# Web Service Planner (WSPR): An Effective and Scalable Web Service Composition Algorithm

*Seog-Chan Oh, The Pennsylvania State University, USA*

*Dongwon Lee, The Pennsylvania State University, USA*

*Soundar R. T. Kumara, The Pennsylvania State University, USA*

## ABSTRACT

*As the emergence of service-oriented architecture provides a major boost for e-commerce agility, the number of available Web services is rapidly increasing. However, when there are a large number of Web services available and no single Web service satisfies the given request, one has to "compose" multiple Web services to fulfill the goal. In this article, toward this problem, we present an AI planning-based Web service composition algorithm named as WSPR. We evaluate the efficiency and effectiveness of WSPR using two publicly available test sets—EEE05 and ICEBE05. In addition, we analyze the two test sets and suggest several improvements to benchmark Web service composition better.*

*Keywords: AI planning; parameter usage distribution; Web services composition*

## INTRODUCTION

Web Services are often considered one of the most important and vital building blocks for the semantic Web (Berners-Lee, 2001). 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% of all types of software will be Web services enabled (Cantera, 2004). In Web services enabled networks, typically, a client program first locates a Web services 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 receives a response from the service. Unlike conventional programming interface, Web services are self-explanatory so that by interpreting XML tags, applications can interpret the semantics of operations. In particular, the problem of practical interest concerns the following two issues. Given a request *r*, among thousands of candidate Web services found in UDDI: (1) How to find matching services that satisfy *r*; (2) How to compose multiple services

to satisfy $r$ when a matching service does not exist. We motivate our work through the following example.

## Motivating Example

Consider the four Web services in Table 1 illustrates in WSDL notation:

- Given the hotel, 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.
- 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 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":

(1) $r_1$: find the address of the hotel *Atherton*, and
(2) $r_2$: find a *Thai* restaurant near the hotel *Atherton* along with driving directions.

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 Ave" 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 driving directions. On the other hand, the Web service `findDirection` can give

*Table 1. Example web services*

```
<message name="findHotel_Request">
      <part name="hotel" type="xs:string">
      <part name="city" type="xs:string">
      <part name="state" type="xs:string">
</message>
<message name="findHotel_Response">
      <part name="address" type="xs:string">
      <part name="zip" type="xs:string">
</message>
```

(a) `findHotel`

```
<message name="findRestaurant_Request">
      <part name="zip" type="xs:string">
      <part name="foodPref" type="xs:string">
</message>
<message name="findRestaurant_Response">
      <part name="name" type="xs:string">
      <part name="phone" type="xs:string">
      <part name="address" type="xs:string">
</message>
```

(b) `findRestaurant`

```
<message name="guideRestaurant_Request">
      <part name="foodPref" type="xs:string">
      <part name="currAddress" type="xs:
string">
</message>
<message name="guideRestaurant_Re-
sponse">
      <part name="rating" type="xs:string">
      <part name="destAddress" type="xs:
string">
</message>
```

(c) `guideRestaurant`

```
<message name="findDirection_Request">
      <part name="fromAddress" type="xs:
string">
      <part name="toAddress" type="xs:string">
</message>
<message name="findDirection_Response">
      <part name="map" type="xs:string">
      <part name="direction" type="xs:string">
</message>
```

(d) `findDirection`

driving directions from one location to another, but cannot locate any restaurant. Therefore, one has to use a chain of Web services to fully satisfy $r_2$. Two possible ways exist. After obtaining the hotel address using `findHotel`, one can do either:

- First, invoke `guideRestaurant` ("Thai," "100 Atherton Ave, 16801, PA") to get the address of the closest restaurant, say "410 S. Allen St. 16802, PA." Then, invoke the Web service `findDirection`("100 Atherton Ave, 16801, PA," "410 S. Allen St. 16802, PA") to get the driving directions.
- Second, invoke `findRestaurant` ("16801," "Thai") to get the address of the closest restaurant, say "410 S. Allen St. 16802, PA." Then, invoke the Web service `findDirection`("100 Atherton Ave, 16801, PA," "410 S. Allen St. 16802, PA") to get the driving directions.

A Web service, $w$, has typically two sets of parameters: $w^i=\{I_1,I_2,...\}$ for SOAP request (as input) and $w^o=\{O_1,O_2,...\}$ 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). Given a request $r$ with input parameters $r^i$ and output parameters $r^o$, finding a set of Web services $w \in W$ such that $r^i \supseteq w^i$ and $r^o \subseteq w^o$ is referred to as the *Web Service Discovery (WSD)* problem. With a simple look-up table, the WSD problem can be trivially solved. Therefore, in the remainder of the article, 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 the problem is referred to as the *Web Services Composition (WSC)* problem, and is elaborated in subsequent section.

## FORMALIZING THE WSC PROBLEM

The problem of interest in the AI planning community primarily concerns scenarios which allow interleaving of actions from different sub-plans within single sequence. By contrast, the WSC problem can be considered as the *information gathering* problem (Kwok & Weld, 1996) where Web services represent information sources, and interleaving between Web services are not found; this enables a highly specialized planning algorithm. The only preconditions to Web services are knowledge preconditions. Furthermore, there are no sibling sub-goal interactions such as those characterizing the Sussman anomaly. For that reason, a Web service composition algorithm does not model the world state as do many other complete AI planners; instead it models the world state as *information state*, a description of the information collected by the algorithm at a particular stage in composition (Russel & Norvig, 2002).

We first cast the WSC problem as a planning problem in STRIPS model of the 4-tuple: $\Pi=<P,W,r^i,r^o>$ where:

(1) $P$ is a set of parameters. In the motivating example, $P$={"hotel_name," "hotel_city," "hotel_address"},
(2) $W$ is a set of Web services. In the motivating example, $W$={`findHotel`, `findRestaurant`, ...},
(3) $r^i \subseteq P$ is the initial input parameters, and
(4) $r^o \subseteq P$ is the desired output parameters.

Note that $\Pi$ 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 pre-conditions and post-conditions of an operator are the ground literals and the goals are also ground literals (Bylander, 1994). Figure 1. illustrates the STRIPS model of the motivation example. A STRIPS model $\Pi$ defines a state space $\Psi=<S,s_0,S_G,\Omega(.),f,c>$ where:

(1) The states $s \in S$ are collection of parameters in $P$,
(2) The initial state $s_0 \in S$ is such that $s_0=r^i$,

(3) The goal state $S_G \in S$ is such that $r^o \subseteq S_G$,
(4) $\Omega(s)$ is the 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 another state $s'$ such that $s'=s \cup w^o$ for $w \in \Omega(s)$, and
(6) $c(w)$ is the invocation cost of $w$.

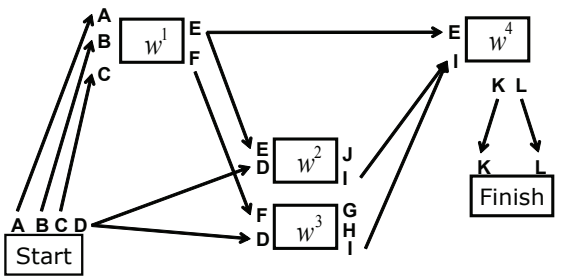
A solution of the state model is a finite sequence of Web services $w_1,w_2,...,w_n$ such that for a sequence of states $s_1,s_2,...,s_{n+1}$ where $s_{i+1}=f(w_i, s_i)$ for $i=1,...n$, $w_i \in \Omega(s_i)$, $s_0 \in s_1$, and $S_G \in s_{n+1}$. Based on $\Psi=<S,s_0,S_G,\Omega(.),f,c>$, the Web service composition problem can formally be defined.

**Definition 1 (*Web Service Composition*)** *Suppose that a request r has initial input parameters $r^i$ and desired output parameters $r^o$. Web Service Composition (WSC) problem is to find a finite sequence of Web services, $w_1,w_2,...,w_n$ such that (1) $w_i$ can be invoked sequentially from 1 to n, (2) $(r^i \cup w^o_1 \cup ... \cup w^o_n) \supseteq r^o$, and (3) the total cost $\sum_{i=1}^{n} c(w_i)$ is minimized.*

**Definition 2 (*State node network*)** *A state node network refers to a directed graph $G_s(V_s,E_s)$ where $i \in V_s$ represent the state $s_i \in S$ and $E_p$ is a set of directed edges $(i,j)$ that connects ordered pairs of $i \in V_s$ and $j \in V_s$. Every arc $(i,j) \in E_s$ is weighted by the invocation cost $c(w)$ where i and j represent $s_i$ and $s_j$ respectively such that $w \in \Omega(s_i)$ and $s_j=(w, s_i)$ (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 in the forward direction only. The cost of a path is the sum of the costs of the associated arcs. Our interest is to obtain a shortest path between $s_0$ and $s_n \supseteq S_G$. Figure 2 illustrates the $G_s(V_s,E_s)$ of the motivating example. It is evident that if $c(w_i)=1$ for $\forall w_i \in W$, then the shortest paths are: (1) $s_0$, $s_1$, $s_2$, $s_5$ and (2) $s_0$, $s_1$, $s_3$, $s_7$. Indeed, it can be formulated as the minimum cost flow problem[1]. Moreover, there are polynomial algorithms to solve the minimum cost flow problems such as label-setting algorithms (when $c(w) \geq 0$) or label-correcting algorithms[2]. However, there are problems with this idea. These algorithms are polynomial in the size of nodes $V_s$ but the

*Figure 1. STRIPS model of the motivation example*




***P:* parameter set**

| | | | |
|---|---|---|---|
| hotelName | A | restaurantName | G |
| hotelCity | B | restaurantPhonenumber | H |
| hotelState | C | restuarantAddress | I |
| foodPreference | D | restaurantRate | J |
| hotelAddress | E | mapHotelRestaurant | K |
| hotelZipcode | F | directionHotelRestaurant | L |

***W:* web services set**

| | |
|---|---|
| findHotel | $w^1$ |
| findRestaurant | $w^2$ |
| guideRestaurant | $w^3$ |
| findDirection | $w^4$ |

number of nodes is exponential in the number of parameters $|P|$ because nodes are states. Before we proceed to investigate the complexity of WSC problem in detail, we need to introduce two matching operations described below:

**Definition 3 (*Full matching*)** *Suppose that a state $s \in S$ is given. Let a Web service $w_i \in \Omega(s)$. If for $w_2 \in W$, $w_1^o \supseteq w_2^i$, then $w_1$ can "fully" match $w_2$.*

**Definition 4 (*Partial matching*)** *Suppose that a state $s \in S$ is given. Let a Web service $w_1 \in \Omega(s)$. If for $w_2 \in W$, $w_1^o w_2^i$ and $w_1^o$' $w_2^i$ 'Æ, then $w_1$ can "partially" match $w_2$.*

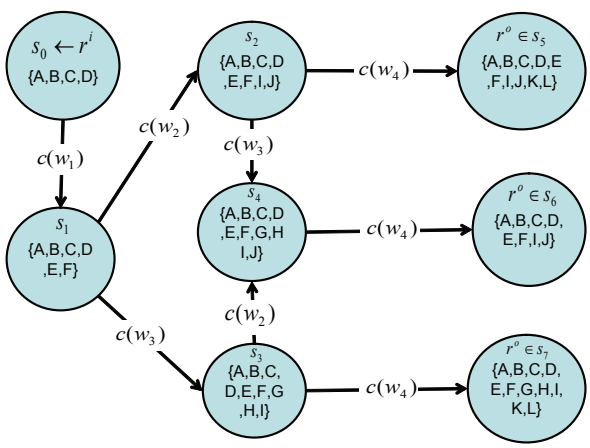
In the motivation example, `findHotel` partially matches `findRestaurant` and `guideRestaurant`. In turn, `findRestaurant` and `guideRestaurant` partially match `findDirection`. However, if both `findHotel` and `findRestaurant` is composed then `findDirection` can be fully matched.

When only full matching is considered in the WSC problem, it can be seen as a single-source shortest path problem defined over $G_s(V_s, E_s)$ because the total number of input and output parameter sets is $2\times|W|$ thereby $|V_s| = 2|W|$. Thus, its computational complexity is tractable. On the other hand, when both full- and partial- matching must be considered concurrently, the problem becomes a decision problem to determine the existence of a solution of $k$ operators or less for propositional STRIPS planning (Nilsson, 2001) with restrictions on negation in pre- and post-conditions. Its computational complexity is proved to be NP-complete (Bylander, 1994). Therefore, when the number of Web services to search is not small, finding an optimal solution to the WSC problem (i.e., the sequence of a sequence of Web services from $r^i$ to $r^o$) is prohibitively expensive, leading to approximate algorithms instead. To address this intractable WSC problem, we will suggest a polynomial-time AI-planning based heuristic algorithm in the following sections.

## CLASSIFICATION OF WSC PROBLEMS AND RELATED RESEARCH WORK

We can classify the WSC problem using the following four facets[3]:

*Figure 2. $G_s(V_s, E_s)$ of the motivation example.*



- **Manual vs. Automatic Workflow Composition:** In building workflows by means of Web services, one can do either (1) manual composition in cooperation with domain experts; or (2) automatic composition by software programs. In the manual approach, human users who know the domain well (e.g., domain ontology) select proper Web services and weave them into a cohesive workflow. Although users may rely on some GUI-based software to facilitate the composition, in essence, it is a manual and labor-intensive task, and thus is not appropriate for large-scale WSC.
- **Deterministic vs. Stochastic Environment:** If we view the WSC as a design-oriented process by ignoring the stochastic real-world environment, AI-inspired planning techniques are suitable for the WSC. However since the classical STRIPS-style planning algorithms assume deterministic behavior of Web services, additional overhead is required to monitor execution time to recover from unexpected behavior of Web services (e.g., failure). There are some research efforts to suggest approaches for dynamically composing Web services in run-time with considerations to the stochastic environment.
- **Simple vs. Complex Operator:** The simple WSC involves only two types of matching operations: (1) full-matching operator and (2) partial-matching operator. If WSC involves the two operators alone, it can be considered as a linear planning (i.e., there are no sibling-sub goal interactions such as those characterizing the Sussman anomaly[4]). More complex WSC, however, can use other operators (e.g., OR, XOR, NOT) or constraints (e.g., request prefers sources in Asia to the ones in Europe) in both sequential and parallel modes. Note that throughout our article, WSC means the simple WSC.
- **Small vs. Large Scale**: General WSC problem to find an optimal workflow can be formulated as an AI planning problem, into which the Satisfiability problem can be reduced (Vossen et al., 1999). We know that the satisfiability problem is NP-complete and therefore, it is unlikely to have a polynomial algorithm for the WSC problem. Note that we defined the WSC problem as an optimization problem. However, it is true that many solutions are currently available with capability of dealing with large scale problems and it is possible because of their polynomial-time nature to approximate optimal solutions.

In Figure 3., we present a decision tree to help select the right solution using the aforementioned four facets. The manual composition approach can rely on software programs that have functions to bind manually-generated workflows to the corresponding concrete resources. To that end, METEOR-S (Sivashanmugam et al., 2003) and Proteus (Ghandeharizadeh, 2003) were suggested. Kepler (Altintas et al., 2004), in particular, provides a scientific workflow editor which allows scientists to effectively query and compose distributed data sources on the Grid. Thus, it is possible to build scientific workflows across diverse scientific domains for analysis and modeling tasks. METEOR-S, Proteus and Kepler adopt the idea of semi-automatic service composition and indicate the trend that GUI-based software and human experts can work together to generate the composite service. However, we believe that by leveraging on various planning-based solutions of AI community, people can solve WSC problem better. If one knows that an automatic composition is plausible for the given WSC problem, then she may apply planning-based automatic composition solutions first to get the initial set of candidate workflows, and perform additional fine-tuning later.

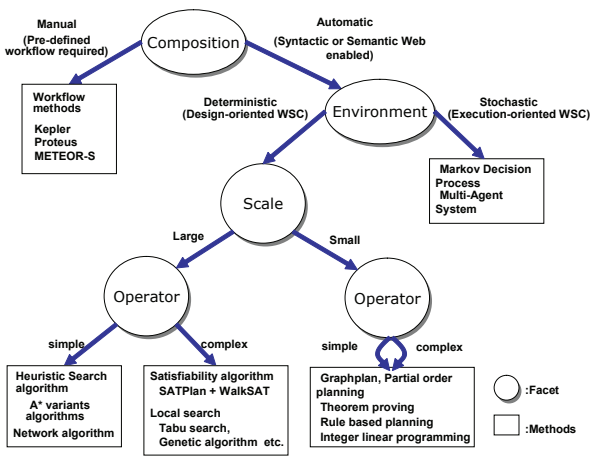
Regarding AI planning solutions for WSC in Figure 3., STRIPS (Nilsson, 2001) is the first

major AI planning system which represents actions in terms of their preconditions and effects and describes the initial and goal states as conjunctions of positive literals. Graphplan (Blum & Furst, 1995) is a general purpose planner for STRIPS-style domains, based on ideas used in graph algorithms. Given a problem statement, Graphplan uses a backward search to extract a plan and allows for some partial ordering among actions. As a satisfiability approach for the planning problems, the SATPlan algorithm (Kautz & Selman, 1999) was introduced. SATPlan belongs to the greedy local search methods for solving SAT problems. The SATPlan algorithm translates a planning problem into propositional axioms and applies a satisfiability algorithm to find a model that corresponds to a valid plan. An excellent survey of modern planning algorithms and their application to WSC problems can be found (Oh, Lee & Kumara, 2005; Rao & Su, 2004; Weld, 1999). However, we should note that these algorithms cannot handle the case when a semantic match is desired. In other words, for the purpose of the semantic match, a promise to use agreed-upon ontology is required between participating parties in advance.

From a planning objective point of view, Graphplan and SAT-based planning systems have the same objective (i.e., minimizing the number of time steps but not necessarily the number of actions to reach a goal). However, through an integer linear programming (ILP) fromulation, other various QoS factors (e.g., response time, service cost, or availability of sources) can be incorporated and optimized. Clearly, exhaustive search is unrealistic because combinatorial explosion renders it non-practical. When the scale is large but WSC problem is free of negation, we can use heuristic search algorithms like A* variant algorithms (Oh et al., 2005). SWORD (Ponnekanti & Fox, 2002) is a rule-based expert system which can automatically determine whether a desired composite service can be realized using existing rules. However, considering the fast growth of Web services, building a full knowledge base by converting all Web services into axioms will be expensive.

Different from the deterministic environment composition above, there are some studies to address the non-determinism inherent in real-world Web services. Markov decision

*Figure 3. A decision tree for classifying WSC problems.*



processes can be used to utilize a stochastic optimization framework in this context but main focus is on the abstract-level strategies instead of the implementation-level details (Doshi et al., 2005). A multi-agent perspective can be used to compose adaptive workflows in dynamic environment (Buhler & Vidal, 2003). However, it is likely to be expensive to enable Web services with agent-level intelligence.

In database community, one of the recent works attempts to support an automated service discovery using Information Retrieval (IR)-like similarity search (Dong et al., 2004). The emergent need of workflows to model e-service applications makes it essential that workflow tasks be associated with Web services. As a result, research efforts have been carried out to enhance workflow systems in their support of Web service composition (Shegalov el al., 2002).

As an independent research branch, *Web service quality testing* has been established where quality and trustworthiness of Web services are considered (Zhang & Zhang, 2005). It is very unlikely that a business organization will dynamically select a partner from the Internet merely based upon the information found from some public registry without high confidence. Therefore, the test of quality and trustworthiness of Web-service based software is a critical work for the success of Web service paradigms.

## THE WSPR (WEB SERVICE PLANNER) ALGORITHM

In the earlier section, we formulated the state model for the WSC problem as $\Psi=<S,s_0,S_G,\Omega(.),f,c>$ and showed that the size of state space is exponential to the size of parameter set. To address this intractable problem, we suggest a polynomial-time algorithm, *WSPR* (pronounced like "whisper"). When a request $r$ is given, WSPR activates two-step search. First, it computes the cost of achieving individual parameters staring 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 regression search leveraging on the results obtained from the first step as guidance. These two-step based approach is essentially in accordance to Graphplan (Blum & Burst, 1995). However, our method is different in that we used a noble heuristic to minimize the number of Web services in a solution while Graphplan and other AI planners originated from Graphplan typically aim at minimizing the number of time steps but not necessarily the number of actions. Note that due to the crossover between STRIPS model and the state model, in the forward searching, $s_0=r^i$ and $S_G \supseteq r^o$ while in the regression searching, $S_G=r^o$ and $s_0 \subseteq r^i$. Since this crossover can cause additional confusion, we will use $r^o$ and $r^i$ as it is in the algorithm rather than $s_0$ and $S_G$. However, we will use $s \in S$ in the algorithm so it must be noted that the state $s$ in the progression space has different meaning with in the regression space. While in the progression space, the states can be thought as sets of preconditions, in the regression space, the states can be thought as sets of effects. The differences will be described in detail below.

### (Step1) Forward Searching

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 the recursive equation as follows:

$$g_{r^i}(p) = \min_{w \in Ow(p)} [c(w) + \max_{p' \in w^i} g_{r^i}(p')]$$
$$= \min_{w \in Ow(p)} [1 + \max_{p' \in w^i} g_{r^i}(p')] \quad (1)$$

where, $c(w)$ is an invocation cost of a Web service and it is assumed to be 1. $Ow(p)$ is a set of Web services, $w \in W$ such that $p \in w^o$. At first, $g_{r^i}(p)$ are initialized to 0 if $p \in r^i$ and to $\infty$ otherwise. Then, the current information state $s$ is set as $r^i$. 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. 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 that contains the set of predecessor Web

*Figure 4. Forward search algorithm of WSPR*

**Input**: $r^i$ , $r^o$
**Output**: $PD_{ws}$
1 : $s = (r^i \setminus r^o)$ ; $C = \phi$ ; $d$=1
2 : while $\neg(s \supseteq r^o)$ do
3 : $\delta = \{w \mid w \in \Omega(s), w \notin C\}$ // if $\Omega(s) = \phi$ , no solution exists.
4 : for $p$ in $w^o$ ( $w \in \delta$ )
5 : if $g_{r^i}(p) = \infty$
6 : $g_{r^i}(p) = d$; $PD_{ws}(p) = w$ ; $s = s \bigcup \{p\}$
7 : $C = C \bigcup \delta$ ; $d$++

services of $p$. In this article, we assume that the invocation cost of Web services is non-negative. However, it is possible to have a negative Web service invocation cost if various QoS are considered (e.g., cost, quality, security). In that case, rather than Equation (1), label-correcting algorithms such as Bellman-Ford algorithm must be selected.

## (Step2) Regression Searching

In the second stage, WSPR approximates the optimal sequence of Web services that connects $r^i$ to $r^o$ by conducting the regression search leveraging on $g_{r^i}(p)$ and $PD_{ws}(p)$ obtained from the first step as guidance. In this article, we propose a heuristic-based greedy algorithm for the backward search. Backward search is an old idea in planning that is known as *regression search*. In regression search, the state $s$ can be thought as a set of effects and we can specify a sub-goal from the state $s$. This algorithm denotes its sub-goal by *subGoal* and starts by setting *subGoal* to($r^o \setminus r^i$). When we denote *wSpace* to be a set of Web services $w \in W$ such that $w_i \in PD_{ws}(p), p \in subGoal$, WSPR selects a Web service from *wSpace* by considering their heuristics at each backward step. This backward selection procedure is repeated until $subGoal \subseteq r^i$. The heuristic used to select a Web service and its underlying hypothesis is as follows:

***Hypothesis***: *Choosing a Web service with bigger contribution to match* ***the subgoal earlier*** *in the search* ***helps reaching to the initial*** *state faster.*

$$h_{sg}(w) = \mid w^o \bigcap subGoal \mid \qquad (2)$$

$h_{sg}(w)$shows that WSPR favors a Web service whose contribution to match the sub-goal is large. However, $h_{sg}(w)$ has other interpretation. A Web service, $w$ with bigger$h_{sg}(w)$ can have a higher chance to match the sub-goal fully leading to preventing proliferation of following search space. 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.

## Remarks

The forward searching procedure has polynomial computation time $O(|W|^2|P|)$ . Note that the length of a sequence of Web services to satisfy a request is limited by $|W|$. Therefore, there are at most $|W|$ iterations and at each iteration, at most $|W|$ Web services and $|P|$ parameters are examined. On the contrary, the regression searching procedure has polynomial computation time $O(|W|^2 \log|P|)$ because it has at most $|W|$ iterations and at each iterations, at most $O(|W| \log|P|)$ time is required to

*Figure 5. Regression search algorithm of WSPR*

**Input**: $r^i$, $r^o$, and $PD_{ws}$
**Output**: $w_1 \Rightarrow \cdots \Rightarrow w_n$
1 : $s = (r^o \setminus r^i)$; $subGoal = s$
2 : while $\neg(subGoal = \phi)$ do
3 : $wSpace = \bigcup_{p \in subGoal} PD_{ws}(p)$
4 : $\chi = \arg\max_{w \in wSpace} h_{sg}(w)$ // a tie breaks at random.
5 : $s = s \cup (\chi^o \setminus r^i)$
6 : $soln = soln \cup \{\chi\}$
7 : $subGoal = (subGoal \cup \chi^i) \setminus s$
8 : $s = r^i$ // a sequence of web service print in forward direction.
9 : while $\neg$ ($soln$=$\phi$ ) do
10: if $w \in \Omega(s)$ and $w \in soln$
11: Print $w$, $\Rightarrow$ ; $s = s \cup w^o$; $soln = soln \setminus \{w\}$

deal with the task to select a Web service $w$ with the largest $h_{sg}(w)$. The time for printing solution can be ignored. In general, since $|P| \succ\succ \log|W|$, $O(|W|^2|P|) \succ\succ O(|W|^2 \log|P|)$. Thus, the performance of WSPR algorithm is controlled by the first forward searching procedure. We verify this experimentally in subsequent sections. Consequently, the three significant dimensions to determine the performance of WSPR algorithm are: (1) the length of a sequence Web service in a solution, (2) $|W|$ of the Web service size and (3) $|P|$ of the parameter set size.

With $r_2$ of the motivating example in the introduction section, we can illustrate the forward searching algorithm.

- At the beginning, $s$=$r_2^i$. Since $\Omega(s)$= {`findHotel`} we can invoke `findHotel`. We can update $g_{r^i}(p)$, $PD_{ws}(p)$ and $s$ as follows: (1) $g_{r^i}$("hotelAddress")=1 and $g_{r^i}$("hotelZipcode")=1, (2) $PD_{ws}$("hotelAddress")= [`findHotel`] and $PD_{ws}$ ("hotelZipcode")= [`findHotel`], (3) $s$=$s$ $\cup$ {"hotelAddress", "hotelZipcode"}.
- With $s$, we can obtain $\Omega(s)$ = {`findRestaurant`, `guideRestaurant`}. By invoking `findRestaurant` and `guideRestaurant`, we can maintain $g_{r^i}(p)$, $PD_{ws}(p)$ and $s$ as follows: (1) $g_{r^i}$ ("restaurantRate")=2, $g_{r^i}$("restaurantAddress")=2, $g_{r^i}$("restaurantName")=2, and $g_{r^i}$("restaurantPhonenumber")=2, (2) $PD_{ws}$ ("restaurantRate") = [`guideRestaurant`], $PD_{ws}$("restaurantAddress")= [`guideRestaurant`,`findRestaurant`], $PD_{ws}$ ("restaurantName")= [`findRestaurant`], and $PD_{ws}$ ("restaurantPhonenumber")= [`findRestaurant`], (3) $s$=$s$ $\cup$ {"restaurantRate", "restaurantAddress"}.
- With $s$, we can obtain $\Omega(s)$= {`findDirection`}. Also, we can maintain $g_{r^i}(p)$, $PD_{ws}(p)$and $s$ as follows: (1) $g_{r^i}$ ("mapHotelRestaurant")=3 and $g_{r^i}$ ("directionHotelRestaurant")=3, (2)$PD_{ws}$ ("mapHotelRestaurant") =[`findDirection`] and $PD_{ws}$ ("directionHotelRestaurant")=[`findDirection`], (3) $s$=$s$ $\cup$ {"mapHotelRestaurant", "directionHotelRestaurant"}.

- We can stop the procedure because $s \supseteq r_2^o$.

We illustrate the regression planning algorithm continued from the end of the forward searching procedure above.

- At the beginning, both $s = r_2^o$ and $subGoal = s$ hold.
- Since both $PD_{ws}$("mapHotelRestaurant") and $PD_{ws}$("directionHotelRestaurant") are `findDirection`, we get $\chi$ =`findDirection`, even without applying the heuristic. Then, we obtain $soln$={`findDirection`}, and update $s = s \cup$ {"restaurantAddress", "hotelAddress"} and $subGoal$={"restaurantAddress", "hotelAddress"}.
- Since $PD_{ws}$("restaurantAddress")=[`guideRestaurant, findRestaurant`] and $PD_{ws}$ ("hotelAddress")= [`findHotel`], $wSpace$ = {`findHotel, findRestaurant, guideRestaurant`}. Since each Web service in $wSpace$ has $h_{sg}(w)=1$ uniformly, we can randomly select $\mathcal{X}$= {`findHotel`} and then, we can update $s = s \cup$ {"hotelAddress", "hotelZipcode"} and $subGoal$={"restaurantAddress"}. We can update $soln$ = {`findHotel, findDirection`}.
- Since $PD_{ws}$("restaurantAddress")=[`guideRestaurant, findRestaurant`], $wSpace$={`findRestaurant, guideRestaurant`}. Since each Web service in $wSpace$ has $h_{sg}(w)=1$ uniformly, we can randomly select $\mathcal{X}$= {`findRestaurant`} and then, we can update $s = s \cup$ {"hotelAddress", "hotelZipcode"} and $subGoal = \phi$. We can update $soln$ = {`findHotel, findDirection, findRestaurant`}.
- Since $subGoal = \phi$, we stop the repetitions.
- Set $s = r_2^o$ and print "`findHotel`" and $\Rightarrow$ because $\Omega(s)$ = [`findHotel`].
- Set $s = s \cup$ {"`hotelAddress`", "`hotelZipcode`"} and print "`findRestaurant`" and $\Rightarrow$ because $\Omega(s)$ = [`findRetaurant`].
- Set $s = s \cup$ {"`restaurantAddress`", "`restaurantRate`"} and print "`findDirection`" and $\Rightarrow$ because $\Omega(s)$ = [`findDirection`].
- Since $soln = \phi$, we stop the regression search with a solution: "`findHotel`" $\Rightarrow$ "`findRestaurant`" $\Rightarrow$ "`findDirection`".

## EXPERIMENTAL VALIDATION

To validate the efficiency of the WSPR, we conducted extensive experiments using two publicly available test sets—EEE05[5] and ICEBE05[6]. Both test sets contain artificially created composition scenarios, and are used for two recent Web services discovery and composition contests. However, the test sets in EEE05 appear to be manually created by human experts (thus smaller in size) while those in ICEBE05 appear to be synthetically generated by software. Throughout the experiments, we used two evaluation metrics: (1) Running time (in millisecond) is measured to see how long it takes for a method to find a solution, and (2) The number of Web services in the solution (#$W$) is counted to measure the quality of the found solution. Methods that take *smaller* running time while producing solutions that use *less* number of Web services are considered to be "good" ones. All experiments were performed on a PC platform with Pentium 4 (1.7GHz), Windows XP, and 523MB RAM. All algorithms were implemented in Python 2.3.

### Experiments with EEE05

This test set contains 100 Web services. Although small, it is still challenging because it is not simple for humans to solve for the optimal configuration in short time. It should be noted that EEE05 contest had offered 15 test requests but six test sets was not able to be solved syntactically; we discard those six test requests (i.e., 4, 6, 7, 9, 11, and 12). Since, this test set and test requests were made by humans and tried to imitate real public Web services so that it is possible to identify parameters used in it. For example, 15th test request has $r_{in}$

={"pickupLocationName," "pickupLocationID," "firstName," "lastName," "middleInitial," "custStreetAddress," "custCityAddress," "custStateAddress," "custZipAddress"} and $r_{out}$={"shipmentTrackingNumber,""shipment Cost"}. For each algorithm, we first checked whether WSPR solves the problem, and if so, we compute *Time* and *#W*. The results are shown in Table 2. For each request, WSPR generated answers that are identical to the right answers announced by EEE05 with reasonable speed (≤0.01 sec).

## Experiments with ICEBE05

According to the description from ICEBE05, the construction of test data is varied in different dimensions as follows:

- Minimal *#W*: They provide test sets with their minimal *#W* between 2 and 4 (test set names starting with Composition-1) and other test sets with their minimal *#W* between 6 to 8 (test set names staring with Composition-2).
- The number of Web services in a test set (i.e., $|W|$): $|W|$ of test sets is one among 3,356 and 5,356 and 8,356. $|W|$ of test sets can be identified from their test set name because 20, 50, and 100 figures included in the name indicate 3,356, 5,356, and 8,356 respectively.
- Number of input and output parameters: The size of input and output parameters were differentiated in the range of 4-8, 16-20, and 32-36.

*Table 2. WSPR performance on EEE05 in terms of #W and Time (ms)*

| Request No. (EEE Test number) | WSPR | |
|---|---|---|
| | *#W* | *Time* |
| 1(1) | **4** | **10** |
| 2(2) | **4** | **10** |
| 3(3) | **3** | **10** |
| 4(5) | **1** | **10** |
| 5(8) | **2** | **10** |
| 6(10) | **1** | **10** |
| 7(13) | **2** | **10** |
| 8(14) | **2** | **10** |
| 9(15) | **1** | **10** |

The test set names in ICEBE05 indicate the combination of the parameters by labeled Composition-1 and Composition-2 followed by the test set size and the number of parameters. For example, Composition1-50-32 means the test set representing service composition-1 with a minimal *#W* between 2 to 4 and $|W|$ of 5,356 and the number of input and output parameters being in the range of 32 to 36.

Both Composition-1 and Composition-2 have nine test sets respectively and each test set has 11 queries. As our experiments will show, solving requests in Composition-2 is harder than Composition-1 in terms of *#W* and *Time*. For example, *#W* in Composition-1 is in the range of 2 and 4 as shown in Table 3. while *#W* in Composition-2 is in the range of 6 and 8 as shown in Table 5. In terms of *Time*, *Time* in Composition-1 is in the range of 49~703 while *Time* in Composition-2 is in the range of 96~2281. This implies that we need more resources (i.e., longer *Time* and larger *#W*) in Composition-2 than Composition-1. We also checked how resource consumption is divided into two phases of WSPR. For this purpose, first, we denoted *fTime* to represent the time consumed in forward search and *bTime* to represent the time consumed in backward search. Then, we recorded *fTime* and *bTime* in each search.

### (1) Composition-1

From the perspective of *#W*, as shown in Table 3., we discovered that 11$^{th}$ request was the simplest while requests from 7$^{th}$ to 10$^{th}$ were the hardest in the sense that 11$^{th}$ request was solved with just one Web service, requests from 7$^{th}$ to

*Table 3. Comparison of #W over requests. All test sets have the same #W for the same request No.*

| Request No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *#W* | 2 | 2 | 2 | 3 | 3 | 3 | **4** | **4** | **4** | **4** | **1** |

*Table 4. Comparison of 11 queries over 9 test sets in terms of their fTime and **bTime (ms)***

| Request No. | *Time* | 20-4 | 20-16 | 20-32 | 50-4 | 50-16 | 50-32 | 100-4 | 100-16 | 100-32 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *fTime* | 46.9 | 109.9 | 187 | 46.9 | 125 | 202.9 | 94 | 219 | 344 |
| | *bTime* | 16 | 2 | 2 | 15.1 | 2 | 31 | 14.9 | 2 | 14.9 |
| 2 | *fTime* | **47** | 110 | 172 | 62 | 171.9 | 217.9 | 109.9 | 203 | 343.9 |
| | *bTime* | **2** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 15 |
| 3 | *fTime* | 62 | 109.9 | 171.9 | 62 | 141 | 217.9 | 110 | 219 | 358.9 |
| | *bTime* | 16 | 2 | 2 | 2 | 2 | 16 | 2 | 2 | 2 |
| 4 | *fTime* | 62.9 | 140.9 | 250 | 108.9 | 202.9 | 328.9 | 139.9 | 312 | 500 |
| | *bTime* | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 5 | *fTime* | 62.9 | 140 | 234.9 | 78 | 187.9 | 328.9 | 139.9 | 296.9 | 671.9 |
| | *bTime* | 2 | 2 | 15.1 | 16 | 15.1 | 2 | 2 | 16 | 2 |
| 6 | *fTime* | 62.9 | 141 | 265.9 | 109.9 | 250 | 258.9 | 157 | 297 | 515 |
| | *bTime* | 15.1 | 2 | 15 | 2 | 2 | 116.1 | 2 | 2 | 2 |
| 7 | *fTime* | 78 | 187 | 328 | 109.9 | 250 | 406 | 187.9 | 391 | 639.9 |
| | *bTime* | 16 | 2 | 2 | 15.1 | 2 | 2 | 2 | 2 | 16 |
| 8 | *fTime* | 92.9 | 203 | 328 | 125 | 328 | 421 | 171.9 | 389.9 | 671.9 |
| | *bTime* | 16 | 2 | 2 | 2 | 16 | 2 | 2 | 2 | 2 |
| 9 | *fTime* | 92.9 | 218.9 | 328 | 109.9 | 250 | 406 | 203 | 375 | **671** |
| | *bTime* | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | **32** |
| 10 | *fTime* | 92.9 | 187 | 327.9 | 108.9 | 233.9 | 453.9 | 187 | 407 | 657 |
| | *bTime* | 2 | 2 | 2 | 2 | 16.1 | 2 | 2 | 14.9 | 2 |
| 11 | *fTime* | 77.9 | 187 | 359 | 78.9 | 156 | 344 | 92.9 | 187.9 | 312 |
| | *bTime* | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

10th required as many as four Web services.

From the perspective of test set's difficulty level, as shown in Table 4 and (a)-(c) of Figure 6, we found that 20-4 test set is the simplest while 100-32 test set is the hardest. These results are very closely related with the size of test sets. In other words, since 20-4 and 100-32 have 3,356 and 8,356 Web services respectively, the search space of 100-32 becomes much bigger than that of 20-4.

In terms of *Time*, 9th request in 100-32 test set takes the longest one (i.e., 703 ms). When it comes to resource consumption in two different phases, most resource consumption was assigned to the forward reasoning stage. No *bTime* did exceed 3 ms as shown in Table 4. It is caused by two reasons: (1) all requests are solved by linearly sequential full-matching composition. Consequently, in the test sets in ICEBE05, we can save $O(|W|\lg|W|)$ computations because we need not measure $h_{sg}(w)$ at all. Instead, we can just select one Web service listed in *wSpace*. In other words, while first forward reasoning step keeps the computational cost as high as $O(|W^2||P|)$, second regression planning step brings the cost down to $O(|W|)$ (2) the solution length obtained is very small. For example, in 100-32 test set, 10th request can be solved with 4 Web services. That is, regression planning reaches to $r^i$ starting from $r^o$ with 4 iterative steps ($\prec\prec |W|$).

### (2) Composition-2

Like Composition-1, from the perspective of *#W*, we discovered that 11th request was the simplest but requests from 7th to 10th were the hardest as illustrated in Table 5. While the 11th request requires just one Web services, request from 7th to 10th requires eight Web services.

Similar to Composition-1, From the perspective of test set's difficulty level, we found that 20-4 is the simplest but 100-32 is the hardest as shown in Table 6 and (d)-(f) of Figure 6. In terms of *Time*, 10th request in 100-32 takes the longest one (i.e., 2280.9 ms). When it comes to resource consumption in two different phases, most resource consumption was assigned to the forward reasoning stage. No *bTime* did exceed 30 ms as shown in Table 6.

**Summary.** Although WSPR spends extra resources to maintain auxiliary structures such as $g_{r^i}(p)$ and $PD_{ws}(p)$ in the parameter space, overall we concluded that it is beneficial because WSPR can be more informed in regression searching step due to the guidance of $g_{r^i}(p)$ and $PD_{ws}(p)$. However, most running time of WSPR is spent in the forward reasoning stage due to the sheer number of Web services to visit. Thus, in order to improve the overall speed of WSPR, better ways to be more informed about parameter space are needed in the future research.

## DISCUSSION ON EEE05 AND ICEBE05 TEST SETS

In this section, we describe the result of study on the test sets used in EEE05 and ICEBE05. Being featured as the first and second world-wide contest, the two challenges obviously attracted considerable interest of many researchers working in the frontier field of Web service research. As the results of our study will show, however, some directions

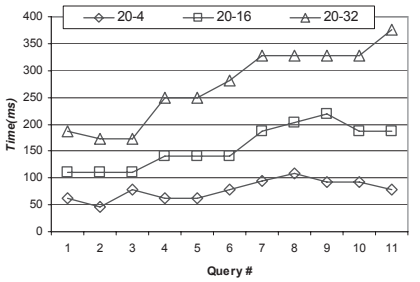
*Table 5. Comparison of #W over queries. All test sets have the same #W for the same request No.*

| Request No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *#W* | 6 | 6 | 6 | 7 | 7 | 7 | **8** | **8** | **8** | **8** | **1** |

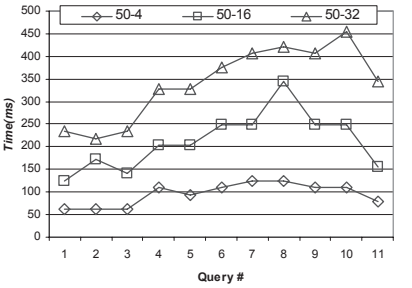
*Table 6. Comparison of 11 queries over 9 test sets in terms of their fTime and bTime (ms)*

| Request No. | *Time* | 20-4 | 20-16 | 20-32 | 50-4 | 50-16 | 50-32 | 100-4 | 100-16 | 100-32 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *fTime* | 187.9 | 828.9 | 672 | 312 | 641 | 1093 | 500 | 1015.9 | 1734.9 |
| | ***bTime*** | 2 | 2 | 2 | 2 | 2 | 15.9 | 16 | 2 | 2 |
| 2 | *fTime* | 187 | 780.9 | 671.9 | 328.9 | 655.9 | 1141 | 516 | 1016 | 1733.9 |
| | ***bTime*** | 16 | 16 | 2 | 2 | 2 | 14.9 | 2 | 15 | 2 |
| 3 | *fTime* | 203 | 796.9 | 671.9 | 312.9 | 641 | 1094 | 500 | 1015.9 | 1717.9 |
| | ***bTime*** | 2 | 2 | 16 | 16 | 2 | 2 | 15 | 2 | 16 |
| 4 | *fTime* | 234.9 | 906 | 780.9 | 359.9 | 750 | 1312.9 | 594 | 1171.9 | 2000 |
| | ***bTime*** | 2 | 16 | 2 | 15.1 | 2 | 15 | 15.9 | 2 | 16 |
| 5 | *fTime* | 233.9 | 922 | 782 | 390.9 | 750 | 1282 | 594 | 1187 | 2000 |
| | ***bTime*** | 2 | 15 | 15 | 2 | 15.9 | 15 | 15 | 16 | 15.9 |
| 6 | *fTime* | 218.9 | 905.9 | 781 | 359.9 | 750 | 1250 | 592.9 | 1171.9 | 2000 |
| | ***bTime*** | 15 | 16 | 2 | 15.1 | 2 | 32 | 16 | 15.1 | 16 |
| 7 | *fTime* | 391 | 1030.9 | 890 | 421.9 | 842.9 | 1467.9 | 655.9 | 1328.9 | 2266 |
| | ***bTime*** | 2 | 16 | 15.9 | 16 | 16 | 15.8 | 32 | 2 | 16 |
| 8 | *fTime* | 264.9 | 1046.9 | 875 | 437 | 828 | 1421 | 657 | 1327.9 | 2266 |
| | ***bTime*** | 2 | 16 | 30.9 | 2 | 16 | 16 | 30 | 2 | 2 |
| 9 | *fTime* | 266 | 1046.9 | 875 | 421 | 844 | 1469 | 671 | 1327.9 | 2264.9 |
| | ***bTime*** | 2 | 2 | 16 | 2 | 16 | 2 | 16 | 16 | 2 |
| 10 | *fTime* | 250 | 1030.9 | 875 | 421.9 | 844 | 1546.9 | 671.9 | 1328.9 | **2265.6** |
| | ***bTime*** | 2 | 15.1 | 30.9 | 2 | 14.9 | 2 | 2 | 2 | **15.3** |
| 11 | *fTime* | **94** | 296.9 | 406 | 108.9 | 250 | 453 | 125 | 266 | 468.9 |
| | ***bTime*** | **2** | 15.1 | 2 | 2 | 16 | 2 | 2 | 2 | 2 |

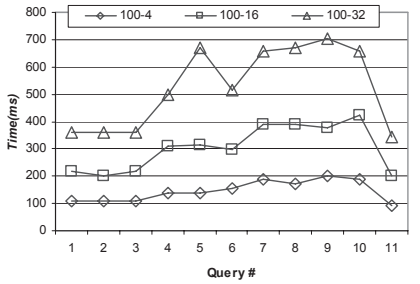
*Figure 6. Comparison of test sets in ICEBE in terms of their Time*



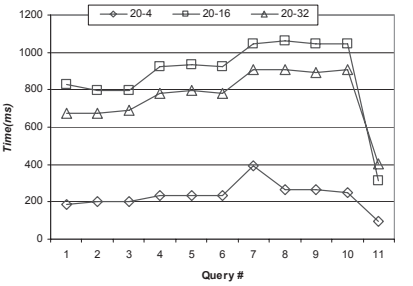

*(a) Comparison of Time in Composition1*



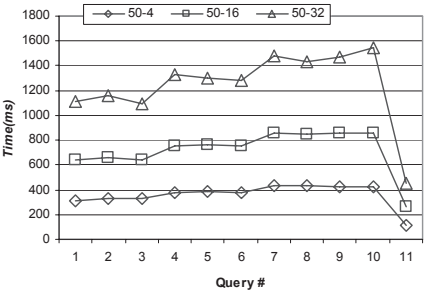

*(b) Comparison of Time in Composition1*



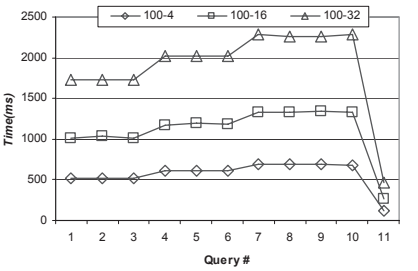

*(c) Comparison of Time in Composition1*




*(d) Comparison of Time in Composition2*




*(e) Comparison of Time in Composition2*




*(f) Comparison of Time in Composition2*

pursued in both challenges are on differing expectations on the current status and future evolution of Web services. For that reason, we attempt to describe some improvements sought to guide the next challenges to take more fruitful directions. To this end, first, we took snapshots of the challenges in the perspective of parameter usage distributions. Second, we investigated the difficulty level of compositions by checking whether the queries can be solved by using full-matching operator alone. Lastly, we discuss the evolution of syntactic to semantic matching approaches.

## Parameter usage distribution

We can define the parameter usage to be the occurrence number of the parameter in the corresponding Web service repository and denote it by #($p$), where $\forall p \in P$. In order to help readers understand the new concept of #($p$) and #($p$) distribution, we illustrate the #($p$) distribution of existing public Web services as shown in Figure 7. To obtain this distribution, we first downloaded 1,544 raw WSDL files[7]. Those WSDL files were originally distributed in different registries (Binding point, Salcentral, Web service list, WebserviceX.NET & XMethod) but gathered by crawlers (Fan & Kambhampati, 2004). Since a lot of files among them had been proved to be invalid judging from their conformity with WSDL DTD (i.e., syntactic grammar error and missing field), we had to prune out those invalid ones first. At the end, we had 670 valid WSDL files left. Then, we converted semantically meaningless parameters into semantically meaningful parameters 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 do not provide any semantics required to understand what the results or returns are. In this case, we replace the terms with their operation name or Web service name. Sometimes, an operation name is a sequence of concatenated words thereby being hard to conduct the replacement tasks. In that case, we did a proper token segmentation and extract primal terms from the tokens using the lexical analysis. For example, if an operation name is "getAuthornamefromPaper" or "searchAuthornamesbyPaper," we extracted "Authornames" and replaced "result(s)" or "returns(s)" with the "Authornames."

After the preprocessing tasks above, we built a set, $P$={all parameters found in the valid 670 WSDL files} and then, we measured #($p$) for $\forall p \in P$. Then, we draw the #($p$) distribution where x-axis represents #($p$) and y-axis represent the number of parameters with same #($p$). The distribution has no humps. We also plotted a power-function, $\alpha(1/k)^{\beta}$ over the #($p$) distribution and obtained that the exponent, $\beta$ is 1.5026. Although 1.5026 does not suffice the requirement to assert that the distribution follows the power law (Denning, 2004), the distribution is highly skewed enough to be seen as a Zipf distribution. Indeed, the parameters such as "license key," "start date," "end date" or "password" have large #($p$) but most variables appear just once. This observation also implies the existence of hub parameters which appear in Web service frequently and serve important roles on the inter-connections between Web services.

Similarly, we have generated the parameter usage distributions and plotted the power-function for the test sets used in EEE05 and ICEBE05. As shown in Figure 8(a)., the test set used in EEE05 has a highly skewed Zipf distribution without any hump. Interestingly, exponent, $\beta$ = 1.3012 of EEE05 is as high as that of the existing public Web services. As a whole, EEE05 test set can be seen as existing public Web services in terms of #($p$) distribution shape. However, the size of the EEE05 is just 100 which is too small to represent the current state or future evolution of Web services.

Regarding ICEBE05 test sets, in this article, we just show three distributions for 100-4, 100-16 and 100-32 of Composition-2 due to the lack of space. However, we can simply assert that the rest of the distributions show a similar shape as the three distributions. As opposed to the EEE05, all ICEBE05 test sets have $\beta \leq 0.5$ and four equal humps. For example, Figure 8(d) shows four humps at

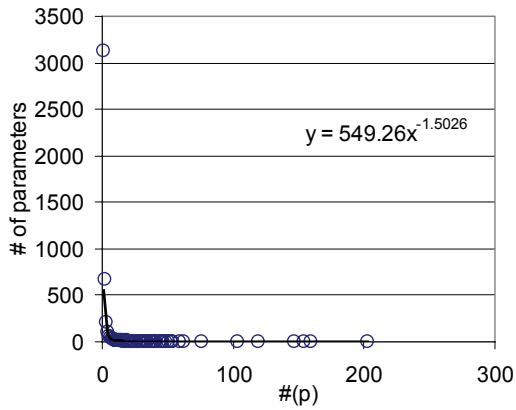
around 1, 100, 200, and 800 with the highest at third hump. This shape of distribution differs considerably from EEE05 and existing public Web services. However, before we proceed to point out that the assumptions to build those test sets are conflicting with the state of existing public Web services, one thing should be noted. That is, many researchers claim that most applications of Web services are likely to be in intra-corporate scenarios rather than on the public Web. Under the intra-corporate scenarios, it is questionable as to whether their #($p$) still follow Zipf-like distributions. In fact, rather than a Zipf distribution, it is likely to be either a bell-shape or a uniform-shape distribution because in general, intra-corporate Web services tend to be access-restricted or well-designed to avoid such an extreme hub parameters because of the load balancing and efficiency problems or the network survivability issues.

However, even though we allow exceptions by acknowledging the existence of different Web service applications, #($p$) distributions with four humps of ICEBE05 still look too artificial.

## Linearly sequential full-matching

As mentioned before, WSC problem can be addressed in the linear planning approach in AI community. The linear planning approach requires only AND operator. In the WSC context, the AND operator can be partitioned into "full-matching" operation and "partial-matching" operation. Interestingly, EEE05 and ICEBE05 can be solved by using "full-matching" operation alone. That is, all requests can be satisfied by chaining Web services in such a way that a predecessor Web service can fully match the successor Web service. Formally speaking, suppose that *Sol* denotes the answer to a request $r$ provided, then $\exists w_i \in Sol$ such that $w_1^i \subseteq r^i$ and either $w_i^i \subseteq w_{i-1}^o$ or $(w_i^i \setminus r^i) \subseteq w_{i-1}^{out}$ for $\forall i=2,...,|Sol|$. For example, the 2nd request of EEE05 can be solved by a chain of Web services: *findMostRelevantStock* → *getStockPriceMany* → *performStockResearch* → *purchaseOptimalStock* such that $findMostRelevantStock^i \subseteq r^i$, $getStockPriceMany^i \subseteq findMostRelevantStock^o$, $performStockResearch^i \subseteq getStockPriceMany^o$, $purchaseOptimalStock^i \subseteq performStockResearch^o$, and $r^o \subseteq purchaseOptimalStock^o$. Likewise, the 4th request in Composition2-100-32 of ICEBE05 can be solved by a chain of Web services[8]: *s34a1827462*

*Figure 7. Parameter degree, #($p$) distribution for existing web services*



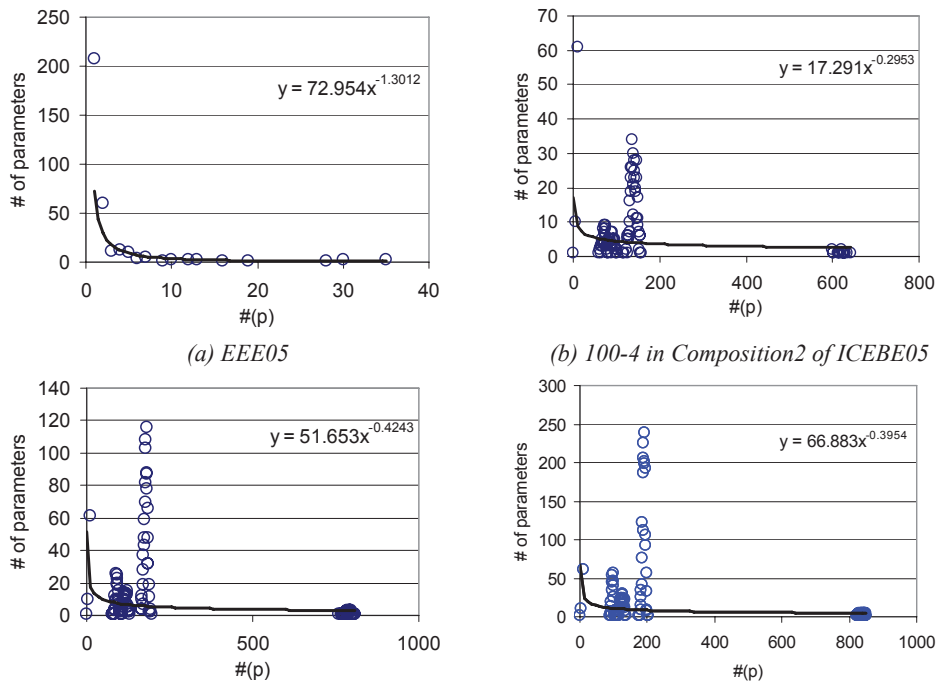
→ *s30a2162659* → *s92a3046444* → *s60a4328073*→*s19a0320284*→*s09a7063054* → *s58a4762157* such that $s34a182746^i \subseteq r^i$, $(s30a2162659^i \backslash r^i) = \{p76a0550626\} \subseteq s34a1827462^o$, $(s92a3046444^i \backslash r^i) = \{p15a6821354\} \subseteq s30a2162659^o$, $(s60a4328073^i \backslash r^i) = \{p54a2653485\} \subseteq s92a3046444^o$, $(s19a0320284^i \backslash r^i) = \{p30a6922384\} \subseteq s60a4328073^o$, $(s09a7063054^i \backslash r^i) = \{p53a0652249\} \subseteq s19a0320284^o$, $(s58a4762157^i \backslash r^i) = \{p67a9261643\} \subseteq s09a7063054^o$ and finally, $r^o \subseteq s58a4762157^o$.

Note that other queries in EEE05 and ICEBE05 also take the same solution scheme (i.e., linearly and sequential full-matching) as the examples described above. When an information gathering problem takes this special scheme, the problem can be solved quickly. In particular, if every step of a composition process requires only one parameter, it can be seen as a simple single source shortest-path problem. For example, in the 4th request in Composition2-100-32 of ICEBE05, the problem can be solved by finding a shortest-path conversely from the goal parameter, $r^o$ = {p94a3566970} to $r^i$ by gathering {p67a9261643}, {p53a0652249}, {p30a6922384}, {p54a2653485}, {p15a6821354} and {p76a0550626}, subsequently.

## Semantic matching

Both EEE05 and ICEBE05 assumed only syntactic (text-based) matching among Web services—if two parameters have the same spellings, then they are matched. In general, however, such restriction is too rigid to be useful. For instance, two parameters with different

*Figure 8. Parameter degree,#(p)distribution for EEE05 and ICEBE05*




*(a) EEE05*

*(b) 100-4 in Composition2 of ICEBE05*

*(c) 100-16 in composition2 of ICEBE05*

*(d) 100-32 in composition2 of ICEBE05*

spellings may have same semantic meaning, and thus are inter-changeable (e.g., "price" and "fee"). Conversely, two parameters with the same spelling may have different meaning too (e.g., "title" may mean either "book title" or "job title"). Therefore, solely relying on the syntactic similarity of parameters for determining if two Web services can be composed together is insufficient. For instance, one request in ICEBE05 had its composition length as long as eight. It is questionable if it is reasonable to compose eight Web services in chain with only syntactic matching approaches. Although the preponderance of Web services is expected in future, syntactic matching is not likely to achieve such a long composition. One of the recent works (Dong et al., 2004) attempts to suggest plausible composition of service operations in real public Web services using "approximate" matching via IR-style keyword search. Interestingly, they found out that there are no compositions with more than two Web services linked. Their results points out two lessons: First, we are still in the formative stage as far as publicly available Web services are concerned, and second, it is not easy to discover correlations between Web services even while using the semantic description matching.

### Summary

Based on the above analysis, we suggest three improvements as follows:

- It would be more realistic if test sets are constructed such that their parameter usage distributions follow either Zipf, Bell-shape, or uniform distributions. The Zipf distribution is more appropriate to reflect public Web services with hub parameters, while bell-shape or uniform distribution is more appropriate to reflect intra-corporate style Web services.
- It would be intellectually more interesting if test queries require more complex operators such as combination of full and partial matching operation. Such an extension would make the WSC problem much harder yet more practical.
- By allowing semantic matching, test sets can mimic real semantic Web services better. Not only the approximate syntactical matching but also the semantic reasoning using RDF or OWL can make test sets richer and more realistic. Developing novel ways to measure the "semantic" distance between parameters and Web services would be an interesting research issue.

## CONCLUSION

In this article, we have considered the Web services composition (WSC) problem, and proposed the WSPR, an AI planning based two-step algorithm that is effective with respect to the speed and the quality of solutions. We experimentally validated our claims using two publicly available test sets—EEE05 and ICEBE05. Finally, we suggested three improvements to test sets for WSC problem in future.

## REFERENCES

Ahuja, R., Magnanti, T. & Orlin, J. (1993). *Network flows*., New Jersey:Prentice-Hall.

Altintas, I., Jaeger, E., Lin, K., Ludaescher, B. & Menon, A. (2004). *A Web Service Composition and Deployment Framework for Scientific Workflows*. Proceedings of the 2nd IEEE International Conference on Web Services (ICWS*)*, San Diego, CA.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The Semantic Web*, Scientific American*. June.

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

Binding point. http://www.bindingpoint.com

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

Bonet, B. & Geffner, H. (2001). Planning as heuristic search, *Artificial Intelligence*, *129*(1-2), 5-33.

Buhler, P. & Vidal, J. M. (2003). *Adaptive Workflow Enactment=Web Services + Agents*. Proceeding of the 1st IEEE International Conference on Web Services (ICWS*)*,

Las Vegas, NV.

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

Cantera, M. (2004). *IT professional services forecast and trends for Web services*. ITES-WW-MT-0116, Gartner Inc.

Carman, M., Serafini L. & Traverso, P. (2003). *Web service composition as planning. Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS),* Trento, Italy.

Christophides, V., C. Houstis, C., Lalis, S. & Tsalapata, H. (1999). *Ontology-driven Integration of Scientific Repositories*. Proceedings of the 4th Workshop on Next Generation Information Technologies and Systems (NGITS), Habart Habaron, Israel.

Constantinescu, I., Faltings, B. & Binder, W. (2004). *Large Scale Testbed for Type Compatible Service Composition.* Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS), Whistler, British Columbia, Canada.

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

Dong, X., Halevy, A., Madhavan, J., Nemes, E. & Zhang, J. (2004). *Similarity Search for Web Services*. Proceedings of the 30th Very Large Data Bases (VLDB), Toronto, Ontario, Canada.

Doshi, P., Goodwin, R., Akkiraju, R. & Verma, K. (2005). Dynamic workflow composition: Using markov decision processes. *International Journal. of Web Services Research*, *2*(2), 1-17.

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

Ghandeharizadeh, S. et al., (2003). *Proteus: A System for Dynamically Composing and Intelligently Executing Web Services.* Proceeding of the 1st IEEE International Conference on Web Services (ICWS), Las Vegas, NV.

Haslum, P. & Geffner, H. (2000). *Admissible Heuristics for Optimal Planning*. Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS), Breckenridge, CO.

Kautz, H. & Selman, B. (1999). *Unifying SAT-based and Graph-based Planning.* Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden.

Kwok, C. T. & Weld, D. S. (1996). *Planning to Gather Information.* Proceedings of the 13th National Conference on Artificial Intelligence (AAAI), Portland, Oregon.

McDermott, D. (1996). *A Heuristic Estimator for Means-ends Analysis in Planning*. Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS), Edinburgh, Scotland.

Nilsson, Nils J. (2001), *Artificial Intelligence: A New Synthesis*. San Francisco: Morgan Kaufmann.

Oh, S.-C., Lee, D. and Kumara, S. (2005), A comparative illustration of AI planning-based Web services composition, *ACM SIGecom Exchanges*, 5(5), 1-10.

Oh, S.-C., On, B., Larson, E. J. & Lee, D. (2005). *BF*: Web Services Discovery and Composition as Graph Search Problem.* Proceedings of the 7th International IEEE Conference on e-Technology, e-Commerce and e-Service (EEE), Hong Kong, China.

Ponnekanti, S. R. & Fox, A. (2002). *SWORD: A developer toolkit for Web service composition*. Proceedings of the 11th World Wide Web (WWS), Honolulu, HI.

Rao, J. & Su, X. (2004). *A Survey of Automated Web Service Composition Methods.* Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), San Diego, CA.

Russell, S. J. & Norvig, P. (2002). *Artificial intelligence: A modern approach*, 2nd edn,

New Jersey: Prentice-Hall.

Salcentral. http://www.salcentral.com.

Shegalov, G., Gillmann M. & Weikum G. (2002). XML-enabled workflow management for e-services across heterogeneous platforms. *The VLDB Journal*, *10*(1), 91-101.

Sivashanmugam, K., Verma, K., Sheth, A. & Miller, J. (2003). *Adding Semantics to Web Services Standards.* Proceeding of the 1st IEEE International Conference on Web Services (ICWS*)*, Las Vegas, NV.

Tuomas S. (1999). *An Algorithm for Optimal Winner Determination in Combinatorial Auctions.* Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI). Stockholm, Sweden.

Vossen, T., Ball, M., Lotem, A. & Nau, D. (1999). *On the use of integer programming models in AI planning.* Proceeding of the 16th National Conference on Artificial Intelligence (AAAI), Orlando, FL.

Web service list.http://www.Webservicelist.com

WebserviceX.NET http://www.webserviex.com

Weld, D. S. (1999). Recent advances in AI planning. *AI Magazine*, *20*(2), 93-123.

xMethod http://www.xmethod.com

Zhang, J. & Zhang, L.-J. (2005). Web services quality testing, *International Journal of Web Services Research*, *2(2)*, 1-4.

## ENDNOTES

1 Suppose that the amount of material must flow from a source node to a sink node. The minimum cost flow problem is to find paths with the minimum total flow cost.

2 Let $v_i$ be the shortest distance from node $i$ to the root node $n$. These quantities are known as labels. Algorithms which find labels through a sequence of iterations are called *label-correcting* algorithms.

3 We do not advocate the four facets for WSC classification – we merely use them to simplify the comparison of our approaches against existing ones from literatures.

4 The problem is considered anomalous because the non-interleaved planners cannot solve it.

5 The URL of EEE05 Web service contest is http://www.comp.hkbu.edu.hk/~eee05/contest/

6 The URL of ICEBE05 Web service contest http://www.comp.hkbu.edu.hk/~ctr/wschallenge/

7 The data is available at http://rakaposhi.eas.asu.edu/PublicWebServices.zip

8 In the original data set, *s34a1827462* is called *servicep34a1827462* (and others as well). We shortened them here for a simpler presentation.

*Seog-Chan Oh has been a PhD student in the Industrial and Manufacturing Engineering department since 2002. He earned his BS and MS from Dongguk University (1993 and 1996, respectively). Before he began his PhD studies, he worked as an IT consultant for seven years at Daewoo Information Systems. His research interests include Web service composition and optimization, AI, MAS and Web intelligence. He has the certification of Professional Engineer at Information Management issued by the Korean government.*

*Dongwon Lee has been an assistant professor in the College of Information Sciences and Technology, at Pennsylvania State University, , since 2002. He earned a BS from Korea University (1993), an MS from Columbia University (1995), and a PhD from UCLA (2002), all in computer science. His research interests include database, XML and Web analysis, and web services and semantic web. He has (co-)authored about 40 scholarly articles in conferences or journals.*

*Soundar R. T. Kumara is distinguished professor of Industrial and Manufacturing Engineering department. His research interests are in intelligent systems with an emphasis on sensor-based equipment monitoring and diagnosis, software agents, and complexity in supply chain. He has more than 150 publications. He has won several awards and is an elected member of the International Academy of Production Research.*