-ends analysis in planning. In *Proc. of the third International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 142-149, Edinburgh, Scotland.

McDermott, D. (2002). Estimated-regression planning for interactions with Web services. In *Proc. of the sixth International Conference on Artificial Intel-*

ligence Planning and Scheduling Systems (AIPS), pp. 67-73, Toulouse, France.

Newman, M. E. J., Strogatz, S. H., & Watts, D. J. (2001). Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, *64*(026118).

Nilsson, J. (2001). *Artificial Intelligence: a new synthesis*. Morgan Kaufmann, San Francisco, CA.

Oh, S., Kil, H., Lee, D., & Kumara, S. (2006). WSBen: a Web-services discovery and composition benchmark. In *Proc. of the fourth International IEEE Conference on Web service (ICWS)*, pp. 239-246, Chicago.

Oh, S., Lee, D., & Kumara, S. (2007). WSPR: a heuristic algorithm for Web-service composition. *Int'l J. of Web services Research (IJWSR)*, *4*(1), pp. 1-22.

Sivashanmugam, K., Verma, K., Sheth, A., & Miller, J. (2003). Adding semantics to Web services standards. In *Proc. of the first IEEE International Conference on Web services (ICWS)*, pp. 395-401, Las Vegas, NV.

Watts, D. J. (1999). *The dynamics of networks between order and randomness*. Princeton Univ. Press, Princeton, NJ.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of small-world networks. *Nature*, *393*(4), pp. 440-442.

XMark. An XML Benchmark Project. Retrieve on September 17, 2006 from, http://monetdb.cwi.nl/xml/.

ENDNOTES

1 This article is a substantial extension from the short version that appeared in the proceedings of the 4th International IEEE Conference on Web Services (ICWS), held in Chicago, USA, 2006. The work of Seog-Chan Oh was done while the author was with the Pennsylvania State University.

2 As of August 2007, according to the estimation of Google Scholar, there are about 2,360 scholarly articles mentioning "Web Services Composition".

3 http://www.comp.hkbu.edu.hk/~ctr/wschallenge/

4 http://insel.flp.cs.tu-berlin.de/wsc06/

5 The Semantic Web Services Challenge (2007): http://sws-challenge.org/wiki/index.php/Main_Page

6 The Web Services Challenge (2007): http://www.wschallenge.org/wsc07/

7 The Services Computing Contest (2007): http://iscc.servicescomputing.org/2007/

8 https://networkx.lanl.gov/

9 http://www.informatik.uni-freiburg.de/~koehler/ipp.html

Seog-Chan Oh has been a researcher at General Motors R&D center since 2007. He earned a BS and an MS from Dongguk University (1993 and 1996, respectively), and a PhD from the Pennsylvania State University (2006). His research interests include Web service composition and its applications to manufacturing field as well as AI, agent, and Semantic Web services. He holds a certificate of professional engineer in Information Management issued by the Korean government.

Dongwon Lee has been an assistant professor at the Pennsylvania State University, College of Information Sciences and Technology, since 2002. He earned a BS from Korea University (1993), an MS from Columbia University (1995), and a PhD from UCLA (2002), all in computer science. His research interests include Databases, XML and Web Analysis, and Semantic Web Services. He has (co-)authored about 70 scholarly articles in conferences and journals.