# Flexible Web Services Discovery and Composition using SATPlan and A* Algorithms

Seog-Chan Oh[1], Dongwon Lee[2], Soundar Kumara[1]

[1] Industrial and Manufacutring Engineering Department
[2] School of Information Sciences and Technology
, Penn State University, USA
{sxo160, dongwon, skumara}@psu.edu

**Abstract.** Agents with web services based interfaces have service description encoded in machine-understandable formats so that they can easily interact with other agents. Therefore, locating right agents and combining them to form more complex services becomes increasingly an important task on the Web. However, when there are a large number of web services based agents available, it is non-trivial to quickly discover and combine them to satisfy the given request. Toward this problem, in this paper, we propose to adopt solutions developed in the context of AI search and planning. In particular, we present two algorithms, *BF-SATPlan* and *BF**, that can flexibly compose web services using a succinct data structure and fast planning technique.

## 1 Introduction

*Web services* [1] are a piece of XML-based software interface that can be invoked over the Internet, and can be roughly viewed as a next-generation successor of CORBA or RPC technique. Currently, web services are often considered as one of the most vital building blocks to fully achieve the vision of the "Semantic Web." In short, web services have a collection of XML-based functions API with input/output parameters. To retrieve specific answers (i.e., output parameters), therefore, one has to find a right web service (or a set of web services combined), and invoke a right function with needed input parameters. Consider the following motivating example.

**(Example 1.)** When a web service, $w$, is invoked with input parameters, $w_{in}$, it returns the output parameters, $w_{out}$. In general, "all" input parameters in $w_{in}$ must be provided (i.e., input parameters are mandatory). For instance, consider the following web service, `findResturant`, (for simplicity, let us assume that all types are string):

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

This web service can be captured as: $w_{in}$ = *{zip, foodPref, streetAddress}* and $w_{out}$ = *{name, phone}*. Suppose a user, $u$ , is looking for the name of "Thai" restaurant near a hotel. First, if $u$ knows the zip code, $u$ can invoke $w$ by providing three necessary input parameters *{16801, Thai, 243blue}*. Second, if $u$ does not know the zip code of the hotel, then $u$ first has to obtain the zip code by invoking other web service, say `findZipcode`, that may return the zip code when a street address is given. That is, $u$ has to invoke `findZipcode`, followed by $w$. We refer to the first case as "discovery" and the second case as "composition." The main problem that we consider in this paper is how to quickly "compose" web services when thousands of them are available. ■

As the usage of web services proliferates dramatically, however, the number of available web services increases as well, and finding the right web services becomes a non-trivial task. Therefore, in this paper, we study about "*how to quickly find or compose the right web services among a large number of candidate web services (in thousands and more) to choose from?*." Toward the problem, we make three contributions in this paper: (1) a succinct data structure using Bloom Filter [2] to facilitate parameter matching between web services; (2) SAT based exhaustive search algorithm for small and static case (BF-SATPlan); and (3) A* based greedy search algorithm for large and dynamic case (BF*). The overview of our approaches is illustrated in Fig. 1.

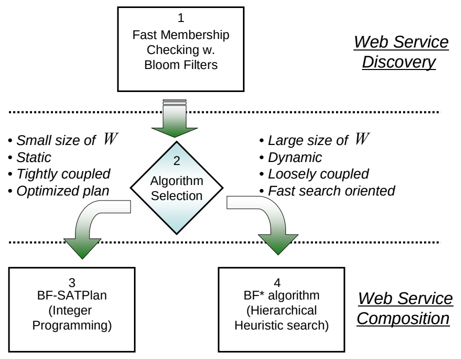

**Fig. 1.** Overview of our approaches.

## 2 Overview of Main Results

A set of web services is denoted by $W$, and the request of a client (e.g., software agent) is denoted by $r$. Both $\forall w \in W$ and $r$ have input and out parameters. When a client $r$ can invoke a web service $w$, since $u$ has to provide all input parameters of w, $r_{in} \supseteq w_{in}$ is the necessary condition for the invocation (Note that we ignore the data type of parameters in this paper). That is, all necessary input parameters of $w$ must be readily available to $r$ to begin with. Therefore, to quickly check if a web service $r$ can invoke another one $w$, one needs to quickly compute the "membership" (e.g.,

$r_{in} \supseteq w_{in}$). For this issue, we propose an efficient hash-based indexing structure using Bloom Filters (Section 3).

When a thousand of web services are available, and a single web service cannot satisfy the given request alone, then one has to "combine" multiple ones. Then, it is possible to have many "solutions" that correctly satisfy the given request while being very different in terms of their performances. For instance, to satisfy a request *r*, one solution may need only two web services, $w_1$ and $w_2$. On the other hand, another solution may require to invoke ten web services, $w_1, \ldots, w_{10}$. The upper-bound of all possible combinations of *n* web services invocation is its power set, $2^n$, which is prohibitively expensive to enumerate and evaluate one by one. Toward this issue, we propose to convert the web service composition problem into the propositional satisfiability problem (SAT). Since it is known that graph planning can be efficiently solved as SAT, then we can find the optimal solution for the given request using one of the developed techniques for SAT problem (e.g., integer linear programming). Our proposed solution, named as **BF-SATPlan**, is thus suitable for a small to medium sized web services (e.g., 285 nodes) and the status of web services community is static and tightly coupled (Section 4).

Lastly, among all possible correct "solutions", the optimal solution may not be always needed. For such a case, we propose a search algorithm, named as **BF***, that finds a sub-optimal solution – very quickly (Section 5). Using the competitive A* algorithm popular in many AI search problems, we show how to capture the web service composition problem into the graph search problem, and how to encode heuristics needed for A* algorithm. When a number of web services are huge (thus finding the optimal solution is infeasible) or the status of web services community is dynamic (thus web services may join or leave at any time), then this BF* algorithm is suitable.

In the following three sections, our major contributions are discussed in detail.

## 3 Fast Membership Checking with Bloom Filters

A *Bloom Filter* [2] is a simple space efficient randomized data structure for representing a set in order to support membership queries efficiently. Since it is based on a myriad of hash functions, it takes *O*(1) to check the membership, and its space efficiency is achieved at the small cost of errors. In the context of our problem, the idea is the following. For a web service, *w*, with $w_{in} = \{I^1, \ldots, I^P\}$ and $w_{out} = \{O^1, \ldots, O^Q\}$, two corresponding bloom filers are prepared, $BF_{in}^w$ and $BF_{out}^w$, respectively, where the bloom filter is a vector of *m* bits initialized to 0. Further, *k* independent hash functions, $H_1, \ldots, H_k$, are given, each with the output range $\{1, \ldots, m\}$, in sync with *m* bits of bloom filters. Then, each parameter $I^i \in w_{in}$ (resp. $O^j \in w_{out}$) is fed to *k* hash functions, and each bit at positions $H_1(I^i), \ldots, H_k(I^i)$ in $BF_{in}^w$ (resp. $H_1(O^j), \ldots, H_k(O^j)$ in $BF_{out}^w$) is set 1.

If the bits at some positions were set to 1 by previous hash functions (due to hash collision), do nothing. Once all parameters in $w_{in}$ and $w_{out}$ are processed this way, the bloom filters, $BF_{in}^{w}$ and $BF_{out}^{w}$, become a succinct in representation of potentially long list of input and output parameters of a web service. To check the membership if $X \in w_{in}$, one checks the bits at positions $H_1(X),\ldots,H_k(X)$ in $BF_{in}^{w}$. If any of them is 0, then one concludes that $X \notin w_{in}$ for sure. Otherwise, one concludes that $X \in w_{in}$ with a small probability of being false. To merge two bloom filters, simple OR of two is sufficient. The salient feature of bloom filter is that one can control the probability of false positive by adjusting *m*, *k*, *p*, and *q* — the precise probability is known as [3]: $\left(1-\left(1-\frac{1}{m}\right)^{k(p+q)}\right)^{k} \approx (1-e^{k(p+q)/m})^{k}$. For instance, with 5 hash functions (i.e., $k$ =5), 10 bits in bloom filter, and upto 100 input/output parameters (i.e., $\frac{(p+q)}{m}$ = 10), the probability of errors becomes mere 0.00943.

## 4 BF-SATPlan

Our proposal, named as BF-SATPlan, is principally derived from the SAT encodings proposed in [4], which showed that planning problems can be efficiently solved by general propositional satisfiability algorithms. However, the satisfiability problems typically work in the so-called "*closed world assumption"* (i.e., all relevant facts are known). However, in our setting, not all facts about web services are known in the beginning. That is, initially, one may only know a name list of web services (maybe from some UDDI-based directory system), but do not necessarily know which one can serve what requests, etc. To solve this issue, BF-SATPlan consists of two steps: (1) build a regression graph out of all web services; and (2) encode the planning problem as the SAT problem on the regression graph, and solve it using *integer linear programming* (ILP) technique.

### 4.1 Web Services Matching as Propositional Logic

Checking if a web service can invoke another web service or not can be captured as propositional logic statements. A *pre-condition* of $w$, $Pre_w$, specifies the state of the world needed for a successful execution of $w$. Symmetrically, a *post-condition* of $w$, $Post_{x,}$ specifies facts that must be valid (true) after $w$ has finished its execution. Therefore, a web service specification can be expressed in a propositional logic: $Pre_w \Rightarrow Post_w$ . For a request $r$ with input parameters $r_{in}$ and output parameters $r_{out}$ , the following holds:

**Proposition 1** A web service $w$ can answer $r$ iff (($r_{in} \supseteq w_{in}$) $\Rightarrow$ $Pre_w$) and ( $Post_w$ $\Rightarrow$ ($r_{out} \subseteq$ ($r_{in} \cup w_{out}$ )) □

That is, if $r_{in} \supseteq w_{in}$, then since all required input parameters of $w$ can be provided, $r$ can invoke $w$, satisfying the pre-condition of $w$. Since $Pre_w$ implies $Post_w$, ($r_{in} \cup w_{out}$ ) is valid and what is desired in $r_{out}$ is a subset of ($r_{in} \cup w_{out}$ ), reaching the goal.

Next, consider a case when multiple web services must be "combined" or "composed" to satisfy the request. Given a request $r$ and two web services $x$ and $y$, for instance, suppose one can invoke $x$ using inputs in $r_{in}$, but the output of $x$ does not have what we look for in $r_{out}$: (( $r_{in} \supseteq x_{in}$ )$\Rightarrow$ $Pre_x$) $\wedge$ $\neg$( $Post_x$ $\Rightarrow$ ( $r_{out}$ $\subseteq$ ($r_{in} \cup w_{out}$ ) )). Symmetrically, the output of $y$ generates what we look for in $r_{out}$, but one cannot invoke $y$ directly since it expects inputs not in $r_{in}$ : $\neg$(( $r_{in} \supseteq y_{in}$ )$\Rightarrow$ $Pre_y$) $\wedge$ ( $Post_y$ $\Rightarrow$ ($r_{out} \subseteq$ ( $r_{in} \cup y_{out}$ ) ). Furthermore, using initial inputs of $r_{in}$ and the outputs of $x$, one can invoke $y$ : ( ($r_{in} \cup x_{out}$ ) $\supseteq y_{in}$ ) $\Rightarrow$ $Pre_y$. Then, the request $r$ can be satisfied by the logical flow: (( $r_{in} \supseteq x_{in}$ )$\Rightarrow$ $Pre_x$) $\wedge$ ( $Post_x$ $\Rightarrow$ $Pre_y$) $\wedge$ ( $Post_y$ $\Rightarrow$ ( $r_{out}$ $\subseteq$ ( $r_{in} \cup w_{out} \cup y_{out}$ ) ). Based on this observation follows the following Lemma (proof is omitted for the interest of space):

**Lemma 1** A chain of web services, $w^1 \Rightarrow \ldots \Rightarrow w^n$, can "jointly" answer $r$, iff (( $r_{in} \supseteq w^1_{in}$**)** $\Rightarrow \mathrm{Pr}\,e_{w^1}$ ) $\wedge$ ( $Post_{w^1} \Rightarrow \mathrm{Pr}\,e_{w^2}$ ) $\wedge \ldots \wedge$ ( $Post_{w^{n-1}} \Rightarrow \mathrm{Pr}\,e_{w^n}$ ) $\wedge$ ( $Post_{w^n}$ $\Rightarrow$ ( $r_{out} \subseteq (r_{in} \cup w^1_{out} \cup \ldots \cup w^n_{out})$ ))□

## 4.2 Regression Graph

Imagine message (as shown in Example 1) are interchanged among web services based agents. Each agent will have a set of agents that they can invoke, and in turn a set of agents who can invoke them. In this step, we aim at drawing such a "map" that shows these invocation relationships.

**Table 1.** Definitions of operators for Regression Graph ($m$' and $m$ are parameter sets).

| Operator | Definition |
|---|---|
| $\oplus_{m'} = m$ | $m$' fully matches $m$ iff $m' \supseteq m$ **.** |
| ( $\Diamond_{m'} = m$ ) | $m$' partially matches $m$ iff $((m' \cap m) \neq \phi) \wedge \neg(\oplus_{m'} = m)$ **.** |
| ( $\otimes_c = m$ ) | A set of parameters $c = \{m^1, m^2, \ldots, m^n\}$ conjunctively match $m$ iff ( $\neg\exists\ m^i \in c$ : $\oplus_{m^i} = m$ **)** $\wedge$ ( $\exists\ c$ **:** $\bigcup_{i=1}^{n} m^i \supseteq m$ , $\Diamond_{m^i} = m$ ) where $n \geq 2$ . |

Table 1 illustrates three "relationship" (i.e., operators) that we use in constructing the regression graph using the notions introduced in Section 4.2.

For example, $\Diamond_{r_{in}} = w_{in}$ means a request *r* only partially matches a web service *w*, and if $c = \bigcup_{i=1}^{n} w_{out}^{i} \cup r_{in}$ and $\otimes_{c} = r_{out}$, then $(r_{in} \cap r_{out} \neq \phi) \wedge (\otimes_{\bigcup_{i=1}^{n} w_{out}^{i}} = r_{out} \setminus r_{in})$. Based on these concepts, the *BF-Regression Graph* is denoted by 6-tuple $\langle W_G, C_G, r_{in}, r_{out}, E_G, D_G \rangle$ where (1) $W_G$ is a collection of nodes made of web services $w \in W$ ; (2) $C_G$ is a collection of conjunctive nodes; (3) the edge set $E_G \subseteq (W_G \times C) \cup (W_G \times r_{in}) \cup (W_G \times r_{out}) \cup (C \times r_{in}) \cup (C \times r_{out})$; (4) $D_G \subseteq W_G$ is a set of "dead" nodes such that $\neg\exists\ w' \in W : (\oplus_{w'_{out}} = w_{in}) \vee (\oplus_{r_{in}} = w_{in}) \vee (\Diamond_{w'_{out}} = w_{in}) \vee (\Diamond_{r_{in}} = w_{in})$; and (5) $r_{in}$ and $r_{out}$ are the initial and goal nodes, respectively. Initially, $W_G = C_G = E_G = D_G = \phi$. **We first define a regression process.**

**Regression process:** for each $w \in W$ ,

Step-1: If $w$ is the dead node, update $D_G$ by $D_G = D_G \cup \{w\}$. Stop.

Step-2: If $\oplus_{r_{in}} = w_{out}$ , a directed arc incident from $w$ to $r_{in}$ , that is, $e_{w \to r_{in}}$ is generated and $E_G = E_G \cup \{e_{w \to r_{in}}\}$. Stop.

Step-3: For $\forall w' \in W$ satisfying $\oplus_{w'_{out}} = w_{in}$ , a directed arc, $e_{w \to w'}$ is generated to connect $w$ with $w'$. $W_G = W_G \cup \{w'\}$ . $E_G = E_G \cup \{e_{w \to w'}\}$ .

Step-4: For $\forall c \subseteq W$ such that $\otimes_{c} = (w_{in} \setminus r_{in})$, a conjunction node $c_g$ is generated and a directed arc incident from $w$ to $c_g$ , $e_{w \to c_g}$ is also generated. $C_G = C_G \cup \{c_g\}$ . $E_G = E_G \cup \{e_{w \to c_g}\}$ . Then, for $\forall w^{i} \in c, i = 1,2,..,n$ , $W_G = W_G \cup \{w^{i}\}$. Finally, directed arcs, $e_{c_g \to w^{i}}$ are generated to link $c_g$ with $\forall w^{i} \in c$ and $E_G = E_G \cup \{e_{c_g \to w^{i}}\}$ .

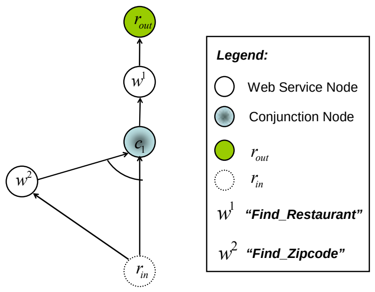

**Fig. 2.** BF-Regression Graph for Example 1

**Procedure 1** (Extracting BF-Regression graph from $\langle W, r_{out}, r_{in} \rangle$)

The graph is obtained by iterating the Regression process on $\langle W, r_{out}, r_{in} \rangle$. Note that any frequently occurring matching process (i.e., $\oplus$, $\Diamond$ and $\otimes$) pertinent to the Regression process, will be efficiently handled by Bloom filter technique in Section 3. The procedure follows three steps:

1. Find a set $s = \{w \in W \mid \oplus_{w_{out}} = r_{out}\}$ and $t = \{c \subseteq W \mid \otimes_c = r_{out}\}$. If $s \neq \phi$ $\vee$ $t \neq \phi$, proceed to next step. Otherwise, no plan exists.
2. Regard $r_{out}$ as a web service and apply the Regression process.
3. Apply the Regression process on $\forall w \in W_G$ until ( $w \in D_G$ ) $\vee$ ( $\oplus_{r_{in}} = w_{in}$ ).

If applying the Procedure 1 to Example 1, we have the regression graph in Fig. 2. Now, we define two preliminary notions for second procedure: (1) $Dj_l \subseteq W_G \bigcup C_G \bigcup r_{in}$ and (2) $Cj_{c_g} \subseteq W_G \bigcup r_{in}$, where $\forall l \in W_G \bigcup r_{out}$ and $\forall c_g \in C_G$.

- Let $s = \{k \in W_G \bigcup C_G \bigcup r_{in} \mid \oplus_k = l\}$. If $s \neq \phi$, $l$ has $Dj_l = s$. Each element of $Dj_l$ satisfy $\Pr e_l$ disjointly.
- $c_g$ has $Cj_{c_g}$, that is, a set of child nodes which satisfy $\Pr e_{c_g}$ conjunctively.

### 4.3 Satisfiability Problem as Integer Linear Programming

**Procedure 2** (Converting Regression BF graph $\langle W_G, C_G, r_{in}, r_{out}, E_G, D_G \rangle$ to ILP)

This procedure is motivated by the well known SATPlan encodings [4]. In SATPlan, a satisfiability (SAT) problem can be expressed as integer linear program (ILP) by converting the CNF clauses in SAT problem to 0-1 linear inequalities [12]. For instance, $x_1 \vee x_2 \vee \neg x_3$ **can be re-written as** $x_1 + x_2 + (1 - x_3) \geq 1; x_1, x_2, x_3 \in \{0,1\}$.

The resulting ILP formulation of Regression BF graph is summarized as follows:

**Variables:** For all nodes belonging to the Regression graph, we have Boolean variables: $w^i$, $c^i$, $r_{in}$ and $r_{out}$ = 0 or 1. Where $\forall w^i$ (resp. $\forall c^i$) comes from $W_G$ (resp. $C_G$). $r_{in}$ (resp. $r_{out}$) corresponds to the node of $r_{in}$ (resp. $r_{out}$).

**Constraints:** Initially, $w^i$ =0 if $w^i$ comes from $D_G$ but $r_{in}$ =1 and $r_{out}$ =1 because a request $r$ is assumed to be given. Other constraints are as follows.

- Conjunctive constraints: $c^i \leq k$ , $\forall k \in Cj_{c^i}$.
- Disjunctive constraints: $l \leq \sum_{i=1}^{n} k_i$ , $\forall k_i \in Dj_l$, if $Dj_l \neq \phi$.

**Objective Function:** The objective function can be set to multiple criteria (e.g., price, duration or reliability). Accordingly, the ILP can compose the service composite to optimize the selected objective function.

If applying the Procedure 2, we can formulate example-1 as an ILP such a way as to minimize the size of a composite web service as following compact form:

$$\min\{\boldsymbol{B}\boldsymbol{x}^T : \boldsymbol{M}\boldsymbol{x}^T \geq \boldsymbol{0}, \boldsymbol{r}_{in} = 1, \boldsymbol{r}_{out} = 1, \boldsymbol{x} \setminus \{\boldsymbol{r}_{in}, \boldsymbol{r}_{out}\} \in \{0,1\}^3\} \quad (1)$$

Where $\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 \end{bmatrix}$, $\boldsymbol{B} = [1,1,0,0,0]$, $\boldsymbol{x} = [\boldsymbol{w}^1, \boldsymbol{w}^2, \boldsymbol{c}_1, \boldsymbol{r}_{in}, \boldsymbol{r}_{out}]$, and $\boldsymbol{0}$ is that all zeros vector. It is clear to obtain the solution of (1). Since there is only one tree to connect $\boldsymbol{r}_{in}$ and $\boldsymbol{r}_{out}$, hence $\boldsymbol{x} = [1,1,1,1,1]$ and $\boldsymbol{B}\boldsymbol{x}^T = 2$.

It should be noted that the increasing size of the BF-Regression graph in procedure 1 has significant negative impact on the computational time for solving ILP formulated in procedure 2. In general, the computational complexity for binary integer program is known as NP-complete, that is, $O(2^n mn)$ by a brute-force enumerative algorithm where $n$ is the number of Boolean variables and $m$ is the number of constraints. In our problem, the number of the BF-Regression graph corresponds to $n$ and number of constraints $m$ increase in proportion to $n$. However, some algorithms including Branch & Bound offer much better average performance than complete enumeration. In [12], an ILP with 285 nodes was solved in 1400 seconds using their suggested novel method. Nonetheless, we suggest to limit the application of BF-SATPlan to the small size of search space because the computational time increases exponentially as the number of nodes increases.

## 5 BF*: A* Based Graph Search Algorithm

In our previous work [6], we presented a preliminary idea of BF* and stratified flooding algorithms for dynamic web service composition problem, Here, we elaborate them with new analysis result.

### 5.1 Stratified Flooding Algorithm

Lemma 1 readily suggests a naive fixedpoint algorithm for solving the case as shown in Algorithm 1. Let Ω (resp. Σ) be a set of web services that have been visited so far (resp. a set of parameters gathered so far). At each iteration, new set of web services, $\delta$, are found that can be invoked using Σ. Since there are only finite number of web services, $W$, and each iteration adds only "new" set of web services (i.e., $w \notin \Omega$), the iterations must end. At some point, if $\Sigma \supseteq r_{out}$, then it means that using the parameters gathered so far (i.e., Σ), one can get the desired output parameters in $r_{out}$, thus solving the case of Lemma 1. The naive stratified flooding algorithm in Algorithm 1 is simple but inefficient since it finds “all” web services (i.e., flooding) that can be invoked, and accumulate them into Ω.

**Algorithm 1:** Stratified Flooding Algorithm

```
let $W \leftarrow$ all web services, $\Omega \leftarrow \phi$ and $\Sigma \leftarrow r_{in}$;
print $r_{in}$, “$\Rightarrow$”;
while $\neg(\Sigma \supseteq r_{out})$ do
$\delta \leftarrow \{w \mid w \in W, w \notin \Omega, w_{in} \subseteq r_{in} \cup \Sigma\}$;
    $\Omega \leftarrow \Omega \cup \delta$;
    $\Sigma \leftarrow \Sigma \cup (\bigcup_{w\in\delta} w_{out})$;
    print $\delta$,"$\Rightarrow$";
print $r_{out}$;
```

**Algorithm 2:** BF* Algorithm

```
let $W \leftarrow$ all web services, $\Omega \leftarrow \phi$ and $\Sigma \leftarrow r_{in}$;
print $r_{in}$, “$\Rightarrow$”;
while $\neg(\Sigma \supseteq r_{out})$ do
$\delta \leftarrow \{w \mid w \in W, w \notin \Omega, w_{in} \subseteq r_{in} \cup \Sigma\}$;
    $w^{\min} \leftarrow w$ with MIN($f(w)$);
    $\Omega \leftarrow \Omega \cup w^{\min}$;
    $\Sigma \leftarrow \Sigma \cup w_{out}^{\min}$;
    print $w^{\min}$,"$\Rightarrow$";
print $r_{out}$;
```

## 5.2 BF* Graph Search Algorithm

A* algorithm is a heuristics-based competitive search algorithm. At each state, it considers some heuristics-based cost to pick the next state with the lowest cost. BF* algorithm based on the A* algorithm is illustrated in Algorithm 2. In our context, A* algorithm can be captured as follows. Given a set of candidate web services to visit next, $N$ ($\notin \Omega$), one chooses $n(\in N)$ with the “smallest” **f(*n*) (=h(*n*) + g(*n*))** such that:

$$\mathbf{h}(n) = 1/(|(r_{out} \setminus \Sigma) \cap n_{out}|), \quad \mathbf{g}(n) = |\Omega|$$

That is, the remaining parameters of $r_{out}$ that are yet to be found are $r_{out} \setminus \Sigma$. Then, the intersection of this and $n_{out}$ is a set of parameters that $n$ helps to find. The more parameters $n$ finds, the bigger contribution $n$ makes to reach to the goal. Therefore, A* favors the $n$ whose contribution to find remaining parameters is the max (i.e., h($n$) is the smallest). In order words, our heuristics is based on the hypothesis that “*visiting a web service with bigger contribution would find the goal faster than otherwise.*”

The missing operational semantics of BF* algorithm is similar to that of A* algorithm (e.g., using OPEN and CLOSED priority queues or normalizing h($n$) and g($n$) properly before addition). Note that more details may be found in [6].

## 5.3 Numerical analysis

We denote web service cluster by Θ that contains web services with similar functions, thus $\Theta = \bigcup_{i=1}^{n} w^i$. Similarly, we assume that, $\Theta_{in} = \bigcup_{i=1}^{n} w_{in}^i$ and $\Theta_{out} = \bigcup_{i=1}^{n} w_{out}^i$.
For the analysis purpose, we define *unit accordance rate* $UA : W \to \Re$ as follows:

$$UA(w) = |\Theta_{out} \cap w_{out}| / |\Theta_{out}|, \qquad \text{where } w \in \Theta$$

Now, we can consider an example where we need to create one feasible itinerary by composing web services selected from five web service clusters, {Airway, Rent-Car, Itinerary, Hotel, Insurance}. In this example, we assume that for $\forall w^i \in \Theta$, $UA(w^i)$ is distributed with fixed interval over from 0 to $\max_{w^i \in \Theta} UA(w^i)$. In addition, we assume that if there is a combination $C \subseteq \Theta$ such that $\sum_{w^i \in C} UA(w^i) \geq 1$ and $C_{out} = \bigcup_{w^i \in C} w^i_{out}$, then $\otimes_{c_{out}} = \Theta_{out}$. For example, if $|\Theta_i| = 4$ and $\max_{w^i \in \Theta} UA(w^i)$ =0.5, then we can have four web services whose $UA(w^i)$ are 0.5, 0.375, 0.25 and 0.125, respectively. Furthermore, if we gather the first three web services, we will then obtain $\Theta_{out}$ because (0.5+0.375+0.25) > 1 according to the assumption. Finally, we assume that $|\Theta_i|$ is the same, $i = 1,2,...,5$.

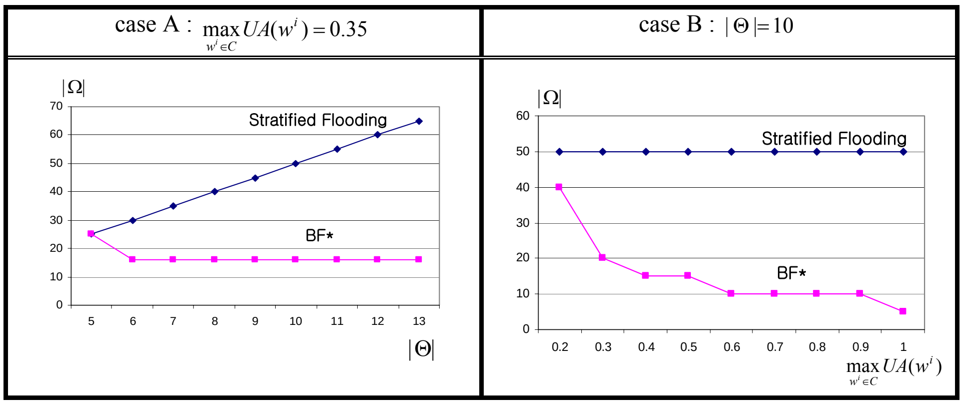

**Fig. 5.** Comparison two algorithms over case A and B

We can get two numerical simulation results in Fig. 5. | Ω| is important to understand how efficient an algorithm is because it represents the number of web services visited. The better an algorithm is, the smaller | Ω| becomes. In Fig. 5, BF* shows better performance against the striated flooding algorithm – as $|\Theta|$ increases and maximum unit accordance rate of $\Theta$ goes close to 1. It is reasonable because BF* reduces | Ω| by forcing not to visit a web service ***n'*** with small **h(*n'*)** after it achieve $\Theta_{out}$.

The main assumption underlying this simulation is that web services would be clustered according to their similarity in terms of input and output parameters and even important is that there would be a few dominant web services reflecting each cluster. Through this simulation, we showed that in particular, BF* has good performance under the assumption. In practice, this assumption is considered valid in the sense that mainly, one or two competitive firms provide the professional web services enough to dominate the corresponding market. As a future work, we plan to perform more extensive simulation analysis by changing the distribution of $UA(w^i)$.

## 6 Related Work

STRIPS [7] is the first major AI planning system that was designed in 1971. The STRIPS language describes actions in terms of their preconditions and effects and describes the initial and goal states as conjunctions of positive literals. Partial order planning (POP) algorithms that dominated the field till mid 90s, explore the space of plans without committing to a totally ordered sequence of actions. They work back from the goal, adding actions to the plan to achieve each sub-goal. GRAPHPLAN [8] is based on processing planning graphs by using a backward search to extract a plan and allowing for some partial ordering among actions. A planning graph can be constructed incrementally, starting from the initial state. Each layer contains a superset of all the literals or actions that could occur at that time step and encodes mutual exclusion, relations among literals or actions that cannot co-occur. Planning graphs yield also useful heuristics for state-space and partial order planners.

[9] introduced the planning as satisfiability approach and the SATplan algorithm which is inspired by the success of the greedy local search for satisfiability 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. [5] is an excellent survey of modern planning, concentrating on, respectively, partial order planning, and GRAPHPLAN and SATplan.

**Table 3.** Comparison of three web service composition approaches.

| Item | **METEOR-S** | **Proteus** | ***Our Approach*** |
|---|---|---|---|
| Planning | Manual composing a template of plans (semantic matching) | Manual composing a template of plans (semantic matching) | Automatic through AI techniques (syntactic matching) |
| Allocation | Allocating web services in design stage | Auto-allocating web services while executing | Planning and allocating occur at the same time |
| Execution | BPEL execution | Automatic | N/A |
| Misc. | Allocation satisfying constraints (e.g., minimum service cost among alternatives) | Allocation based on scheduling policies (e.g., Least response time) and SQL user query | Presenting two algorithms for both tightly and loosely coupled business environment |

As dynamic discovery of web services and composition problem, run-time adaptability of a composed process is another research issue in this area. METEOR-S [10] project has addressed this issue for workflows. This allows the process designers to bind web services to an abstract process, based on business and process constraints and generate an executable process. Proteus [13] was suggested as a framework that consumes a user request to compose a plan that incorporates available web services, and execute the plan seamless. We compare these two approaches with our suggestion as in table 3.

## 7 Conclusion

We have presented two algorithms, *BF-SATPlan* and *BF*,* that are suitable to discover and compose a large number of web services under static and dynamic environments. However, there remain several future research directions. First, current work only handles syntactic matches between web services. To support semantic matches (e.g., "Lastname" for "Familyname"), more ingenious methods (e.g., Ontology matching under Semantic web) need to be developed. Second, once web service composition is done, in practice, one may still have to "negotiate" or "auction" with agents to execute plans. This requires the support for loops [11]. Finally, we are currently implementing the proposed algorithms, and plan to do extensive experiments afterward.

## References

1. W3C Web Services Activity (web site), http://www.w3c.org/2002/ws/
2. Bloom. B.: Space/Time Tradeoffs in Hash Coding with Allowable Errors. Comm. ACM, 13(7) (1970) 422–426
3. Fan, L., Cao, P., Almeida, J., Broder, A. Z.:Summary Cache: A Scalable WideArea Web Cache Charing Protocol. IEEE/ACM Trans. on Networking, 8(3) (2000) 281–293
4. Kautz, H et al.: Encoding plans in propositional logic. Proc. KR (1996)
5. Weld, D. S.: Recent advances in AI planning. AI Magazine, Vol. 20(2) (1999)
6. Oh, S., On, B., Larson, E. J., Lee, D.: BF*: Web Services Discovery and Composition as Graph Search Problem. IEEE Int. Conf. on e-Technology, e-Commerce and e-Service (EEE), Hong Kong, China, March (2005)
7. Fikes, R.E., Nilsson, H. J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, Vol. 2 (1971) 189-205
8. Furst, M., Blum, A. L.: Fast planning through planning graph analysis. Artificial Intelligence, Vol. 90 (1997) 281-300
9. Kautz, H., Selman, B.: Blackbox: A new approach to the application of theorem proving to problem solving. Technical report (1998)
10. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. J. Information Technology and Management (2004)
11. McDermott, D.: Estimated-regression planning for interactions with Web Services. 6th Int. Conf. on AI Planning and Scheduling. AAAI Press (2002)
12. Vossen, T., Ball, M., Lotem, A., Nau, D.: On the use of integer programming models in AI planning. Sixteenth National Conference on Artificial Intelligence (1999)
13. Ghandeharizadeh et al.: Proteus: A System for Dynamically Composing and Intelligently Executing Web Services. Proc. ICWS, Las Vegas, Nevada, June (2003)