Effective Web Service Composition in Diverse and Large-Scale Service Networks

Seog-Chan Oh, Dongwon Lee, and Soundar R.T. Kumara

Abstract—Web services are considered to be a potential *silver bullet* for the envisioned *Service Oriented Architecture*, in which loosely coupled software components are published, located, and executed as integral parts of distributed applications. The main research focus of Web services is to achieve the interoperability between distributed and heterogeneous applications. Therefore, flexible composition of Web services to fulfill the given challenging requirements is one of the most important objectives in this research field. However, until now, service composition has been largely an error-prone and tedious process. Furthermore, as the number of available Web services increases, finding the right Web services to satisfy the given goal becomes intractable. In this paper, toward these issues, we propose an AI planning-based framework that enables the automatic composition of Web services, and explore the following issues. First, we formulate the Web service composition problem in terms of AI planning and network optimization problems to investigate its complexity in detail. Second, we analyze publicly available Web service sets using network analysis techniques. Third, we develop a novel Web service benchmark tool called *WSBen*. Fourth, we develop a novel AI planning-based heuristic Web service composition algorithm named *WSPR*. Finally, we conduct extensive experiments to verify WSPR against state-of-the-art AI planners. It is our hope that both WSPR and WSBen will provide useful insights for researchers to develop Web service discovery and composition algorithms, and software.

Index Terms—Web service composition, AI planning, complex network, benchmark.

1 Introduction

Web services are often considered to be one of the most important and vital building blocks for the "Semantic Web" [1]. As such, the industrial support of Web services has grown drastically in recent years. For example, it is expected that by 2007, 72 percent of all application development software will support Web services, and 45 percent of all types of software will be Web services enabled [2]. 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.

The main research focus of Web services is to achieve interoperability between distributed and heterogeneous applications. Therefore, flexible composition of Web services in order to fulfill the requirements of the tasks is one of the most important objectives in this research field. To date, however, many people have considered the service composition to be an ad hoc, time-consuming, and error-prone process involving repetitive low-level programming [3]. To remedy these problems, in recent years, a plethora of research work and products on Web service composition (WSC) problems have appeared. In addition, the Web service research community has hosted competitive programs (e.g., EEE05 [4] and ICEBE05 [5]) to solicit algorithms and softwares to discover pertinent Web services and compose them to make value-added functionality. We first show how a WSC problem typically forms by illustrating a small size problem.

1.1 Motivating Example

In Web service enabled networks, typically a client program first locates a Web service server that can satisfy certain requests from a yellow page (UDDI) and obtain a detailed specification (WSDL) about the service. Then, using the known API in the specification, the client sends a request to the Web service considered via a standard message protocol (SOAP), and in return, it receives a response from the service. Web services are self-explanatory; by interpreting XML tags, applications can understand the semantics of operations. In particular, a problem of practical interest concerns the following two issues. Given a request r, among thousands of candidate Web services found in UDDI: 1) how we can find matching services that satisfy r and 2) how we can compose multiple services to satisfy r when a matching service does not exist. Consider the four Web services in Table 1, as illustrated in WSDL notation:

- Given the hotel name, city, and state information, `findHotel` returns the address and zip code of the hotel.
- Given the zip code and food preference, `findRestaurant` returns the name, phone number, and address of the restaurant with matching food preference and closest to the zip code.

- *S.-C. Oh is with the General Motors Research and Development Center, 30500 Mound Road, Warren, MI 48090. E-mail: seog-chan.oh@gm.com.*
- *D. Lee is with the College of Information Sciences and Technology, Pennsylvania State University, 313 A Information Sciences and Technology Building, University Park, PA 16802. E-mail: dlee@ist.psu.edu.*
- *S.R.T. Kumara is with the Department of Industrial and Manufacturing Engineering, Pennsylvania State University, 363 Leonhard Building, University Park, PA 16802. E-mail: skumara@psu.edu.*

Manuscript received 21 Jan. 2008; revised 25 June 2008; accepted 21 Aug. 2008; published online 11 Sept. 2008.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TSC-2008-01-0004.
Digital Object Identifier no. 10.1109/TSC.2008.1.

TABLE 1
Web Service Interface Examples

```
<xsd:element name='findHotel_Request'>
  <xsd:element name='hotel' type='xsd:string'/>
  <xsd:element name='city' type='xsd:string'/>
  <xsd:element name='state' type='xsd:string'/>
</xsd:element>
<xsd:element name='findHotel_Response'>
  <xsd:element name='addr' type='xsd:string'/>
  <xsd:element name='zip' type='xsd:string'/>
</xsd:element>
```

(a)

```
<xsd:element name='findRestaurant_Request'>
  <xsd:element name='zip' type='xsd:string'/>
  <xsd:element name='foodPref' type='xsd:string'/>
</xsd:element>
<xsd:element name='findRestaurant_Response'>
  <xsd:element name='name' type='xsd:string'/>
  <xsd:element name='phone' type='xsd:string'/>
  <xsd:element name='addr' type='xsd:string'/>
</xsd:element>
```

(b)

```
<xsd:element name='guideRestaurant_Request'>
  <xsd:element name='foodPref' type='xsd:string'/>
  <xsd:element name='currAddr' type='xsd:string'/>
</xsd:element>
<xsd:element name='guideRestaurant_Response'>
  <xsd:element name='rating' type='xsd:string'/>
  <xsd:element name='destAddr' type='xsd:string'/>
</xsd:element>
```

(c)

```
<xsd:element name='findDirection_Request'>
  <xsd:element name='fromAddr' type='xsd:string'/>
  <xsd:element name='toAddr' type='xsd:string'/>
</xsd:element>
<xsd:element name='findDirection_Response'>
  <xsd:element name='map' type='xsd:string'/>
  <xsd:element name='direction'
  type='xsd:string'/>
</xsd:element>
```

(d)

(a) `findHotel`. *(b)* `findRestaurant`. *(c)* `guideRestaurant`. *(d)* `findDirection`.

- Given the current location and food preference, `guideRestaurant` returns the address of the closest restaurant and its rating.
- Given the start and destination addresses, `findDirection` returns a detailed step-by-step driving direction and a map image of the destination address.

Now, consider the following two requests from "State College, PA, USA":

- r_1: find the address of the hotel "Atherton," and
- r_2: find a "Thai" restaurant near the hotel "Atherton" along with a driving direction.

To fulfill r_1, invoking the Web service `findHotel` is sufficient. That is, by invoking `findHotel`("Atherton," "State College," "PA"), one can get the address of the hotel as "100 Atherton Street" with the zip code of "16801." However, none of the four Web services can satisfy r_2 alone. Both Web services `findRestaurant` and `guideRestaurant` can find a "Thai" restaurant near the hotel but cannot provide a driving direction. On the other hand, the Web service `findDirection` can give a driving direction from one location to another but cannot locate any restaurant. Therefore, one has to use a chain of Web services to fully satisfy r_2. There are two possible methods to carry out this task. After obtaining the hotel address using `findHotel`, one can do either of the following:

1. Invoke `guideRestaurant`("Thai," "100 Atherton Street, 16801, PA") to get the address of the closest restaurant, e.g., "410 S. Allen St. 16802, PA." Then, invoke the Web service `findDirection`("100 Atherton Street, 16801, PA," "410 S. Allen St. 16802, PA") to get a driving direction.
2. Invoke `findRestaurant`("16801," "Thai") to get the address of the closest restaurant, e.g., "410 S. Allen St. 16802, PA." Then, invoke the Web service `findDirection`("100 Atherton Street, 16801, PA," "410 S. Allen St. 16802, PA") to get a driving direction.

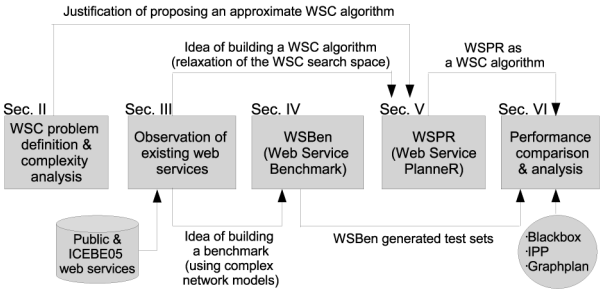


Fig. 1. Organization of this paper.

The above motivating example shows a simple and small WSC problem, wherein multiple Web services must be composed to satisfy a given request. The real-world composition problems, however, are much more complicated than the motivating example, as the number of Web services are rapidly increasing and Web services are connected to other services in different structures depending on applications.

This paper's main contribution is to propose WSBen and WSPR for a Web service benchmark and a WSC algorithm, respectively. Compared to our previous works [6], [7], WSBen is significantly updated by incorporating the complex network theories for populating its testing sets, and WSPR is also improved in terms of effectiveness, efficiency, scalability, and robustness by adopting better heuristics discovered through the study on real Web service networks. To deliver our detailed study on the topic of Web services, we take a developmental progression; formalize the WSC problem, observe the topology of real Web service networks, build benchmarks that simulate these observed topologies, and design a WSC algorithm, and finally evaluate it against the generated benchmarks as well as existing testing sets. Accordingly, our paper has its organization laid out to fit the above developmental progression. For details, as shown in Fig. 1, this paper is organized as follows:

1. Section 2 defines the WSC problem in terms of the AI planning problem, where we will show that the WSC problem is NP-Complete and justify our development of a polynomial-time approximation algorithm to solve the problems.
2. Section 3 observes existing Web services, through which we become aware that the WSC problem can arise in diverse domains whose characteristics can be captured by investigating the underlying network topologies.

TABLE 2
Summary of Notations

Notation	*Meaning*
w,W	Web service, Set of Web services
w^i,w^o	Input and output parameter sets of w
p,P	Parameter, set of parameters
r,r^i,r^o	Request, initial and goal parameter sets of r
Π	Propositional STRIPS model consisting of 4-tuple
Ψ	State space model corresponding to Π
s, S	State, set of states
s_0	Initial state, $s_0 \in S$ such that $s_0 = r^i$
S_G	Goal states $s \in S_G$ such that $r^o \subseteq s$
$\Omega(s)$	Set of Web services $w \in W$ such that $w^i \in s$
$c(w)$	Invocation cost of w
$G_s(V_s, E_s)$	State node network
$G_p(V_p, E_p)$	Parameter node network
$G_{op}(V_{op}, E_{op})$	Operation node network
$G_{ws}(V_{ws}, E_{ws})$	Web-service node network
$G^f_{op}(V^f_{op}, E^f_{op})$	Full-matching operation node network
$G_{cl}(V_{cl}, E_{cl})$	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 ($baTS$, $erTS$, and $nwsTS$ are instances)

3. In Section 4, based on the implications from observing existing Web services, we develop a flexible Web service benchmark tool called WSBen, which utilizes complex network models to generate diverse Web service sets for testing the robustness of existing and newly introduced composition algorithms.
4. In Section 5, we develop an effective WSC algorithm titled WSPR, and in Section 6, we evaluate the performance of our proposal through the experimental validation using multiple criteria including effectiveness, efficiency, scalability, and robustness.

We can indicate three main assumptions of this work. First, Web services defined in this paper are assumed to have no side effects such as delete effects. Second, we assume that Web services can be composed only using their interfaces defined with WSDL and the semantic matchmaking is out of the paper scope. Third, we expect that the interface matching between Web services can be preprocessed by leveraging on off-the-shelf semantic- or context-based models in a different granularity level. This implies

TABLE 3
Summary of Definitions

Definition	*Title*
2.1	Web-Service Discovery Problem (WSD)
2.2	Web-Service Composition Problem (WSC)
2.3	State Node Network ($G_s(V_s, E_s)$)
2.4	Full matching
2.5	Partial matching
3.1	Parameter Node Network ($G_p(V_p, E_p)$)
3.2	Operation Node Network ($G_{op}(V_{op}, E_{op})$)
3.3	Web-Service Node Network ($G_{ws}(V_{ws}, E_{ws})$)
3.4	Random Network
3.5	Regular Network ($Rg_{(N,k)}$)
3.6	Small-World Network
3.7	Scale-free Network
4.1	Parameter Cluster Network ($G_{cl}(V_{cl}, E_{cl})$)

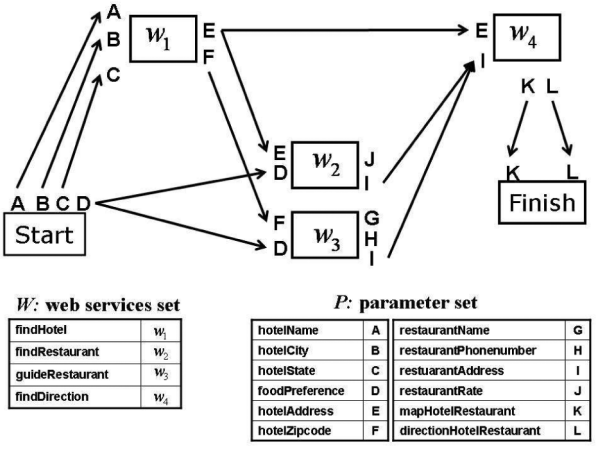


Fig. 2. STRIPS model of the motivating example.

that composition algorithms will simply concentrate on the effective composition alone and not worry about individual matching between parameters (Tables 2 and 3).

2 Problem Definition

A Web service w typically has two sets of parameters: $w^i = \{I_1, I_2, \ldots\}$ for SOAP request (as input) and $w^o = \{O_1, O_2, \ldots\}$ for SOAP response (as output). When w is invoked with all input parameters, w^i, it returns the output parameters, w^o. We assume that in order to invoke w, all input parameters in w^i must be provided (i.e., w^i are mandatory).

Definition 2.1 (Web service discovery (WSD) problem). *Suppose that a request r has initial input parameters r^i and desired output parameters r^o. The WSD problem is defined as to find a set of Web services w, such that 1) $r^i \supseteq w^i$ and 2) $r^o \subseteq w^o$.*

With a simple lookup table, the WSD problem can be easily solved. Therefore, in the remainder of this paper, we focus on the case where no single Web service can fully satisfy the request r, and therefore, one has to compose multiple Web services. This type of problem is referred to as the *WSC* problem and can be formally defined using a planning problem in the STRIPS [8] four-tuple model, $\Pi = \langle P, W, r^i, r^o \rangle$, where

1. P is a set of parameters. In the motivating example in Section 1, $P = \{$"hotel-name," "hotel-city," "hotel-address," $\ldots\}$,
2. W is a set of Web services. In the motivating example, $W = \{$"findHotel," "findRestaurant," $\ldots\}$,
3. $r^i \subseteq P$ is the initial input parameters,
4. $r^o \subseteq P$ is the desired output parameters.

Note that $\Pi = \langle P, W, r^i, r^o \rangle$ is a propositional STRIPS planning in which an initial state is a finite set of ground atomic formulas, indicating that the corresponding conditions are initially true and that all other relevant conditions are initially false. In addition, the preconditions and postconditions of an operator as well as the goals are the ground literals [9]. Fig. 2 illustrates the STRIPS model of the motivating example.

We can transform a STRIPS model Π into a state space model $\Psi = \langle S, s_0, S_G, \Omega(\cdot), f, c \rangle$, where

1. The state $s \in S$ is a collection of parameters in P,
2. The initial state $s_0 \in S$ is such that $s_0 = r^i$,
3. The goal states $s \in S_G$ are such that $r^o \subseteq s$,
4. $\Omega(s)$ is 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,
5. The transition function $f(w, s) = s'$ that maps a state s into a state s' such that $s' = s \cup w^o$ for $w \in \Omega(s)$,
6. $c(w)$ is the invocation cost of w.

A solution of the state model is a finite sequence of Web services $w_1, w_2, \ldots, w_n$, where a sequence of states $s_0, s_1, \ldots, s_n$ exists, such that $s_i = f(w_i, s_{i-1})$ for $i = 1, \ldots, n$, $w_i \in \Omega(s_{i-1})$, and $s_n \in S_G$ [10], [11]. Based on Ψ, the WSC problem can be formally defined as follows:

Definition 2.2 (WSC problem). *Suppose that a request r has initial input parameters r^i and desired output parameters r^o. The WSC problem is to find a finite sequence of Web services, $w_1, w_2, \ldots, w_n$ such that 1) w_i is invoked sequentially from 1 to n, 2) $(r^i \cup w_1^o \cup \ldots \cup w_n^o) \supseteq r^o$, and 3) the total cost $\sum_{i=1}^{n} c(w_i)$ is minimized.*

The WSC problem can be considered to be the information gathering problem [12], where Web services represent information sources and interleaving between Web services is not found. For that reason, WSC algorithms model the world state as an information state, which is a description of the information collected by the algorithm at a particular stage in composition [13]. Therefore, the WSC problem is a relatively simple delete-free problem from a planning perspective but has quite different characteristics than a usual planning problem. Classical planning problems (e.g., blocks world problem) have generally considered a small number of actions (e.g., move block) in the assumption of a large number of initial and goal propositions (e.g., hundreds of blocks to move). On the contrary, WSC problems generally deal with a large number of actions (e.g., hundreds of travel agent services) with a limited number of propositions involved (e.g., registering one hotel) [14].

Definition 2.3 (state node network). *A state node network is a directed graph $G_s(V_s, E_s)$, where $s_i \in V_s$ represents a state, and E_p is a set of directed edges (or arcs) (s_i, s_j) that connects ordered pairs of $s_i \in V_s$ and $s_j \in V_s$. Every arc $(s_i, s_j) \in E_s$ is weighted by the invocation cost $c(w)$, where $w \in \Omega(s_i)$ and $s_j = f(w, s_i)$ (i.e., $s_j = s_i \cup w^o$).*

Given a network $G_s(V_s, E_s)$, we consider paths in which arcs are traversed only in the forward direction. The cost of a path is the sum of the costs of the associated arcs. Our interest is to obtain the shortest path between s_0 and $s_n (\in S_G)$. Fig. 3 illustrates the $G_s(V_s, E_s)$ of the motivating example. Note that s_5, s_6, and s_7 contain the goal parameters, "K" and "L," where "K" and "L" refer to "mapHotelRestaurant" and "directionHotelRestaurant," respectively. Therefore, s_5, s_6, and s_7 belong to S_G. It is evident that if $c(w_i) = 1$ for all $w_i \in W$, then the shortest paths are: 1) s_0, s_1, s_2, $s_5 (w_1 \rightarrow w_2 \rightarrow w_4)$ and 2) s_0, s_1, s_3, s_7 $(w_1 \rightarrow w_3 \rightarrow w_4)$. Indeed, the problem of finding the shortest paths from s_0 to $s_n (\in S_G)$ can be formulated as the minimum cost flow problem.

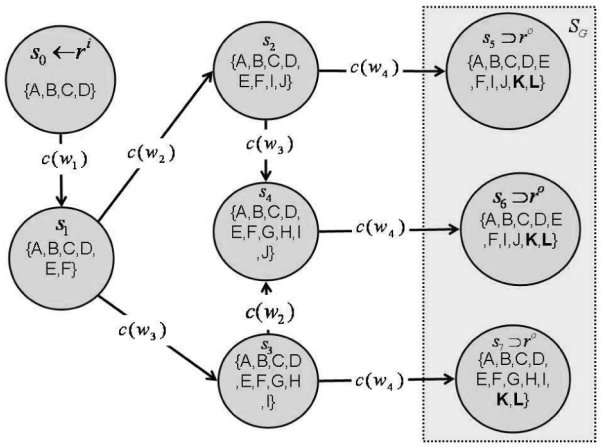


Fig. 3. $G_s(V_s, E_s)$ of the motivating example.

Let b_i denote the amount of flow that enters the network at node $s_i \in V_s$, $i = 0, \ldots, n$. Let $fl_{i,j}$ denote the amount of flow that is greater than 0 on arc $(s_i, s_j) \in E_s$. Note that $fl_{i,j}$ will automatically be an integer due to the unimodularity property of network flow problem. If $b_i > 0$, the node is a source that supplies b_i units of flow. If $b_i < 0$, the node is a sink that demands b_i units of flow. In the minimum cost flow problem, one unit of flow is analogous to be a pathfinder in a real world who finds a shortest path from a source node to a sink node. Suppose $s_0 = r^i$, and $s_n \in S_G$. Although we are interested in obtaining the shortest path between s_0 and s_n, it is also possible to consider the issue of finding the shortest path from all states to a given goal state by setting $b_i = 1$, for $i = 0, \ldots, n-1$ and $b_n = -n$. Simply, this will put a supply of one unit at every other non-goal state node, and a demand of n at the goal state node so that the total supply and demand are equal. Suppose that a cost of $c_{i,j}$ per unit flow on arc $(s_i, s_j) \in E_s$ is $c(w)$, where $s_j = f(w, s_i)$. Then, the minimum cost flow problem is given as follows:

$$minimize \sum_{(s_i, s_j) \in E_s} c_{i,j} fl_{i,j} \tag{1}$$

subject to

$$\sum_{(s_i, s_j) \in E_s} fl_{i,j} - \sum_{(s_k, s_i) \in E_s} fl_{k,i} = b_i, \tag{2}$$

$$fl_{i,j} \geq 0 \qquad \forall (s_i, s_j) \in E_s. \tag{3}$$

This formula indicates that flows must be feasible and each node conserves flow. Specifically, Objective function (1) indicates that we are interested in getting the shortest plan (or path). Constraint (2), called the "flow conservation equations," indicates that the flow may be neither created nor destroyed in the network. In the conservation equation, $\sum_{(s_i, s_j) \in E_s} fl_{i,j}$ represents the total flow out of node s_i, while $\sum_{(s_k, s_i) \in E_s} fl_{k,i}$ indicates the total flow into node s_i. This equation requires that the net flow out of node s_i, $\sum_{(s_i, s_j) \in E_s} fl_{i,j} - \sum_{(s_k, s_i) \in E_s} fl_{k,i}$, should be equal to b_i. Note that if $b_i < 0$, then there should be more flow into i than out of i. Feasible flows exist when $\sum_{i=0}^{n} b_i = 0$.

This minimum cost flow problem can have a dual problem. With $b_0 = b_1 = \cdots = b_{n-1} = 1$, the dual problem with $\lambda_n = 0$ is

$$maximize \sum_{i=0}^{n-1} \lambda_i \tag{4}$$

subject to

$$\lambda_i \leq c_{i,j} + \lambda_j \qquad \forall (s_i, s_j) \in E_s. \tag{5}$$

λ_i refers to the cost associated with the path from s_i to s_n. This maximization problem suggests that in the optimal solution, λ^*, if all components are fixed except for λ_i^*, then λ_i^* tends to become as large as possible and is subject to the feasibility constraint (5). Therefore, λ_i^* satisfies the following Bellman equation with $\lambda_n^* = 0$:

$$\lambda_i^* = \min_{(s_i, s_j) \in E_s} \left\{ c_{i,j} + \lambda_j^* \right\}, \qquad i = 0, \ldots, n-1. \tag{6}$$

λ_i^*s are also called *labels*, and there are polynomial algorithms to solve the equation outlined in (6), such as the label setting algorithm (when $c_{i,j}$ is nonnegative) or label-correcting algorithm [15]. However, there are problems with this idea. These algorithms are polynomial in the size of nodes $|V_s|$, but the number of nodes are exponential in the number of parameters $|P|$. This is because nodes are information states, and the information state is a collection of parameters in P. Before we proceed to investigate the complexity of the WSC problem in detail, we need to introduce two matching operations described below.

Definition 2.4 (full matching). *Suppose that a state $s \in S$ is given. Let a Web service $w_1 \in \Omega(s)$. If for $w_2 \in W$, $w_1^o \supseteq w_2^i$, then w_1 can "fully" match w_2.*

Definition 2.5 (partial matching). *Suppose that a state $s \in S$ is given. Let a Web service $w_1 \in \Omega(s)$. If for $w_2 \in W$, $(w_1^o \not\supseteq w_2^i)$ and $(w_1^o \cap w_2^i \neq \emptyset)$, then w_1 can "partially" match w_2.*

In the motivating example, `findHotel` partially matches `findRestaurant` and `guideRestaurant`. In turn, `findRestaurant` and `guideRestaurant` partially match `findDirection`. However, if both `findHotel` and `findRestaurant` are composed, then `findDirection` can be fully matched. When only full matching is considered in the WSC problem, it can be seen as a single-source shortest path problem, which is defined over $G_s(V_s, E_s)$. On the other hand, when both full and partial matching operations 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 postconditions.

Theorem 2.6. *The WSC problem with full and partial matching operations is NP-complete.*

Proof. For proof, see Appendix A. □

Theorem 2.6 implies that when the number of Web services to search is not small, finding an optimal solution to the WSC problem defined in Definition 2.2 is prohibitively expensive. Therefore, we will propose an AI-planner-based approximate algorithm instead. Note that if not specially provided, a WSC problem is considered to have both full and partial matching cases throughout this paper.

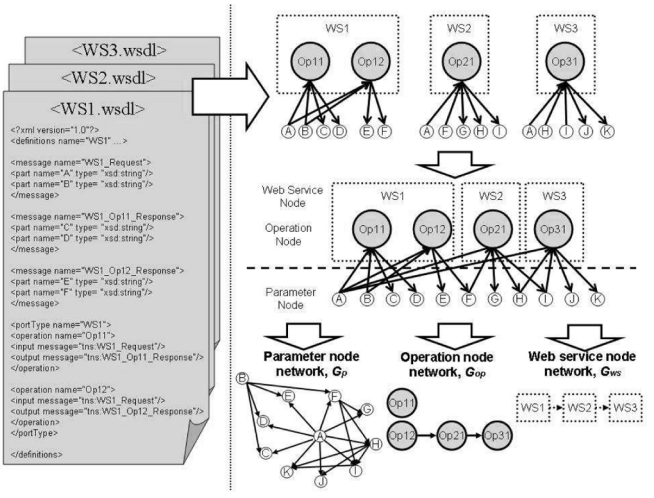


Fig. 4. Web service networks.

3 Study of Existing Web Services

In this section, we will observe public Web services and the ICEBE05 test sets and reveal their network features by means of complex network properties. For this purpose, we first define Web service networks in a novel way, as it is then easy to investigate the real-world Web service structure.

A set of Web services forms a network (or directed graph). There are different kinds of models to determine nodes and edges of the network depending on the granularity level: Web service level (coarse granularity), operation level, and parameter level (fine granularity) models. Fig. 4 illustrates that three WSDL files can be converted into a bipartite graph structure that consists of three distinct kinds of nodes (parameter, operation, and Web service node) and directed edges between bipartite nodes (operation nodes and parameter nodes). An edge incident from a parameter node to an operation node suggests that the parameter is one of the inputs of the corresponding operation. Conversely, an edge incident from an operation node to a parameter node implies that the parameter is one of the outputs of the corresponding operation. The graph in Fig. 4 has three Web services, labeled WS1, WS2, and WS3. WS1 has two operations, Op11 and Op12. WS2 and WS3 have one operation, Op21 and Op31, respectively. The graph also displays 11 parameters, labeled A through K. According to the node granularity, we can project the upper graph into three different Web service networks.

Definition 3.1 (parameter node network). *A parameter node network is a directed graph, $G_p(V_p, E_p)$, in which V_p is a set of all parameter nodes and E_p is a set of directed edges from input parameters $p_i \in V_p$ to output parameters $p_j \in V_p$; i.e., there exists an operation that has an input parameter matching p_i and an output parameter matching p_j.*

Definition 3.2 (operation node network). *An operation node network is a directed graph $G_{op}(V_{op}, E_{op})$, in which V_{op} is a set of all operation nodes and E_{op} is a set of directed edges from*

operation $op_i \in V_{op}$ to operation $op_j \in V_{op}$; i.e., op_i can fully or partially match op_j.

Definition 3.3 (web service node network). *A Web service node network is a directed graph $G_{ws}(V_{ws}, E_{ws})$, in which V_{ws} is a set of all Web service nodes and E_{ws} is a set of directed edges from Web service node $ws_i \in V_{ws}$ to $ws_j \in V_{ws}$; i.e., there exists one or more edges between any operation in ws_i and any operation in ws_j in an operation node network.*

For example, in Fig. 4, $A \rightarrow Op11 \rightarrow C$ is projected into $A \rightarrow C$ in the parameter node network, G_p. Similarly, since Op12 partially matches Op21 and, subsequently, Op21 partially matches Op31, Op12 $\rightarrow$ Op21 $\rightarrow$ Op31 is shown in the operation node network, G_{op}. Also, since WS1 possesses Op12, and WS2 possesses Op21, WS1 $\rightarrow$ WS2 appears in the Web service node network, G_{ws}. As mentioned previously, in operation node networks, a directed arc, $(i, j) \in E_{op}$ suggests that $i \in V_{op}$ can match $j \in V_{op}$ either fully or partially. Slightly differently, we can define a new type of operation node network by restricting the partial match and only allowing the full match. We differentiate this operation node network by using the *f* symbol: $G^f_{op}(V^f_{op}, E^f_{op})$.

To investigate the properties of public Web services, we first downloaded 1,544 raw WSDL files (downloadable at http://rakaposhi.eas.asu.edu/PublicWebServices.zip) that Fan and Kambhampati [16] gathered from real-world Web service registries such as `Bindingpoint`, `Salcentral`, `WebserviceList`, `WebserviceX`, and `xMethods`. After weeding out invalid WSDL files that do not conform to the WSDL DTD, 670 valid WSDL files remained. Then, we converted semantically invalid parameters such that those parameters can capture their underlying semantics when they have evident semantic sources. For example, many operations of WSDL files have terms like "result(s)" or "return(s)" in their output parameters, which make it hard to interpret what the terms mean; i.e., what the "results" or "returns" means. In this case, we replaced the terms with their operation name or Web service name. Sometimes, an operation name is a sequence of concatenated words; therefore, it is hard to do the replacement tasks. In such a case, we did a proper token segmentation and extracted the primal terms from the tokens using lexical analysis. For example, if an operation name is "getAuthornamefromPaper" or "searchAuthornamesbyPaper," we extracted "Authornames" and used it to replace "result(s)" or "returns(s)." After these preprocessing tasks, we built a set, $P =$ {all parameters found in the valid 670 WSDL files}.

3.1 Complex Network Models

Complex network models have been used to investigate real-world networks. Especially, "random," "small-world," and "scale-free" types have been shown to model many real-world networks sufficiently [17]. Commonly, in random, scale-free, and small-world complex networks, nodes signify the elements of the system and edges represent the interactions between them.

Definition 3.4 (random network). *A random network is defined with 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 [18].*

Definition 3.5 (regular network). *$Rg_{(N,k)}$ is defined as a regular network on N nodes, if node i is adjacent to nodes $[(i + j) mod\ N]$ and $[(i - j) mod\ N]$ for $1 \leq j \leq k$, where k is the number of valid edge of each node. If $k = N - 1$, $Rg_{(N,k)}$ becomes a complete N-node 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 [19] with probability p. L_{random} is identical to $L(1)$.
- C: the average clustering coefficient. Suppose that for a node i with v_i neighbors, $C_i = \frac{2E_i}{v_i(v_i-1)}$, 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 3.6 (small-world network). *Small-world networks are characterized by a highly clustered topology like regular lattices and small average short distance. Specially, $C \gg C_{random}$ and $L \gtrsim L_{random}$ [20].*

By using the Watts-Strogatz model [21], [19], we can construct networks that have small-world network 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 an $Rg_{(N,k)}$ 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, then (i, j) is removed from the graph. The long-range connections (shortcuts) 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 and has a small average path length.

Definition 3.7 (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.*

Albert and Barabasi defined several extended models [22], [23], [20] to provide the scale-free properties. The extended model uses an algorithm to build graphs that

TABLE 4
Features of Public Web Service Networks

Features	G_p	G_{op}	G_{ws}
# of nodes	11,301	5,180	984
# of arcs	81,897	161,647	15,968
Network diameter	21	10	6

TABLE 5
Small-World Network Properties of Giant Components in the Public Web Services

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_p	4.3185	3.4244	0.2229	0.0021
G_{op}	2.859	1.983	0.3056	0.0362
G_{ws}	2.271	1.9222	0.4809	0.0874

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), and q (probability of edge rewiring). The procedure starts with m_0 isolated nodes and performs one of the following three actions at every step:

1. 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:

$$\Pi_i = \frac{v_i + 1}{\sum_j (v_j + 1)}, \tag{7}$$

   where Π_i is the probability of selecting the ith node, and v_i is the degree of node i. The process is repeated m times.
2. 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 (7), and the new link l_{iz} is added.
3. 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 (7).

Once the desired 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 there are no correlations among the edges.

3.2 Public Web Services

Table 4 shows the general information about each of the public Web service networks. First, we investigate the scale-free network properties of public Web services. Figs. 5 and 6 show G_p, G_{op}, and G_{ws} of public Web services with their corresponding outgoing edge distributions. In the outgoing edge distributions, the x-axis represents the number of outgoing edges and the y-axis represents the number of nodes with the same outgoing edges. We can apply the power function, $P_w(v)$, to each of the outgoing edge distributions and check γ, the exponent value of $P_w(v)$. Even though all γ values do not exceed 2, they are highly skewed so that their shape is similar to the Zipf distribution.

We also investigated the small-world network properties of public Web services. Table 5 shows the average path length L and clustering coefficient C for three giant connected components extracted from each of the three public Web service networks (G_p, G_{op}, and G_{ws}), compared to random graphs with the same number of nodes and average degree of a node. It is interesting that G_p, G_{op}, and G_{ws} show small-world network properties: $L \gtrsim L_{random}$ and $C \gg C_{random}$. This result suggests that public Web

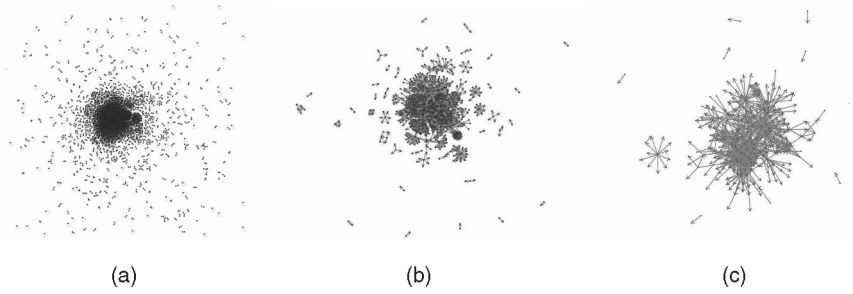


Fig. 5. Public Web services. (a) G_p. (b) G_{op}. (c) G_{ws}.

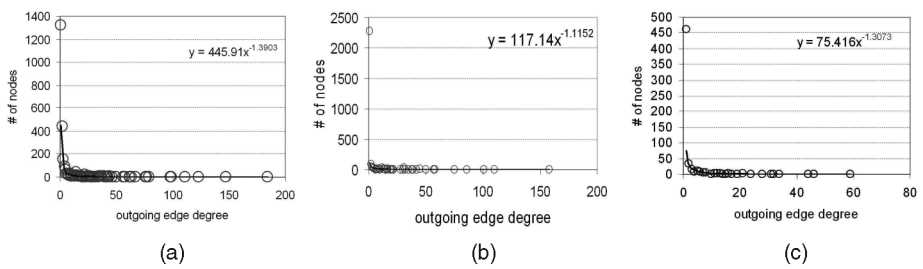


Fig. 6. Outgoing edge distribution of public Web services. (a) G_p. (b) G_{op}. (c) G_{ws}.

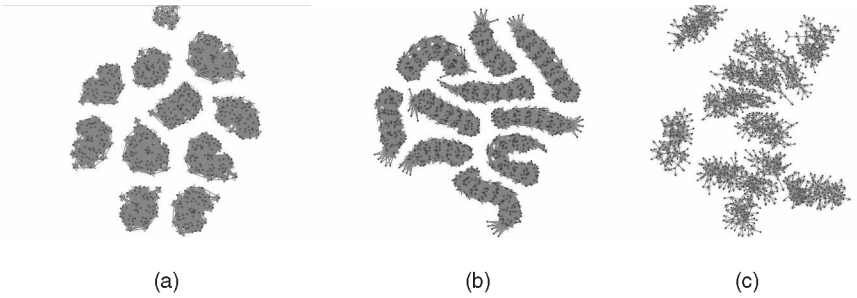


Fig. 7. ICEBE05 test set. (a) G_p. (b) G_{op}. (c) G_{ws}.

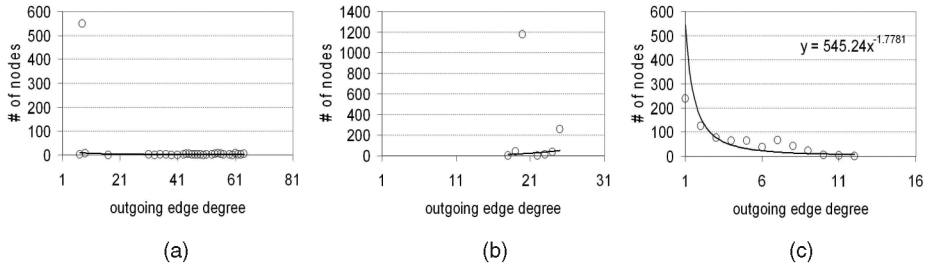


Fig. 8. Outgoing edge distribution of ICEBE05 test set. (a) G_p. (b) G_{op}. (c) G_{ws}.

services have such "shortcuts" to connect nodes that would otherwise be much farther apart than L_{random}.

3.3 ICEBE05 Test Sets

The ICEBE05 [5] test sets are autogenerated from software by the ICEBE05 organization, who provides 18 test sets. We have selected "composition2-20-4," as a representative of the 18 test sets because all of them are similar in nature. Figs. 7 and 8 show G_p, G_{op}, and G^f_{op} with their outgoing edge distributions. The shapes of the graphs differ considerably from the real public Web services. Each graph consists of 10 islands, and those islands (the connected component) are uniform in terms of their network topologies (number of nodes, number of arcs, and the connectivity pattern). Each island is likely to approximate workflow Web service domains (e.g., business, scientific, and medical workflows). The parameters used in workflows tend to be domain specific or constitute professional terms, and thus their G_ps are likely to have a regular network property. Moreover, the nodes in G_{op} and G^f_{op} of workflow domains are likely to be connected to a few number of neighboring Web services in the succeeding stage of workflow. This can occur because workflows can be constructed such that the underlying networks do not follow scale-free network properties, in order to avoid skewed resource consumption or increase the network survivability.

Table 6 shows the general information about each of the ICEBE05 Web service networks. Note that all Web services in ICEBE05 have only one operation, meaning that G_{op} and G_{ws} are identical. This is why the network diameters of G_{op} and G^f_{op} are 7 identical. In fact, all test requests of ICEBE05 can be solved using the full matching operation alone. In the event that the WSC problem has this kind of special scheme, it can be addressed quickly using a simple shortest path algorithm with a single source.

Considering Fig. 8, 86 percent of the nodes of G_p have an outgoing edge degree of 7 uniformly and 77 percent of the nodes of G_{op} has an outgoing edge degree of 20 uniformly. Hence, the distribution can be fitted by the peak distribution, and this peak shape is the strong evidence to prove that G_p and G_{op} do not follow the scale-free network, which is the case with the public Web services (refer to Fig. 6). On the contrary, the outgoing edge distribution of G^f_{op} has $\gamma = 1.5927$, so that it can be regarded as the Zipf-like distribution. However, it is not clear if it has scale-free properties, because the largest outgoing degree is just 12. This number is too small to view the node as a hub, considering that the total node number of G^f_{op} is 1,229.

Table 7 shows the average path length L and clustering coefficient C for three giant connected components extracted from each of the three ICEBE05 Web service networks, compared to random graphs with the same number of nodes and average degree of a node. G_p shows the random network characteristics: $L \approx L_{random}$ and $C \approx C_{random}$. On the contrary, G_{op} and G^f_{op} show special characteristics: $C = 0$ and $L \approx L_{random}$. As an extreme case, if

TABLE 6
Features of the ICEBE05 Web Service Networks

Features	G_p	G_{op}	G^f_{op}
# of nodes	736	1,774	1,229
# of arcs	8,569	31,905	2,599
Network diameter	7	7	7

TABLE 7
Small-World Network Properties of Giant Connected Component in the ICEBE05 Web Services

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_p	1.7689	1.6905	0.3188	0.3055
G_{op}	3.1613	1.8122	0	0.1883
G^f_{op}	4.8629	3.4552	0	0.0345

a graph is a tree in which no circle exists, then C becomes 0, because there are no triangles in the graph.

In summary, we observed existing Web services and learned several implications, which are given as follows:

- WSC problems can arise in diverse scenarios. However, they can be captured by investigating which of the network topologies would fit into their Web service networks, especially complex networks.
- Conversely, we can use the complex network models to generate Web service test sets. This idea is fundamental to the design of WSBen.
- A Web service network can be relaxed into a parameter node network by ignoring operation and Web service information. This relaxation idea can be adopted for building a new WSC algorithm. In fact, WSPR conducts a polynomial-time forward search over the relaxed parameter-based search space.

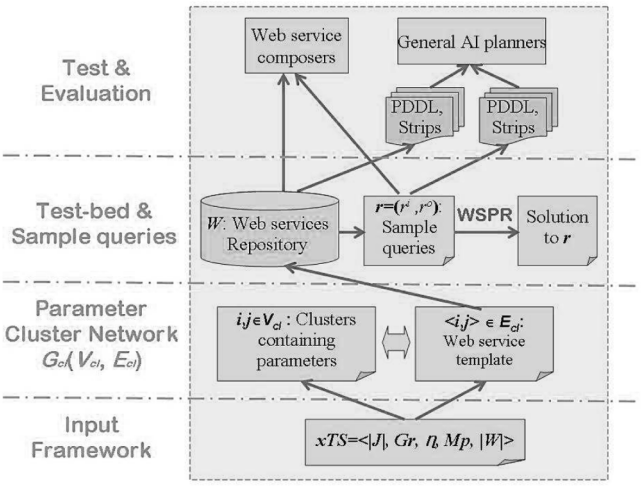


Fig. 9. Overview of WSBen.

4 WSBen: Web Services Discovery and Composition Benchmark Tool

In this section, we present a novel benchmark tool titled WSBen [6], which provides a set of functions to simplify the generation of test environments for WSD and WSC algorithms. The main contributions of WSBen is to provide diverse Web service test sets based on three complex network models such as "random," "small-world," and "scale-free" types. Note that we already showed in Section 3.2 that real Web service networks have characteristics of "small-world" and "scale-free" complex networks. At a higher level, a Web service can be assumed to be a transformation between two different application domains, and each can be represented by a parameter cluster. This assumption is the basis for developing WSBen. From the perspective of graph theory, WSBen builds a *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 4.1 (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 nonoverlapping parameters denoted by Pa_i and Pa_j, respectively, where $Pa_i \cap Pa_j = \emptyset$. Each directed edge is also called a Web service template from which WSDL files are generated.*

Fig. 9 shows the overview of WSBen, which 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 $xTS = \langle |J|, G_r, \eta, M_p, |W| \rangle$. xTS applies existing complex 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, $\langle i, j \rangle \in E_{cl}$ 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 $r = \langle r^i, r^o \rangle$. 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 [24] and STRIPS formats, enabling concurrent comparison with state-of-the-art AI planners.

Considering $xTS = \langle |J|, G_r, \eta, M_p, |W| \rangle$, if the first four tuples are grounded, one can build a parameter cluster network, where clusters are nodes and Web service templates are directed edges. 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 one of the following three models that are developed to simulate "random," "small-world," and "scale-free" complex networks, respectively:
   - 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. Note that the generation of this graph costs $O(|J|^2)$ because it starts with creating $\frac{|J|(|J|-1)}{2}$ edges.
   - Newman-Watts-Strogatz($|J|, k, p$): The initialization is a regular ring graph with k neighbors. During the generation process, new edges are added randomly with probability p for each edge. Note that no edges are removed, differing from the 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 $\gamma = 2.9 \pm 0.1$, 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. Cooccurrence probability of each cluster is measured by the following probability:

$$\Delta_j = \frac{k_j}{\max_{j \in V_{cl}} k_j} \eta, \tag{8}$$

where Δ_j is the cooccurrence 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. Δ_j is designed to reflect the observation discovered in Section 3.2, where small number of hub parameters (e.g., name, postal code, and password) exclusively possess most of edges in Web service networks.

3. $|Pa_j|$ is measured based on the following equation:

$$|Pa_j| = \frac{M_p}{\Delta_j}, \tag{9}$$

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., $\forall i,\ j \in V_{cl},\ Pa_i \cap Pa_j = \emptyset$).

Once a complete parameter cluster network $G_{cl}(V_{cl}, E_{cl})$ is built, WSBen generates $|W|$ number of Web services.

Fig. 10 illustrates how WSBen builds G_{cl} and generates WSDL files based on the G_{cl}. Suppose that $xTS = \langle 8, \text{Barabasi-Albert}(8, 2), 0.8, 1.5, 100\rangle$ is given. Then, the generation steps are given 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 Fig. 10. That is, $|Pa_5| = 9$, as $\Delta_5 = \frac{k_j}{\max_{j \in V_{cl}} k_j} \times \eta = \frac{1}{5} \times 0.8 = 0.16$, resulting in $|Pa_5| = \frac{M_p}{\Delta_i} = \frac{1.5}{0.16} \simeq 9$.
3. Pa_j is specified. For example, $P_5 = \{17, 18, 19, 20, 21, 22, 23, 24, 25\}$ as shown in Fig. 10 because $|P_5| = 9$ and for $\forall j \in V_{cl}$, $P_5 \cap Pa_j = \emptyset$. 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 by

   a. Randomly choosing $\langle i, j\rangle \in E_{cl}$.
   b. Selecting input parameters from cluster i with the cooccurrence probability of Δ_i and output parameters from cluster j with the cooccurrence probability of Δ_j.

   For example, in Fig. 10, `ws1` is instantiated from a Web service template $\langle 3, 1\rangle \in E_{cl}$. 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 is 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. Note that each parameter in one cluster can map into either an input parameter or an output parameter. In the case where no parameter is generated, dummy parameters "S" and "T" are filled in the input and output parameters, respectively.

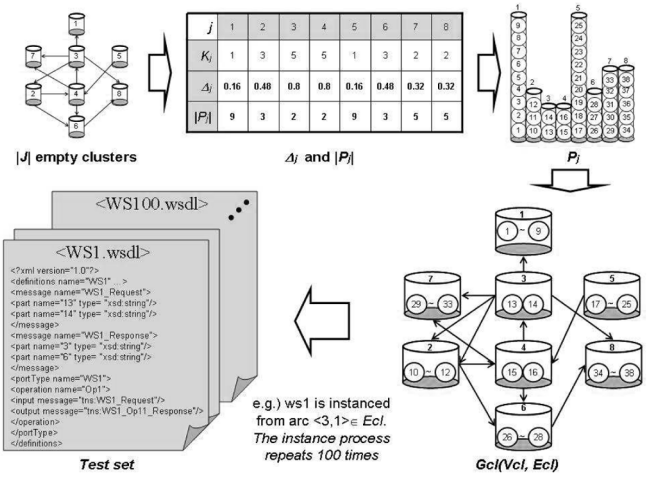


Fig. 10. Test set generation with $\langle 8, \text{Barabasi-Albert}(8, 2), 0.8, 1.5, 100\rangle$.

For experimental purposes in Section 6, we build three test set frameworks by specifying xTS as follows:

1. $baTS = \langle 100, \text{Barabasi-Albert}(100, 6), 0.8, 5, |W|\rangle$,
2. $nwsTS = \langle 100, \text{Newman-Watts-Strogatz}(100, 6, 0.1), 0.8, 5, |W|\rangle$,
3. $erTS = \langle 100, \text{Erdos-Renyi}(100, 0.06), 0.8, 5, |W|\rangle$.

Note that if the first four elements of xTS are grounded, a G_{cl} is generated. For each G_{cl} of the three test set frameworks, seven different sizes of test sets are generated by varying $|W|$ as 1,000, 3,000, 5,000, 10,000, 20,000, 30,000, and 50,000, respectively. This results in 21 test sets (three frameworks $\times$ seven different test sizes). Each G_{cl} of the three test set

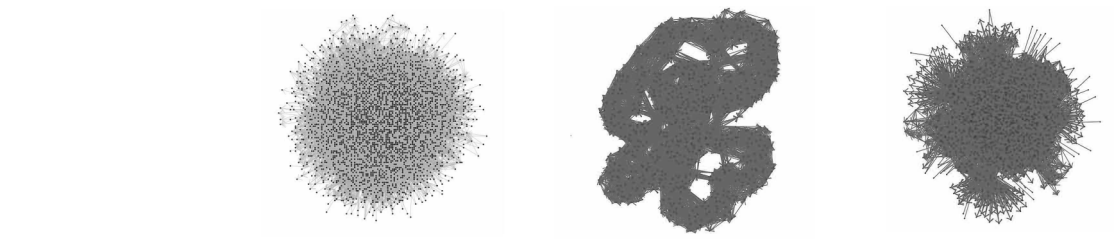


Fig. 11. G_p of $baTS$, $nwsTS$, and $erTS$ when $|W| = 1{,}000$.

frameworks has a different $|P|$. For example, $baTS$ has 4,231, while $nwsTS$ and $erTS$ have 751 and 1,392, respectively. Fig. 11 shows the G_p for each of $baTS$, $nwsTS$, and $erTS$ when $|W| = 1{,}000$. Each of the 21 test sets has five test requests. The test request r is constructed such that r^o is farthest away from r^i in a parameter space. In order to create a test request r, we used WSBen, 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 $g_{r^i}(p)$. The details about obtaining $g_{r^i}(p)$ will be discussed in Section 5. 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 requests for each test set.

5 WSPR: Web Service PlanneR Algorithm

We showed that the size of state space is exponential to the size of the parameter set. To address this intractable WSC problem, we propose a polynomial-time heuristic algorithm titled WSPR, which is a type of AI planning algorithm with two search steps. When a request r is given, first, WSPR computes the cost of achieving individual parameters starting from r^i by conducting the forward search. Second, it approximates the optimal sequence of Web services that connects r^i to r^o by conducting the regression search, leveraging on the results obtained from the first step as guidance.

This two-step-based approach is essentially in accordance to Graphplan [25]. Especially, our forward search corresponds basically to the algorithm used by one of the best known planners, FF [26] that uses relaxed Graphplan by ignoring delete lists. However, our method is significantly different in that we use a novel heuristic to minimize the number of Web services in a solution. In contrast, Graphplan and other AI planners originated from Graphplan typically aim at minimizing the number of time steps but not necessarily the number of actions (Web services).

(Step 1) forward search. In the first stage, WSPR obtains $g_{r^i}(p)$—the cost of achieving $p \in P$ from a state r^i. This cost can be characterized by the solution of a recursive equation as follows:

$$g_{r^i}(p) = \min_{w \in Ow(p)} \left[c(w) + \max_{p' \in w^i} g_{r^i}(p') \right], \tag{10}$$

where $c(w)$ is an invocation cost of a Web service. Since the average service response time can be seen as an invocation cost, in this case, we will assume $c(w) = 1$. $Ow(p)$ is a set of Web services: $Ow(p) = \{w \in W | p \in w^o\}$. 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). 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 for $\forall p \in r^o$, $g_{r^i}(p)$ are obtained (lines 2-6 in Algorithm 1). If $\Omega(s)$ does not increase further, there remains no additional search space, which means that no solution exists. We name a Web service w as a predecessor Web service of $p \in P$ if w is the first Web service to generate p. We denote $PD_{ws}(p)$ to be an inverted index [15] that contains a set of predecessor Web services for p. In this paper, we assume that the invocation cost of Web services is nonnegative. However, it is possible to have a negative Web service invocation cost if various Qualities of Service (QoS) are considered (e.g., cost, quality, security). In that case, rather than (10), label-correcting algorithms such as the Bellman-Ford algorithm [15] must be used.

Algorithm 1: Forward search algorithm of WSPR.

```
   Input  : r^i and r^o
   Output: PD_ws
1  s = r^i; C = ∅; d = 1;
2  while ¬(s ⊇ r^o) do
3      δ={w | w ∈ Ω(s), w ∉ C};
4      for p in w^o (w ∈ δ) do
5          if g_{r^i}(p) = ∞ then
6              g_{r^i}(p) = d; PD_ws(p) = PD_ws(p) ∪ w; s =
               s ∪ {p};
7      C = C ∪ δ; d++ ;
```

We can prove the correctness of the forward search of WSPR by using the loop invariants technique [15].

Theorem 5.1 (correctness of forward search). *The forward search of WSPR runs on* $\Psi =< S, s_0, S_G, \Omega(\cdot), f, c >$ *with* $c(w) = 1$ *for all* $w \in W$ *and terminates after obtaining* $g_{r^i}(p)$ *for all parameter* $p \in S_G$ *if at least a path exists from* r^i *to* r^o.

Proof. For proof, see Appendix B. □

(Step 2) regression search. In the second stage, WSPR approximates the optimal sequence of Web services that connects r^i to r^o by conducting the regression search, as directed by $g_{r^i}(p)$ and $PD_{ws}(p)$, which are obtained from the first step. In this paper, we propose a greedy algorithm-based backward search. The backward search is an old idea in planning that is also known as *regression search*. In the

regression search, state s can be thought as a set of effects and we can specify a subgoal from state s. This algorithm denotes its subgoal by $subGoal$ and sets $subGoal$ to r^o in the beginning (line 1 in Algorithm 2). We can denote $wSpace$ to be a set of Web services $w \in W$, such that $w_i \in PD_{ws}(p)$, where $p \in subGoal$. Then, WSPR selects a Web service from $wSpace$ by considering their heuristics at each backward step (lines 3-6 in Algorithm 2). This backward selection procedure is repeated until $subGoal \subseteq r^i$ (line 2 in Algorithm 2). The heuristic used for selecting a Web service and its underlying hypothesis is given as follows:

Hypothesis 1. *Choosing a Web service with a greater contribution to match subGoal earlier in the regression search helps reach the initial state faster:*

$$h_{sg}(w) = [w^o \cap subGoal]. \tag{11}$$

$h_{sg}(w)$ implies that WSPR favors a Web service with a bigger contribution to match the subgoal. However, $h_{sg}(w)$ has another important interpretation. A Web service w with bigger $h_{sg}(w)$ has a higher probability to match the subgoal fully, leading to preventing proliferation of the following search spaces. In other words, our heuristic attempts to avoid a partial matching case or reduce the size of partial matching Web services as much as possible.

In the case that multiple Web services with the same $h_{sg}(w)$ value are in the OPEN list [15], we choose a Web service with the largest edge degree, as a tie-break rule. This rule can increase the chance of finding better Web services by fertilizing the succeeding search space, because the Web service with the larger edge degree is likely to be connected to the more Web services.

Experimentally, we will assess the performance difference provided by the WSPR heuristic. To this end, we compare the effectiveness and efficiency of WSPR with those of WSPR without the heuristic.

Algorithm 2: Regression search algorithm of WSPR.

Input : r^i, r^o, and PD_{ws}
Output: $w_1 \Rightarrow \cdots \Rightarrow w_n$
1 $subGoal = r^o$;
2 **while** $\neg(subGoal = \emptyset)$ **do**
3 $\quad wSpace = \bigcup_{p \in subGoal} PD_{ws}(p)$;
4 $\quad \chi = \arg \max_{w \in wSpace} h_{sg}(w)$;
5 $\quad soln = soln \cup \chi$;
6 $\quad subGoal = [subGoal \setminus (\chi^o \cup r^i)] \cup \chi^i$;
7 $s = r^i$;
8 **while** $\neg(soln = \emptyset)$ **do**
9 $\quad$ **if** $w \in \Omega(s)$ *and* $w \in soln$ **then**
10 $\quad\quad$ Print w, "$\Rightarrow$";
11 $\quad\quad s = s \cup w^o$;
12 $\quad\quad soln = soln \setminus w$;

We can use the loop invariants technique to prove the correctness of the regression search of WSPR.

Theorem 5.2 (correctness of regression search). *The regression search of WSPR terminates after obtaining a set of Web services to form a path from r^i to r^o.*

Proof. For proof, see Appendix C. □

The forward search procedure has the polynomial computation time $O(|W|^2|P|)$. First, the length of a sequence of Web services to satisfy a request is limited by $|W|$. Therefore, there are at most $|W|$ iterations. Second, at each iteration of the forward search, the maximum $|W|$ Web services and $|P|$ parameters are examined. Consequently, the computational complexity of the forward search procedure costs $O(|W|^2|P|)$. On the contrary, the regression search procedure has the polynomial computation time $O(|W|^2 log|W|)$. First, the regression search procedure has at most $|W|$ iterations, and at each iteration, the maximum $O(|W|log|W|)$ time is required to conduct the sorting task to select a Web service w with the largest $h_{sg}(w)$. The time taken to print a solution can be ignored (lines 8-12 in Algorithm 2). In general, $|P| \gg log|W|$, so that $O(|W|^2|P|) \gg O(|W|^2 log|W|)$. In other words, the forward search takes much longer than the regression search. As a result, the performance of WSPR is dominated by the forward search procedure. In this remark, we find that there are three significant dimensions to determine the performance of WSPR: 1) the length of a sequence of Web services in a solution, 2) $|W|$ of the Web service size, and 3) $|P|$ of the parameter set size.

We can use (10) 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 denoted by A, we can regard the following cost function:

$$g^{max}_{r^i}(A) = \max_{p \in A} g_{r^i}(p). \tag{12}$$

The cost of achieving a set of parameters cannot be lower than the cost of achieving each of the parameters in the set. Thus, $g^{max}_{r^i}(A)$ is the lower bound of the optimal cost of achieving r^o from r^i. In the experimental section, we will use $g^{max}_{r^i}$ as a baseline assessment to compare different methods.

6 Experimental Validation

In this section, we compare the performance of WSPR and other three AI planners, Blackbox 4.2, IPP 4.1, and Graphplan in terms of effectiveness and computational efficiency. The choice of the three AI planners is based on their reputation in the AI community. Also, they all provide their publicly accessible versions downloadable through the Web. Besides effectiveness and efficiency, we also investigate the scalability of WSPR with respect to increasing the test set size. We also study the robustness of WSPR in the presence of diverse test sets and composition scenarios.

To validate an algorithm for the WSC problem, one needs both test sets and test requests. We prepared three types of test sets as follows:

1. EEE05 test set [4]: human-generated test sets that are small-scale with only 100 Web services but with nontrivial test requests.
2. ICEBE05 test sets [5]: synthetically generated large-scale test sets.

TABLE 8
Results of the EEE05 Test Set

Test ID	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$
1	**4**	0.04	**4**	**0.01**	**4**	0.11	**4**	**0.01**
2	**4**	0.04	**4**	**0.01**	**4**	0.12	**4**	**0.01**
3	**6**	0.07	**6**	**0.01**	**6**	0.21	**6**	**0.01**
5	**1**	**0.01**	**1**	**0.01**	**1**	0.06	**1**	**0.01**
8	**2**	**0.01**	**2**	**0.01**	**2**	0.08	**2**	**0.01**
10	**1**	**0.01**	**1**	**0.01**	**1**	0.06	**1**	**0.01**
13	**2**	**0.01**	**2**	**0.01**	**2**	0.08	**2**	**0.01**
14	**2**	**0.01**	**2**	**0.01**	**2**	0.08	**2**	**0.01**
15	**1**	**0.01**	**1**	**0.01**	**1**	0.07	**1**	**0.01**

3. WSBen-generated test sets: $baTS$, $nwsTS$, and $erTS$ in Section 4.

Note that we could have generated test sets using real 1,544 Web services that we had gathered in Section 3. However, 1,544 Web services were not large enough for us and the lack of correlation among real Web services (e.g., the network diameter of G_{op} is just five) makes it hard to generate "challenging" test sets. Throughout the experiments, we use two evaluation metrics as follows:

1. $\mathcal{T}(Time)$: 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.

The smaller both values get, the better the solutions are. In other words, algorithms that take less running time while producing right solutions that use fewer Web services are considered to be "good."

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 which Blackbox forms a satisfiability (SAT) problem. For the SAT problem, Blackbox applies the local-search SAT solver, Walksat [27], so that Blackbox can run even with a large number of operators. All three AI planners are each an optimal parallel planner that minimizes the number of time steps but not necessarily the number of actions (i.e., the number of Web services). 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 $\mathcal{T}$. 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. 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 2.4 GHz with 8-Gbyte RAM. We also compare WSPR and "WSPR without heuristic" in order to assess the performance difference brought by the WSPR heuristic.

6.1 EEE05 and ICEBE05 Test Sets

The first set of experiments deals with the EEE05 contest set. In this experiment, all methods solved the optimal solutions for all test queries within 0.1 second, as shown in Table 8, where boldfaced entries mean the win of the test request. The EEE05 test set is a synthetic one that contains artificially created composition scenarios, and the test set and test requests appear to be manually created by human experts. For example, test request 15 has an input set {"pickup LocationName," "pickupLocationID," "firstName," "lastName," "middleInitial," "custStreetAddress," "custCityAddress," "custStateAddress," "custZipAddress"} and an output set {"shipmentTrackingNumber," "shipmentCost"}. The test set contains just 100 WSDL files. Although the test set is small, the EEE05 test set is still challenging because it is not simple for humans to solve them optimally in a short time. Note that the EEE05 contest originally offered 15 test requests, but six test requests (i.e., 4, 6, 7, 9, 11, and 12) out of 15 are discarded since there are syntax errors in the requests.

TABLE 9
Results of the Composition2-100-32 Test Set of ICEBE05

Test ID	Blackbox		IPP		WSPR	
	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$
1	**6**	**4.36**	**6**	11.66	**6**	4.92
2	**6**	**4.35**	**6**	11.55	**6**	4.86
3	**6**	5.81	**6**	12.73	**6**	**4.92**
4	**7**	**5.8**	**7**	12.76	**7**	6.06
5	**7**	**5.8**	**7**	12.75	**7**	6.07
6	**7**	**5.86**	**7**	12.83	**7**	6.02
7	**8**	**7.25**	**8**	13.81	**8**	7.31
8	**8**	7.34	**8**	13.85	**8**	**7.24**
9	**8**	**6.97**	**8**	13.93	**8**	7.18
10	**8**	**7.04**	**8**	13.96	**8**	7.26
11	**1**	3.42	**1**	11.86	**1**	**0.68**

The second set of experiments deals with the ICEBE05 contest set. ICEBE05 provides 18 test sets with their complexities varied in different dimensions. Note that all test requests of ICEBE05 can be solved by the full matching operation, so that $\#W = \max_{p \in r^o} g_{r^i}(p)$. We found that "Composition2-100-32" was the most difficult test set among 18 test sets and the results of the experiments with "Composition2-100-32" are shown in Table 9. More details about the ICEBE05 test sets and the computational complexity of their problems are revealed in [7]. All methods, except Graphplan, had no problem in solving the optimal solutions for all test requests within a reasonable time. We found that Graphplan fails to read the operation file when the number of operation in the file exceeds 5,000; "Composition2-100-32" has 8,356 Web services (or operations, in the PDDL and STRIPS operation files). Regarding $\#W$, all methods had no problem to generate the minimal length solution to each of the test requests. In terms of $\mathcal{T}$, Blackbox and WSPR shows similar results, while IPP shows the worst performance.

6.2 Test Sets Generated by WSBen

The third set of experiments deals with $baTS$, $nwsTS$, and $erTS$. Table 10 shows the results of the five test requests of $baTS$ with $|W| = 5{,}000$. Graphplan and IPP run out of memory for all cases. Blackbox also fails except for the fourth request, but the solution length of the fourth request is longer than that of WSPR. WSPR finds all solutions without difficulty. Regarding $\mathcal{T}$, WSPR solves all requests within 30 seconds, but Blackbox takes 609 seconds to solve the fourth request. The experiment results of $baTS$ with

TABLE 10
Results of $baTS$ with $|W| = 5{,}000$

test request	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$
r_1	-	-	-	-	-	-	**59**	**22.3**
r_2	-	-	-	-	-	-	**50**	**24.9**
r_3	-	-	-	-	-	-	**68**	**20.1**
r_4	69	609.3	-	-	-	-	**45**	**19.5**
r_5	-	-	-	-	-	-	**83**	**29.1**

TABLE 12
Results of $erTS$ with $|W| = 5{,}000$

test request	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$
r_1	43	**7.8**	-	-	43	44.2	**19**	13.7
r_2	24	96.1	-	-	24	47.4	**13**	**9.1**
r_3	42	95.6	-	-	42	48.4	**15**	**8.2**
r_4	48	80.5	-	-	48	47.7	**22**	**8.9**
r_5	17	**8.5**	-	-	-	-	**12**	8.6

$|W| = 5{,}000$ implies that the comparison of AI planners and WSPR is in vain once the number of Web services exceeds 5,000. Judging from the results above, the WSPR heuristic with the strategy in support of locating a fully matching Web service first in a tie situation is in effect when the underlying network topology follows the scale-free network.

Table 11 shows the results of the five test requests of $nwsTS$ with $|W| = 5{,}000$. Graphplan runs out of memory for all cases. IPP also fails to solve the second request. WSPR finds all solutions, and four out of the five solutions are better than the other planners. Regarding $\mathcal{T}$, WSPR shows better performance as well. Compared to $baTS$, Blackbox and IPP still run even though $\#W$ and $\mathcal{T}$ are not as good as WSPR. According to the results above, the WSPR heuristic is in effect when the underlying network topology follows a small-world network.

Table 12 shows the results of the five test requests of $erTS$ with $|W| = 5{,}000$. Graphplan fails for all cases and IPP fails to solve the fifth request. WSPR and Blackbox have no problem in obtaining solutions, but WSPR shows better performance in terms of $\#W$ and $\mathcal{T}$ than others overall. The results suggest that the WSPR heuristic is still in effect even when the underlying network topology follows the random network.

From the above experiments using diverse test sets, we can understand how different network models of G_{cl} influence 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, $baTS$ based on the Barabasi-Albert model needs more partial matching Web services to fulfill the given requests than others. As such, the increasing need for partial matching Web services leads to increasing $\#W$ and $\mathcal{T}$. This is the reason three AI planners almost failed to run in the $baTS$ case.

TABLE 11
Results of $nwsTS$ with $|W| = 5{,}000$

test request	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$	$\#W$	$\mathcal{T}$
r_1	47	61.9	-	-	47	22.0	**30**	**7.8**
r_2	**29**	58.4	-	-	-	-	40	**8.7**
r_3	39	30.9	-	-	39	21.4	**28**	**7.8**
r_4	38	7.1	-	-	38	21.6	**26**	**7.0**
r_5	39	3.5	-	-	39	20.3	**26**	**6.5**

We change our focus from the comparison of WSPR and AI planners to the assessment of scalability of WSPR. Figs. 12 and 13 show how WSPR and WSPR without heuristic operate in different domains, while the test size increases from 1,000 to 50,000. Note that $\#W$ and $\mathcal{T}$ at each size are averaged over the solutions to the five requests. The line with the triangle mark at the bottom in each graph in Fig. 12 links values, where each value is averaged over the five lower bounds of optimal solutions at each size. In fact, the lower bound corresponds to the total time steps counted from the input to the desired output parameters in the forward search of WSPR so that $\#W$ cannot be less than the lower bound.

Regarding $\#W$, WSPR and WSPR without heuristic have no difficulties to address requests even when $|W| = 50{,}000$. However, WSPR overruns WSPR without heuristic completely. Regarding $\mathcal{T}$, WSPR and WSPR without heuristic scale up smoothly, with WSPR being slightly faster than WSPR without heuristic. We can simply assert that WSPR will continue to perform with this increasing smooth pattern in terms of $\mathcal{T}$, even if $|W|$ continues to increase. Thus, the time to compute the heuristic in the regression search step is trivial and WSPR is scalable, regardless of different topologies of underlying Web service networks.

Fig. 14 illustrates the solutions of WSPR and Blackbox to the fourth request of $nwsTS$ with $|W| = 3{,}000$, respectively. Note that WSPR and Blackbox have the same scheme in their forward search stage because both aim to minimize the number of time steps to reach the goal. Therefore, both WSPR and Blackbox have the same number of time steps of eight. However, in their regression search schemes, considerable differences exist. When the size of a test set becomes very large, Blackbox converts the plan graph into a set of clauses to form a SAT problem. For the SAT problem, Blackbox applies Walksat, an incomplete local SAT search algorithm. The algorithm is limited to seek an assignment of the variables that satisfies a given formula without considering full or partial matching. It is natural because the SAT problem is a decision problem, not an optimization problem. As a result, Walksat search strategy can produce poor solutions in the WSC problem domain.

For example, at the sixth time step in Fig. 14, Blackbox composes a set of four Web services, such as {`ws38`, `ws366`, `ws369`, `ws710`}, to match the subgoal at the seventh time step that is the set of input parameters of two Web services, such as {`ws371`, `ws1364`}. This decision results in an exponentially increasing number of Web services for the subsequent subgoal. Due to the exponentially growing number, as previous results show, the regression search of Blackbox often fails before it reaches the initial state. On the contrary, at

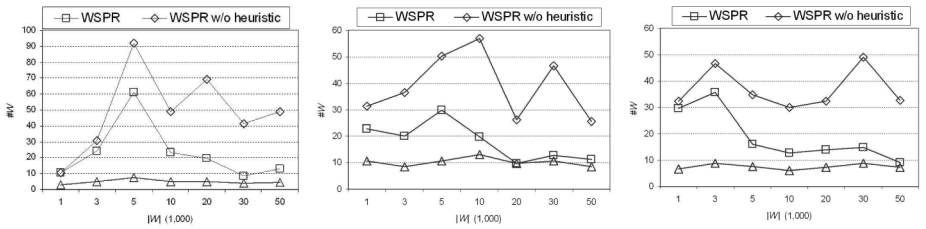


Fig. 12. $\#W$ of WSPR over the test sets. (a) $baTS$. (b) $nwsTS$. (c) $erTS$.

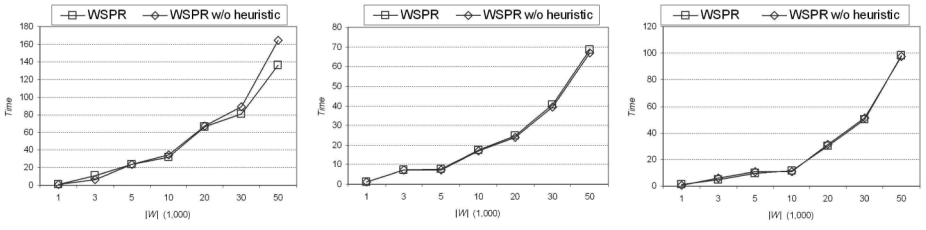


Fig. 13. $\mathcal{T}$ of WSPR over the $nwsTS$ test sets. (a) $baTS$. (b) $nwsTS$. (c) $erTS$.

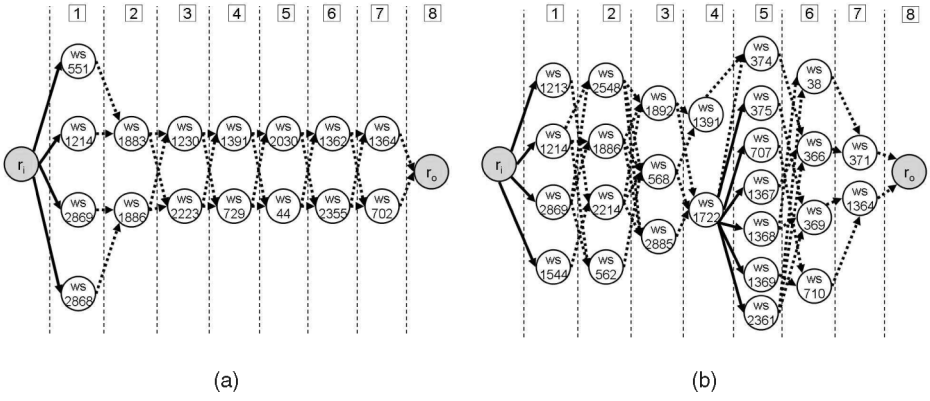


Fig. 14. Solutions to $nwsTS$ with $|W| = 3{,}000$. (a) WSPR. (b) Blackbox.

the same time step, WSPR, which is designed to favor the full matching Web services, composes a set of two Web services, such as {ws1362, ws2355}, to match the subgoal that is the set of input parameters of two Web services, such as {ws1364, ws702}. Thus, the size of subsequent subgoal does not explode and continues this pattern until it reaches the initial state, thereby reducing $\#W$.

The main findings about WSPR from the experiments are given as follows:

1. *Effectiveness.* As confirmed by the experimental evaluation, WSPR shows better results in 80 percent of all cases in terms of $\#W$, compared to the other prominent AI planners including Blackbox, IPP, and Graphplan. In particular, WSPR outperformed WSPR without heuristic. This implies that the heuristic of WSPR with the strategy in support of locating fully matching Web service first in a tie situation is in effect.
2. *Computational efficiency.* We measured how quickly WSPR generates the correct solutions in comparison with the other AI planners, as more data are applied to the problem. The experimental validation showed that WSPR outperformed the other methods with significant differences in terms of $\mathcal{T}$.
3. *Scalability.* The experiments proved that WSPR and WSPR without heuristic did not blow up exponentially when the test set size increases. This implies that both algorithms have scalability and the two-step approach of WSPR is responsible for scalability. Note that the time to compute the WSPR heuristic in the regression search step was ignorable.
4. *Robustness.* As confirmed by the experimental evaluation, WSPR can be seen as a robust solution for the WSC problem because it can perform persistently well in diverse WSC scenarios that arise in different test sets, including EEE05, ICEBE05 characterized by the tree structure, and WSBen-generated test sets: $baTS$, $nwsTS$, and $erTS$ featured by complex and random graph structures.

7 Related Research Work

There are three approaches from different communities that have been established to address WSC (Table 13). First is the manual composition approach, which adopts the idea of

TABLE 13
WSC Approaches and Models

Approaches	Models
Semi-automatic	METEOR-S, Proteus, Kepler, e-Flow, etc.
AI planning	Graphplan, SATPlan, Blackbox, etc.
Semantic composition	DAML-S, SHOP2, OntoMat-Service, etc.

semiautomatic service composition where GUI-based software and human experts can work together to generate composite services by allowing for binding manually generated workflows to the corresponding concrete Web services either statically or dynamically. To that end, METEOR-S [28], Proteus [29], Kepler [30], and e-Flow [31] were suggested.

The second approach is to leverage on the various planning-based solutions of AI community. Regarding representative AI planners capable of being used by WSC solvers, there are Graphplan [25], the SATPlan algorithm [32], and Blackbox. We conducted a further detailed survey on these algorithms and analyzed their distinct solving processes with respect to WSC context [33].

The third approach is to utilize Semantic Web features for the service matching and composability. Medjahed et al. [3] propose an ontology-based framework for the automatic composition of Web services. Sirin et al. [14] demonstrate how an OWL reasoner can be integrated with an AI planner to overcome automatic WSC problems, which work later extends to the development of SHOP2. OntoMat-Service, the semantic framework proposed by Zeng et al. [34], provides a Web service planning module, which computes logically possible service flows based on a knowledge base with a set of pre- and postconditions, goals, and so forth, for Web services.

Although the above three approaches are designed with different directions in expectation of future Web services, we however believe that the three approaches are complementary enough to be incorporated to generate better solutions. As an example, we can assume that Web services are published according to Semantic Web standards and recommendations, and they are composed by means of AI planning algorithms initially but automatically for a large scale, and then the initial solutions can be fine-tuned by human experts using GUI-based software.

8 Conclusion

So far, we have explored several novel issues by laying out the developmental organization of this paper. First, we formulated the WSC problem in terms of AI planning and network optimization problems to investigate its complexity. Second, we analyzed publicly available Web service sets using complex network analysis techniques. Third, we proposed WSBen, a novel Web service benchmark tool, which may be reused by follow-up research studies. Fourth, we developed WSPR, a novel AI planning-based heuristic WSC algorithm and verified its performance against state-of-the-art AI planners. WSBen (downloadable at http://pike.psu.edu/sw/wsben/) and WSPR (downloadable at http://pike.psu.edu/sw/wspr/) in GUI versions are as shown in Fig. 15. Although it is our hope that WSPR and

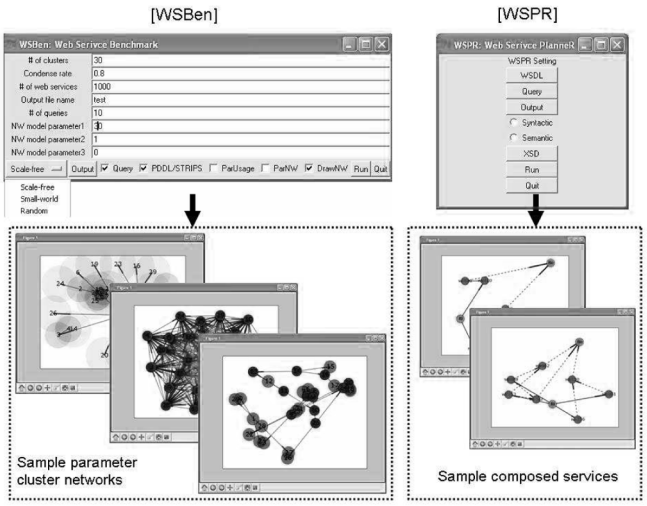


Fig. 15. GUI versions of WSBen and WSPR.

WSBen will provide useful insights for follow-on researchers, it is true that the two proposals still have limitations to address. For example, WSBen is only and most applicable when users know the underlying network topology of the targeted Web service cluster. In addition, WSBen is currently limited to generate only WSDL files so that WSBen cannot incorporate semantic information that is written in OWL. Similar to WSBen, WSPR is hardly used in conjunction with Semantic Web because it is designed with an assumption that the matchmaking is sufficiently resolved by WSDL interface information. It is, however, possible that this assumption is invalidated in the near future as Semantic Web becomes populated. If such a case happens, WSPR needs to be revisited by adding reasoning modules to set up or interpret semantic composition conditions. For these reasons, we have consideration on future research studies to address limitations and relax assumptions of current work, such as: 1) Mixed Integer Programming-based multicriteria framework for WSC in order to relax assumptions like single criterion for determining effectiveness, 2) Open-loop control-based composition to handle the dynamic Web environment issues, and 3) Semantic matchmaking by incorporating the agent reasoning technology. Finally, we hope that future follow-on researchers benefit not only by our findings but also the list of future researches we have initiated.

Appendix A
Proof of Theorem 2.6

Proof. We revised the result of Bylander [9] in Web service context. In this proof, the WSC problem is considered as a decision problem of determining whether an instance of propositional STRIPS planning has a solution of k or fewer operators, where k is given as part of input. The first part of this proof proves that the WSC problem is NP, and the second part proves that the WSC problem is NP-hard by building a polynomial reduction from the 3SAT [13].

NP. Web services without negative effects can never negate a condition. Thus, a previous state is always a subset of succeeding states. Also, Web services within a

service sequence that have no effect can always be removed. Hence, if a solution exists, the length of the smallest solution can be no longer than the total number of Web services. Since only a linear number of nondeterministic choices are required, the WSC problem is in NP. In other words, a solution to a WSC problem can be verified in polynomial time.

NP-hard. 3SAT can be reduced to the WSC problem in polynomial time (3SAT $\leq_p$ WSC). Let $U = \{u_1, u_2, \ldots, u_m\}$ be the set of variables used in an instance of 3SAT. Let n be the number of clauses. The equivalent WSC problem to the instance of 3SAT can be constructed using the following types of parameters or conditions:

- $T(k)$: if $u_k = \text{true}$ is selected, $T(k)$ is true; otherwise false.
- $F(k)$: if $u_k = \text{false}$ is selected, $F(k)$ is true; otherwise false.
- $V(k)$: if some value (i.e., either true or false) for u_k has been selected, then $V(k)$ is true; otherwise false.
- $C(j)$: if the jth clause is satisfied, it is true; otherwise false.

The initial state and goal can be specified as follows:

- $r^i = \emptyset$,
- $r^o = V(1) \wedge V(2) \wedge \cdots \wedge V(m) \wedge C(1) \wedge C(2) \wedge \cdots \wedge C(n)$.

That is, the goal is to select a value for each variable (i.e., V) and satisfy each of the clauses. For each variable u_k, four Web services are needed:

- $wt_k : wt^i_k = \emptyset$ and $wt^o_k = T(k)$.
- $wf_k : wf^i_k = \emptyset$ and $wf^o_k = F(k)$.
- $wv_{k1} : wv^i_{k1} = T(k)$ and $wv^o_{k1} = V(k)$.
- $wv_{k2} : wv^i_{k2} = F(k)$ and $wv^o_{k2} = V(k)$.

These four Web services are used to select a value for u_k and to ensure that a value has been assigned to u_k. Note that nothing prevents the double selection of both true and false for one variable. To prevent the double assignment, we require two more Web services, wc_{j1} and wc_{j2} described below. If the jth clause contains a variable u_k, wc_{j1} is needed. On the contrary, if the jth clause contains a negated variable $\bar{u_k}$, wc_{j2} is needed:

- wc_{j1}: $wc^i_{j1} = T(k)$ and $wc^o_{j1} = C(j)$, where u_k is contained in the jth clause.
- wc_{j2}: $wc^i_{j2} = F(k)$ and $wc^o_{j2} = C(j)$, where $\bar{u_k}$ is contained in the jth clause.

If the 3SAT problem is satisfiable, then r^o is true because all $V(k)$ and $C(j)$ must be true. On the other hand, if the 3SAT problem is not satisfiable, then at least one of $C(j)$ must be false, resulting that r^o is false.

In addition, if the 3SAT formula is satisfiable, then $2m + n$ Web services are sufficient, because

- Only one value between true and false is selected for each variable. Therefore, m Web services are required.
- m number of Web services are required to ensure that m number of variables are set by a true or false value.
- n number of Web services are required to indicate that n number of clauses are determined to be either true or false.

On the contrary, if the 3SAT formula is not satisfiable, then both values must be selected for some variables to achieve the goal, implying that more than $2m + n$ Web services are needed. Thus, the 3SAT formula is satisfiable if and only if there is a WSC of size $k = 2m + n$. □

APPENDIX B
PROOF OF THEOREM 5.1

Proof. We use the following loop invariant: At the start of each iteration of the "while" loop of lines 2-7 in Algorithm 1, we obtain $g_{r^i}(p)$ for each parameter $p \in w^o$, where $w \in \delta$.

Then, it suffices to show that for each parameter $p \in w^o$, we obtain $g_{r^i}(p)$ at the time when p is added to state s:

- **Initialization:** Initially, $s = \emptyset$, so that the invariant is trivially true.
- **Maintenance:** For the purpose of contradiction, let u be the first parameter for which we do not obtain $g_{r^i}(u)$ when u is added to state s. This assumption implies that there is another state including u, which occurs afterward providing $g_{r^i}(u)$. However, this is false because $c(w) = 1$ for all $w \in W$. In other words, if $s_{i+1} = f(w_i, s_i)$, $g_{r^i}(y) = g_{r^i}(x) + 1$, where $x \in (s_i \setminus s_{i-1})$ and $y \in (s_{i+1} \setminus s_i)$. Therefore, we obtain $g_{r^i}(p)$ at the time when p is first added to state s.
- **Termination:** The termination condition is $s \supseteq r^o$. This implies that $s \in S_G$. Thus, we have obtained $g_{r^i}(p)$ for all parameters $p \in S_G$ if at least a path exists from r^i to r^o. □

APPENDIX C
PROOF OF THEOREM 5.2

Proof. We use the following loop invariant: at the start of each iteration of the "while" loop of lines 2-6 in Algorithm 2, we obtain a set of Web services, $soln$, that can be temporarily sequenced and executed to form the path from $subGoal$ to r^o. Then, it suffices to show that for the Web service w that was added recently in $soln$, w can be invoked by $subGoal$. Once we show that $subGoal$ can invoke w, we rely on the $subGoal$ relation described at line 6 in Algorithm 2 to show that the subsequent invocation holds at all times thereafter:

- **Initialization:** Initially, $subGoal = r^o$ and $soln = \emptyset$, making the invariant trivially true.
- **Maintenance:** Let A be a Web service added to $soln$ at time t, where $t = 1, 2, \ldots, T(\geq \max_{p \in r^o} g_{r^i}(p) - 1)$. As soon as A is added to $soln$, $subGoal(t)$ is updated such that $subGoal(t) = [subGoal(t+1) \setminus (A^o \cup r^i)] \cup A^i$. That is, $subGoal(t)$ contains A^i, which is a set of preconditions necessary to invoke A. Therefore, the updated $subGoal(t)$ can

always invoke A. After invoking A, we can obtain new information state $s = subGoal(t) \cup (A^o \cup r^i)$. Note that r^i is always available throughout the planning process. Since $s \supseteq subGoal(t+1)$, there is at least one Web service whose input parameters are included in s as long as $soln \neq \emptyset$. In this manner, we can invoke all Web service in $soln$ until $soln = \emptyset$.

- **Termination:** The termination condition is $subGoal \setminus r^i = \emptyset$. This implies that r^i satisfies $subGoal$. In the forward search of WSPR, we saw that there is a state s, such that $s \supseteq r^o$. This suggests that there is at least one path from r^i to r^o. Since there exists a path from r^i to r^o, there is at least one Web service w belonging to $soln$ at termination, such that w can be invoked only by r^i.

□

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.,* vol. 284, no. 5, pp. 34-43, 2001.

[2] M. Cantera, "IT Professional Services Forecast and Trends for Web Services," Technical Report ITES-WW-MT-0116, Gartner, 2004.

[3] B. Medjahed, A. Bouguettaya, and A. Elmagarmid, "Composing Web Services on the Semantic Web," *The VLDB J.,* vol. 12, pp. 333-351, 2003.

[4] EEE05, http://www.comp.hkbu.edu.hk/eee05/contest/, Mar. 2005.

[5] ICEBE05, http://www.comp.hkbu.edu.hk/ctr/wschallenge/, May 2006.

[6] S.-C. Oh, H. Kil, D. Lee, and S. Kumara, "WSBen: A Web Services Discovery and Composition Benchmark," *Proc. Fourth Int'l IEEE Conf. Web Service (ICWS '06),* pp. 239-246, 2006.

[7] S.-C. Oh, D. Lee, and S. Kumara, "WSPR: A Heuristic Algorithm for Web Service Composition," *Int'l J. Web Services Research (IJWSR '07),* vol. 4, no. 1, 2007.

[8] N.J. Nilsson, *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann, 2001.

[9] T. Bylander, "The Computational Complexity of Propositional Strips Planning," *Artificial Intelligence,* vol. 69, nos. 1-2, pp. 165-204, 1994.

[10] B. Bonet and H. Geffner, "Planning as Heuristic Search," *Artificial Intelligence,* vol. 129, nos. 1-2, pp. 5-33, 2001.

[11] P. Haslum and H. Geffner, "Admissible Heuristics for Optimal Planning," *Proc. Fifth Int'l Conf. Artificial Intelligence Planning and Scheduling Systems (AIPS '00),* pp. 140-149, 2000.

[12] C. Kwok and D. Weld, "Planning to Gather Information," *Proc. 13th Nat'l Conf. Artificial Intelligence (AAAI '96),* pp. 32-39, 1996.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 2002.

[14] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN Planning for Web Service Composition Using SHOP2," *J. Web Semantics,* vol. 1, no. 4, pp. 377-396, 2004.

[15] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2001.

[16] J. Fan and S. Kambhampati, "A Snapshot of Public Web Services," *SIGMOD Record,* vol. 34, no. 1, pp. 24-32, 2005.

[17] R. Albert and A. Barabasi, "Statistical Mechanics of Complex Networks," *Rev. Modern Physics,* vol. 74, no. 1, pp. 47-95, 2002.

[18] P. Erdos, R. Graham, and J. Nesetril, *The Mathematics of Paul Erdos.* Springer-Verlag, 1996.

[19] D. Watts and S. Strogatz, "Collective Dynamics of 'Small-World' Networks," *Nature,* vol. 393, no. 4, pp. 440-442, 1998.

[20] J. Delgado, "Emergence of Social Conventions in Complex Networks," *Artificial Intelligence,* vol. 141, pp. 171-185, 2002.

[21] D. Watts, *The Dynamics of Networks between Order and Randomness.* Princeton Univ. Press, 1999.

[22] R. Albert and A.-L. Barabasi, "Topology of Evolving Networks," *Physical Rev. Letters,* vol. 85, pp. 5234-5237, 2000.

[23] R. Albert, H. Jeong, and A.-L. Barabasi, "The Diameter of the World Wide Web," *Nature,* vol. 401, pp. 130-131, 1999.

[24] D. McDermott, "A Heuristic Estimator for Means-Ends Analysis in Planning," *Proc. Third Int'l Conf. Artificial Intelligence Planning Systems (AIPS '96),* pp. 142-149, 1996.

[25] A. Blum and M. Furst, "Fast Planning through Planning Graph Analysis," *Artificial Intelligence,* vol. 90, pp. 281-300, 1997.

[26] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation through Heuristic Search," *J. Artificial Intelligence Research,* vol. 14, pp. 253-302, 2001.

[27] B. Selman and H. Kautz, "Domain-Independent Extension to GSAT: Solving Large Structured Satisfiability Problems," *Proc. 13th Int'l Joint Conf. Artificial Intelligence (IJCAI '93),* pp. 290-295, 1993.

[28] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding Semantics to Web Services Standards," *Proc. First IEEE Int'l Conf. Web Services (ICWS '03),* pp. 395-401, 2003.

[29] S. Ghandeharizadeh, C. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J. Ambite, M. Cai, C.-C. Chen, P. Pol, R. Schmidt, and S. Saihong, "Proteus: A System for Dynamically Composing and Intelligently Executing Web Services," *Proc. First IEEE Int'l Conf. Web Services (ICWS '03),* pp. 17-21, 2003.

[30] I. Altintas, E. Jaeger, K. Lin, B. Ludaescher, and A. Menon, "A Web Service Composition and Deployment Framework for Scientific Workflows," *Proc. Second IEEE Int'l Conf. Web Services (ICWS '04),* pp. 814-815, 2004.

[31] H. Sun, X. Wang, B. Zhou, and P. Zou, "Research and Implementation of Dynamic Web Services Composition," *Lecture Notes in Computer Science,* Sept. 2003.

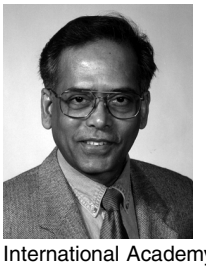
[32] H. Kautz and B. Selman, "Unifying Sat-Based and Graph-Based Planning," *Proc. 16th Int'l Joint Conf. Artificial Intelligence (IJCAI '99),* pp. 318-325, 1999.

[33] S.-C. Oh, D. Lee, and S. Kumara, "A Comparative Illustration of AI Planning-Based Web Services Composition," *ACM SIGecom Exchanges,* vol. 5, no. 5, pp. 1-10, 2005.

[34] L. Zeng, B. Benatallah, M. Kalagnama, and Q.Z. Sheng, "Quality Driven Web Services Composition," *Proc. 12th Int'l Conf. World Wide Web (WWW '03),* pp. 411-421, 2003.

Seog-Chan Oh received the BS and MS degrees from Dongguk University in 1993 and 1996, respectively, and the PhD degree from Pennsylvania State University in 2006. Before he began his PhD studies, he was an IT consultant for seven years at Daewoo Information Systems. He has been a researcher at the General Motors Research and Development Center since 2007. His research interests include Web services and Semantic Web, optimization, AI, data mining, and agent technology. He has been certified as a Professional Engineer in Information Management by the Korean government.

Dongwon Lee received the BS degree in computer science from Korea University in 1993, the MS degree in computer science from Columbia University in 1995, and the PhD degree in computer science from the University of California, Los Angeles in 2002. He is currently an associate professor in the College of Information Sciences and Technology, Pennsylvania State University. His research interests include databases, XML and Web analysis, and Web services and the Semantic Web.



Soundar R.T. Kumara is an Allen E. Pearce/Allen M. Pearce chaired professor in the Department of Industrial and Manufacturing Engineering, Pennsylvania State University. His research interests are in intelligent systems with an emphasis on sensor-based equipment monitoring and diagnosis, software agents, and complexity in the supply chain. He has more than 150 publications. He has won several awards. He is an elected member of the International Academy of Production Research.