Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 783476, 21 pages
doi:10.1155/2012/783476

Research Article

A Cost-Effective Planning Graph Approach for Large-Scale Web Service Composition

Szu-Yin Lin,[1] Guan-Ting Lin,[1] Kuo-Ming Chao,[2] and Chi-Chun Lo[1]

[1] *Institute of Information Management, National Chiao Tung University, Hsin-Chu 300, Taiwan*
[2] *Faculty of Computing and Engineering, Coventry University, Coventry CV1 5FB, UK*

Correspondence should be addressed to Szu-Yin Lin, szuyinlin@gmail.com

Received 16 November 2011; Revised 25 January 2012; Accepted 28 January 2012

Academic Editor: Jung-Fa Tsai

Web Service Composition (WSC) problems can be considered as a service matching problem, which means that the output parameters of a Web service can be used as inputs of another one. However, when a very large number of Web services are deployed in the environment, the service composition has become sophisticated and complicated process. In this study, we proposed a novel cost-effective Web service composition mechanism. It utilizes planning graph based on backward search algorithm to find multiple feasible solutions and recommends a best composition solution according to the lowest service cost. In other words, the proposed approach is a goal-driven mechanism, which can recommend the approximate solutions, but it consumes fewer amounts of Web services and less nested levels of composite service. Finally, we implement a simulation platform to validate the proposed cost-effective planning graph mechanism in large-scale Web services environment. The simulation results show that our proposed algorithm based on the backward planning graph has reduced by 94% service cost in three different environments of service composition that is compared with other existing service composition approaches which are based on a forward planning graph.

1. Introduction

Research on Web Service Composition (WSC) has become increasingly important in recent years due to the growing number of web services over the Internet and the challenge of automating the process. Particularly with the development of cloud computing, there will be more and more diverse Web services deployed and published on cloud environments. Web services are Internet-based software components which have the capabilities of delivering service cross-platforms and languages. The W3C organization has defined "Web Services" that "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically

Web Services Description Language WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [1]". The W3C has also pointed out that "We can identify two major classes of Web services, REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of stateless operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations [2]".

Since the growth of Web services to a large number is happening and possible interactions among them are huge, searching, analying, and processing them to find the required services to achieve user goals is very difficult via a manual process. This also means service composition problem has become increasingly sophisticated and complicated in the real world [3]. Therefore, the issue of finding solutions efficiently via composing services to form a complex composited service is one of the important studies.

The process of combining and linking existing Web services to create new service is known as WSC. In other words, WSC problems can be considered as a service matching problem, which means that the output parameters of a Web service can be used as inputs of another Web service. The aim of WSC is to provide a means for composing diverse Web services to accomplish user request which cannot be satisfied by a single Web service. The Web services composition approaches can be broadly classified as static or dynamic based on the process and the way of composing services. A static Web service composition is constructed to solve the particular problem through manually identifying Web services by their capabilities. They are composed by a series of known Web services and a set of known data in order to obtain the expected results. Dynamic Web service composition is to automatically select Web services and compose those at the execution/run time. The aim is to build and utilize an automated service discovery and its associated execution mechanism to produce the required composite services. There have been numerous methods proposed for solving the problem of service composition, such as workflow [4] and AI planning [5]. The Web service composition is commonly described by using the Web Services Business Process Execution Language (BPEL) [6] which is an XML-based language that provides particular functionalities for processes, such as define variables, create conditionals, design loops, and handle exception. It utilizes Web services as the model for the decomposition and the composition of the process. However, BPEL promotes the development of workflow and the integration of business processes.

Nowadays, numerous researches focus on finding and developing new approaches to fit in with WSC. The task of WSC usually assumed that the composition process generates a composition workflow, which starts from the known variables from the requirements or the related constraints to the expected goal. Therefore, many algorithms based on AI planning techniques that can facilitate to automat Web service composition have been proposed [3, 5, 7–9], but it is still a great challenge for solving large-scale WSC problem to obtain multiple flexible service composition solutions with acceptable service cost and execution time. It can assume that Web services as actions, and the process of composing them to produce the desired result as planning, so planning graph is one of the most suitable techniques could be used for WSC problem. However, there are very few studies using planning graph approach to achieve WSC problem, especially with considering both sides of cost and effectiveness in large-scale Web services composition.

Therefore, this paper proposes a new cost-effective planning graph approach based on backward strategy for large-scale Web service composition on cloud environment, which can find multiple solutions and recommend a list of best composite services composition to

users. In addition, we can recommend the approximate match services which may not totally meet to user requests, but the user may accept the services and it uses fewer amounts of Web services and less nested levels of composite Web services. The main research objectives in this paper are (1) to present a novel framework and composition processes for WSC on cloud environment, (2) to design a cost-effective WSC algorithm which can obtain multiple service composition solution using fewer number of Web services with low cost and in acceptable execution time, (3) the proposed approach must process large-scale Web services which amount over 10000, and (4) to provide an approximate solution when there is no composite solution which exactly corresponds to the request.

The rest of this paper is structured as follows. Section 2 describes the related works. Section 3 proposes the planning graph service composition algorithm based on the backward strategy. Section 3 presents the details of experiment and its results, and we give a summary discussion about the result. Finally, Section 4 concludes this study and proposes the future work.

2. Related Works

The planning graph, which is a representation technique by AI planning, provides a very powerful search technique in a space [10] to improve the efficiency of AI algorithms. A WSC problem can be modeled as a planning graph. The input parameters of the composition request are mapped to the initial state, and the output parameters of the composition request are mapped to goal propositions. If a planning graph reaches a proposition level which contains all required parameters, then it searches backward from the last level of the graph for a solution. However, the disadvantage of this approach is the difficulty of designing a strategy to trade off two key criteria that are cost and effectiveness. Therefore, there are always two problems of time consuming and redundant actions existed in the solution.

The planning graph is a layered graph whose edges are only allowed to connect two nodes from one layer to next layer. And the planning graph's layers are with an alternating sequence of action layer and proposition layer. The proposition layer contains a finite set of states, and the action layer contains a finite set of actions (the action has preconditions, negative effects, and positive effects). For example, the first layer of planning graph, P0, is a proposition layer which contains the initial states of the planning problem. The next layer, A1, is an action layer which contains a set of actions which preconditions can be satisfied by P0, and P1 is the union of the states of P0 and the effects of all A1's actions. Those preconditions of actions in A1 are connected to the state nodes in P0 by incoming arcs, and those positive or negative effects in P1 are connected to the state nodes in P1 by outgoing arcs. The process continues until it reaches the goal states or the fixed-point level of the graph.

The study conducted by [10] showed a Planning Graph planner technique called GRAPHPLAN. The GRAPHPLAN algorithm is operated in two main steps which alternate within a loop: graph expansion and solution extraction. The solution extraction can be formulated as a constraint solving problem [11] or as a search problem [12]. In Peer's survey [9], GRAPHPLAN's advantages include good performance, soundness, completeness, generation of shortest plans, and termination on unsolvable problems. However, the original GRAPHPLAN algorithm has some limitations: (1) its representation language is restricted to pure STRIPS operators, so no conditional or universally quantied effects are allowed; (2) the performance can decrease drastically if too much irrelevant information is contained in the specification of a planning task [13].

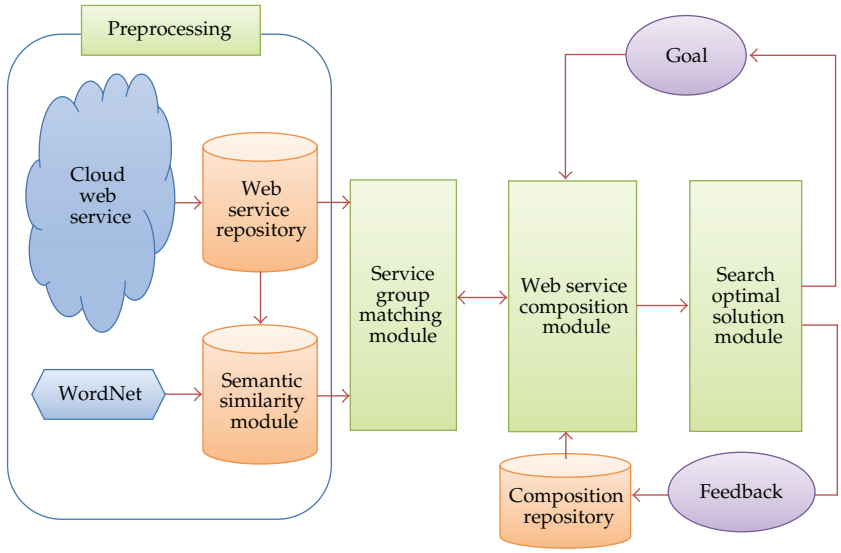

Figure 1: The flow diagram of the service composition mechanism.

Zheng and Yan [7] also transformed the problem of service composition into the problem of simplified planning graph based on forward search, which could be constructed in polynomial time. In classical AI planning technique, generating final solutions by a backward search is a popular approach, but it is the most time consuming technique. Researches have been working on it to improve it. However, forward search could improve efficiency, but the redundant Web services during the construction of the planning graph lead to the increase of service cost. Zheng and Yan [7] put efforts into using forward search in planning graph algorithm to solve WSC problem, and it shows a good result which can find a solution in polynomial time but encounters some drawbacks: (1) there are many redundant Web services existed in the solution of service composition, and (2) it is lack of flexible search mechanism which can recommend multiple solutions for service composition when few input unknown parameters occur. In other words, the composition algorithm based on the forward strategy aims to minimize the search time, but there are many possible redundant and unnecessary Web services included in the final solution.

3. Backward Planning Graph Approach for Web Services Composition

In this section, we will introduce the proposed service composition mechanism which includes four steps, such as preprocessing, service group matching, service composition, and search optimal solution. Those modules will be described in following subsections.

The overview and its flow diagram of the proposed service composition mechanism are shown in Figure 1, which contains the following four main processes and modules

(I) Preprocessing. There are two components involved in the first step. One is the Web service repository, and the other is semantic similarity module. The Web Service Repository will search Web services from distributed UDDIs on cloud and store those services in a repository database and entries in the repository will be updated regularly. Therefore, the input of this mechanism is Cloud Web Services. Semantic

Similarity Module precalculates the semantic similarity values between any two concepts and stores the similarity values in a semantic similarity database for retrieval.

(II) Service group matching module. It utilizes Web Service Repository and Semantic Similarity Module to select the Web services that can satisfy the query, and group them based on the degree of their similarity. It will provide a set of service groups for service composition.

(III) Web Service Composition Module. It will query Service Group Matching Module to get services which are required by composition algorithm. Web Service Composition Module will generate multiple service composition solutions according to the goal which is described in final output parameters of each solution. With the expansion of levels in backward planning graph composition algorithm, the goal will be refined at each iteration.

(IV) Search Optimal Solution Module: It will calculate the score of each solution and choose the most suitable solution of WSC from these identified solutions according to the given goal.

3.1. Preprocessing

In large-scale WSC, the number of querying services could be large. Querying Web service entries registered in distributed UDDIs at runtime in process of service composition, the efficiency is likely lower than those entries stored in one centralized database. So, all Web service entries to be used in this approach will be stored to a centralized structured Web Service Repository. In addition, the calculation of semantic similarity between concepts is a time consuming task which is not efficient to meet dynamic service composition, so it will be preprocessed by Semantic Similarity Module. Service Group Matching Model according to the repository and the relationships of concept similarity to respond the query. The preprocessing task requires two components.

(I) Web Service Repository

The aim of service repository is to virtualize service discovery. We query Web services registered in distributed UDDIs on cloud and parse the WSDL of Web services to store them in a repository database, as shown in Figure 2. It will search regularly Web services from distributed UDDIs and analyze the structure to update database. The structure is to facilitate the process of service composition by including a set of input parameters, output parameters, and their associated service name.

(II) Semantic Similarity Module

The semantic module is for discovering the relationship between Web services. According to the definition of lexicon and classification on WordNet, it transforms the description of services into the concept and relationship of Ontology, as shown in Figure 3. The similarities between concepts can be calculated based on their semantic similarity. Through these functions, obtaining semantic similarity values between semantic concepts become possible. The values of similarity will fall between 0 and 1, and the higher value represents higher similarity. Those similarity values are precalculated and stored in a database. However, this

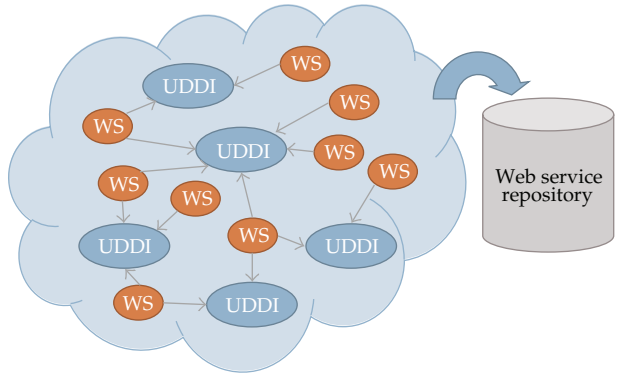

Figure 2: Capture web services into repository from cloud.

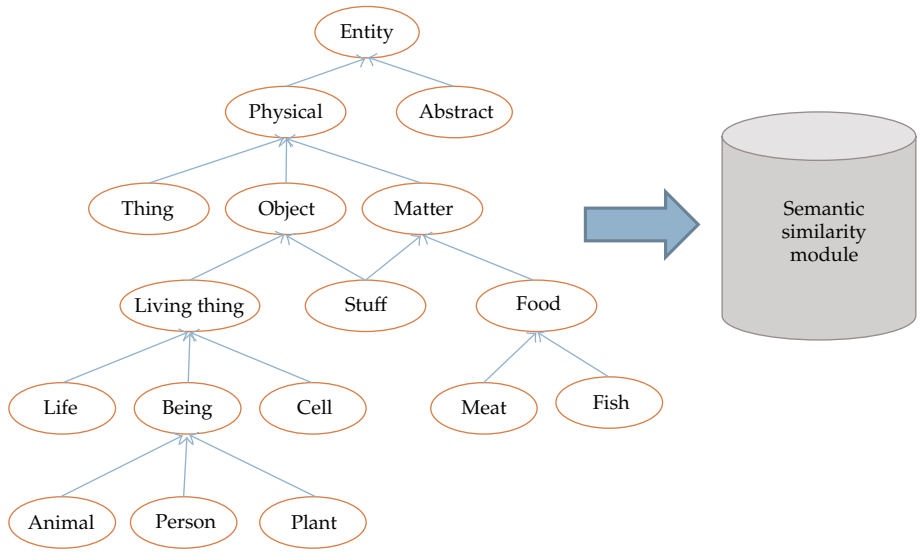

Figure 3: Import semantic concepts from WordNet.

module is useful but optional in the proposed service composition approach, so that it is not introduced to experiments in Section 4.

3.2. Service Group Matching Module

There will be full of many similar Web services on cloud, due to rapid expansion of solution space, so we could group similar Web services together based on their semantics as a service group. "Service group" is a concept that we proposed in this algorithm, which means that a group of Web services have certain degree of similarity in their input parameters and output parameters. This module will provide appropriate extracted service groups according to system requests or user queries for service composition module. It will be the key to reduce the amount of system queries and calculation. Service group matching algorithm includes three steps.

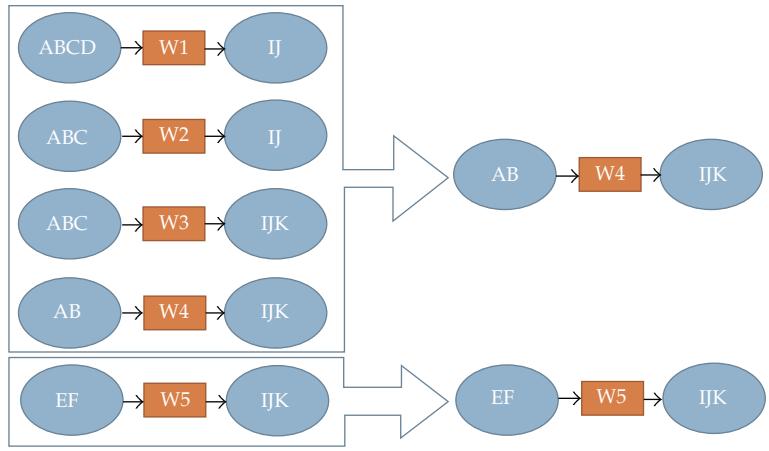

Figure 4: The example of service group extraction.

(I) Semantic Parameter Expansion

Semantic expansion is based on Sematic Similarity Module (SSM), which records relationships between semantic concepts. Querying the SSM according to the request will get a set of concepts which can meet the request. Then, the set of parameters can be used to select the Web services which satisfy the query and group them based on the degree of their similarity.

(II) Query Web Service Repository

From the previous step, we have a set of parameterized queries. Using them to query the database by matching services in the repository which output parameters can provide one of query parameter sets. It is assumed that there exists at least one service that can produce the expected output.

(III) Extract Web Service Group

Those Web services that are collected from the previous step will be classified into different groups and each group can be represented by one service. The extraction rule of service group is "effect $(w) \subseteq$ effect $(wg) \wedge$ precond $(w) \supseteq$ precond(wg)." The effect (w) and precond (w) mean the output and input results of Web services, respectively. For example, in Figure 4, there are five services, W3 has input parameters $\{A, B, C\}$ and output parameters $\{I, J, K\}$. W4 has input parameters $\{A, B\}$ and output parameters $\{I, J, K\}$. W4 uses fewer inputs and gets the same outputs, and then it concludes that W4 contains W3, and so on. It helps to reduce the search space of solutions.

3.3. Web Service Composition Module

In the proposed algorithm, a planning graph approach based on the backward strategy is adopted to solve the problem of large search space. The aim of a backward search is to find the initial states from end or intermediate states, so we propose an algorithm for solution

$i \leftarrow 0,\ P_0 \leftarrow s_0,\ G \leftarrow P_0$
repeat
  $G \leftarrow$ Expand Based On Backward (G)
  $S \leftarrow$ Extract Solutions (G, S)
  $S \leftarrow$ Reduce Solutions (S)
  $i = i + 1$
  Validate Solution (G, S)
Compute Score (G, S)
Output (Search Solution (S))

Algorithm 1

extraction from a planning graph, which help to find the initial state. The main composition algorithm is shown as follows.

3.3.1. Algorithm Compose (G, S)

$G = \langle P_0, A_1, P_1, \ldots, A_i, P_i \rangle$ is a simplified planning graph.
$S = \{s_1, \ldots, s_n\}$ is a set of solution candidates (Algorithm 1).
Web Service Composition Module includes the following four steps.

(I) Expand the Planning Graph

In this step, we will expand the planning graph with more action level for backward search. From the last proposition level in the planning graph, a list of expected parameters can be obtained, and then Service Group Matching Module can be interrogated to get service groups. Those service groups can be added to a new action level and arrange a new proposition level.

3.3.2. Algorithm Expand Based on Backward (G)

$G = \langle P_0, A_1, P_1, \ldots, A_i, P_i \rangle$ is a simplified planning graph.
$W = \{w_1, \ldots, w_n\}$ is a set of web services.
$WG = \{wg_1, \ldots, wg_n\}$ is a set of web service groups (Algorithm 2).

(II) Extract Solutions from the Planning Graph

After the previous step, we can get a planning graph $G\langle P_0, A_1, P_1, \ldots, A_i, P_i \rangle$ which contains action and proposition levels. In the solution extraction process, we have to trace to obtain possible solutions from the planning graph layer by layer, so keep those lists of service composition first and then expand them to the next new layer in the planning graph. In other words, we find out the service combinations to extend the service composition solutions from the action level. This step is for finding feasible composition solutions which correspond to the initial state of the request according to the planning graph established in previous steps.

```
for w ∈ W do
    for wg ∈ WG do
        if effect (w) ⊆ effect (wg) ∧ precond (w) ⊇ precond (wg) then
          wg ← wg ∪ w
        if effect (w) ⊇ effect (wg) ∧ precond (w) ⊆ precond (wg) then
          wg ← wg ∪ w
          effect (wg) ← effect (w)
          precond (wg) ← precond (w)
    if not be filled with service group then
      WG ← WG ∪ new wg
A_{i+1} ← WG
return G
```

Algorithm 2

```
for s ∈ S do
    for a ∈ A_i do
        available (a) ← true
        S ← S − s
do
    required ← inputs (s)
    ns ← new solution
    parent (ns) ← s
    for a ∈ A_i do
        if available (a) ∧ required ∩ effect (a) ≠ NULL then
            required ← required − (required ∩ effect (a))
            available (a) ← false
            ns ← ns ∪ a
    if required = NULL then
        S ← S ∪ ns
        break
while (required = NULL)
return S
```

Algorithm 3

3.3.3. Algorithm Extract Solutions (G, S)

$S = \{s_1, \ldots, s_n\}$ is a set of solution candidates.
inputs (s): a set of input parameters of solution s.
parent (s): the parent node of solution s.
available (a): it records action a whether available or not (Algorithm 3).

(III) Reduce Solutions

From the above step, it extracts possible solutions to form a set of service compositions. Two strategies in this research can be used to select the most appropriate solution. One of the strategies is that removing the solutions which utilize the number of services more than the threshold given in any new expansive level. "Service Threshold" means the max number of

```
for s ∈ S do
    if count (s) > Service Threshold then
        S ← S – s
for s ∈ S do
    if initial (s) ⊃ {initial (s2) | s2 ∈ S}) then
      S ← S – s
return S
```

Algorithm 4

services used in any composition solution. It is the key to reduce redundant solutions and facilitate the efficiency of composition algorithm. It can be determined by users according to their server performance. The Service Threshold value is set to 3000 by default value which is very huge in the experiments. The other is to remove the solutions which have too many services and similar to other short solutions in each action level. The process helps to filter large number of unwanted solutions and identified the most appropriate ones.

3.3.4. Algorithm Reduce Solutions (S)

initial (s): a set of initial input parameters of solution s.
count (s): a number of web services of solution s.
Service Threshold: a number of max services used in one solution (Algorithm 4).

(IV) Validate and Score Solutions

In order to filter out less appropriate solutions, a threshold value is given in this step to remove solutions with lower precision. "Threshold" means the minimum tolerable precision of output and input parameters matching results of Web services. It can also be described as the threshold of solution precision. Users can set the value to 1 when they want to obtain the completely matching solutions, otherwise, decrease this threshold value to allow unmatched but possible solutions. In addition, we repeat the above steps to expand the planning graph for finding all solutions and then calculate the score for each solution to find the best one. If the process stops in this step by threshold or finishes complete planning graph but still without solution, it means that there is no final solution found in this composition problem. The validate algorithm is shown as below, and the score approach for search optimal solution is in the Section 3.4.

3.3.5. Algorithm Validate Solution (G, S)

Initial (s): a set of initial input parameters of solution s.
precise (s): the precise of solution s that correspond to user request.
intersection: the amount of all same concepts.
union: the amount of all different concepts.
Precise Threshold: The threshold of solution precision (Algorithm 5).

Here is an example to explain the proposed composition mechanism. Assume a user's request which includes a set of input parameter $r_{\text{in}} = \{\text{A}, \text{B}, \text{C}, \text{D}\}$ and a set of output

```
for s ∈ S do
    intersection ← Count (initial (s) ∩ initial (g))
    union ← Count (initial (s) ∪ initial (g))
    precise (s) ← intersection/union
if Precise Threshold ≤ precise (s) then
        Search Solution (S) ← Search Solution (S) ∪ s
```

Algorithm 5

Table 1: The example of Web services.

Web service	Input parameters	Output parameter
W1	A, B, C	E, F
W2	A, B	H
W3	D, E	I
W4	E, F	J
W5		K, L
W6	G	L
W7	H	M
W8	I, J, K	M, N
W9	L	N

parameters $r_{\text{out}} = \{\text{M}, \text{N}\}$, and there are nine Web services in our Web Service Repository. Table 1 shows the details of the example of Web Service Repository.

Figure 5 shows the expanded planning graph result of the above given example. $\{\text{A}, \text{B}, \text{C}, \text{D}\}$ and $\{\text{M}, \text{N}\}$ are the input and output parameters of the composition request. At first, we search Web services which can output $\{\text{M}, \text{N}\}$, then we get $\{w_7, w_8, w_9\}$ which can support the proposition 4 (P4), our goal. Those three Web services will be involved in action 3 (A3). We collect input parameters of Web service in action 3 (A3), and we will get proposition 3 (P3). The rest of proposition and action are like this, and so forth. From the previous step, a planning graph is generated to extract solutions. We utilize our proposed algorithm to extract solutions which store them in a tree structure to form a solution tree for tracing. Every leaf node in solution tree means that there is a solution from leaf node to root. The result is shown in Figure 6. Many possible paths that can reach initial state of the tree have been discovered. This is one of advantages of adopting the backward strategy, so that multiple solutions for user request can be found.

$\{\text{M}, \text{N}\}$ are the output parameters of user request. It is located in proposition 4 (P4), so we need to find the combinations of Web services in action 3 (A3), which corresponds to $\{\text{M}, \text{N}\}$. And we will get two combinations, which are $\{\text{W7}, \text{W9}\}$ and $\{\text{W8}\}$. Those combinations will be added to the root as its child. Now, there are two nodes at second level. The solution node composited by $\{\text{W7}, \text{W9}\}$ requires a set of input parameters $\{\text{H}, \text{L}\}$, so we need to find the combinations of Web services in action 2 (A2), which can correspond to $\{\text{H}, \text{L}\}$. So we get $\{\text{W2}, \text{W5}\}$ and $\{\text{W2}, \text{W6}\}$, and so forth.

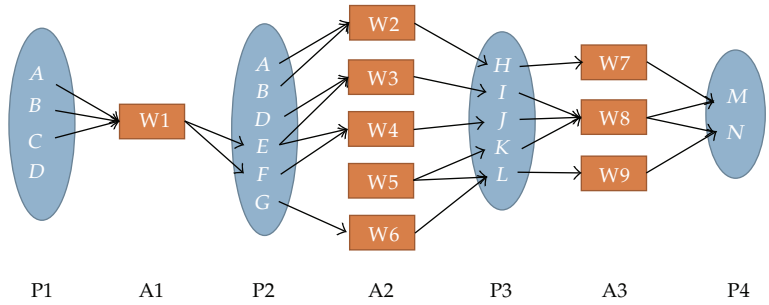

Figure 5: The simplified planning graph for the above example.

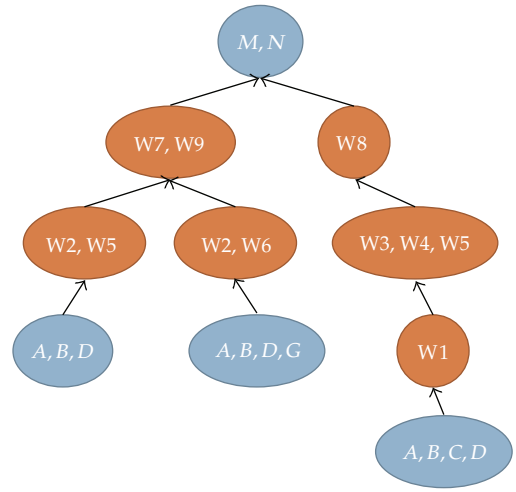

Figure 6: The solution tree for the planning graph for the above example.

3.4. Search Optimal Solution Module

After establishing the solution tree, it found a number of service composition solutions which possibly satisfy user request. Then, the Search Optimal Solution Module needs to give a score on each solution and select the highest score one. At first, we calculate the precision of the matched solutions with user request. We utilize the ratio of the number of intersections to the number of unions, which are between the solution's initial states and the user request's inputs. The precision equation is in the following.

$$\text{Precise } (s_1, s_2) = \frac{\cap s_1 s_2}{\cup s_1 s_2}. \tag{3.1}$$

Equation (3.1) shows the precision. In this equation, $\cap s_1 s_2$ represents the amount of the same concepts and $\cup s_1 s_2$ represents the amount of all different concepts, where S_1 and S_2 both are lists of concepts. This equation evaluates the similarity between the solution's initial states and the user request's inputs. It also means the difference between two lists of concepts. After the previous step, we have the likelihood of the solutions to achieve the goal.

For calculation of the score for each solution, we need to calculate matching degree between the levels of the solutions. The matching equation is illustrated in the following

$$\begin{aligned} &\mathrm{Mat}\,(s_1, s_2) = (\mathrm{KM} - \alpha(\cup s_1 s_2 - \cap s_1 s_2)) \\ &\quad \text{where } \mathrm{KM} = \mathrm{KM}\langle \mathrm{Sim}(c_1, c_2)\rangle \ (c_1 \in s_1, c_2 \in s_2). \end{aligned} \tag{3.2}$$

Equation (3.2) shows the matching score. In the above equation, KM represents classical Kuhn-Munkres algorithm which solved the assignment problem, and Sim represents the similarity between any two concepts in s_1 and s_2. This equation is to evaluate the matching score of two solutions. With the previous two formulas, we can calculate the solution score. It sums the matching scores between levels of the solutions and divides by the number of levels to gain the average. Then, we get the average of matching scores, and multiple by the matching precision of the solution and the request. The score equation is shown as below.

$$\mathrm{Score}\,(\mathrm{slu}) = \mathrm{Precise}\,(\mathrm{slu}\cdot \mathrm{in},\ r\cdot \mathrm{in})\ \frac{\sum_{s_1,s_2}^{\mathrm{slu}} \mathrm{Mat}\,(s_1, s_2)}{n}. \tag{3.3}$$

Equation (3.3) shows the solution score. Solution slu is a list of nodes from node leaf to the root, which represents each level of service composition, slu $\cdot$ in represents the input parameters of the solution slu, $r \cdot$ in represents the input parameters of the request r, and the number of levels is represented by n. The Precision calculates the matching precision between the input parameters of the request r and the input parameters of the solution slu.

Briefly, our proposed mechanism is particularly suitable for large-scale service composition due to the problems that service composition is an error-prone and complicated process. In this study, we designed a cost-effective planning graph mechanism for finding multiple service composition solutions with lower service cost to overcome the above problems. Furthermore, in some cases, the exact solution does not exist, and our proposed mechanism still can recommend approximate solutions. On the other hand, in order to provide multiple solutions, it must need to trace each possible solution. Our proposed mechanism can filter out and reduce the less appropriate solutions in order to avoid exponential growth in complexity.

4. Experiments

In this section, the simulation design, assumption, performance metrics, and simulation results are described. Moreover, there is a brief discussion in the later section.

4.1. Simulation Design

We established a simulation platform which is based on the proposed mechanism for validating our algorithm. In this platform, WSBen [14], which is widely used to evaluate the efficiency and effectiveness in Web service discovery and composition, is utilized to generate different test sets to validate the proposed algorithm. The simulation platform will carry out the algorithm according to the test dataset and the requests derived from WSBen.

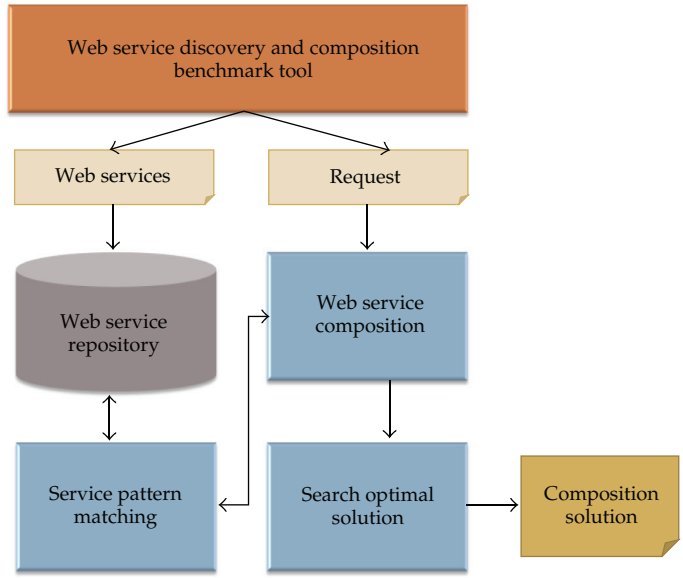

Figure 7: The architecture of simulation platform.

The architecture of simulation platform is shown in Figure 7. WSBen was set up with the test data including a set of Web services and a set of feasible requests for testing WSC algorithm. In the simulation, a program extracts the information of Web service generated by WSBen and then stores it in the Web Service Repository which includes service name, input parameters, and output parameters. The Service Group Matching Module is in charge of Web services selection according to the service query which has been processed by the Web Service Composition Module. In addition, it also groups the selected Web services as Web service groups for the composition algorithm. The Web Service Composition Module including a composition algorithm composes a composite service according to the composition request, which interacts with the service matching module in the discovery and selection process until a possible solution had been found, if there is any. Therefore, it will generate a list of candidate composite Web services which can fulfill the required services. The Search Optimal Solution Module calculates each candidate solution to obtain a score in order to generate an ordered recommendation list for selection.

4.2. Simulation Assumption

WSBen is a Web services generation and benchmark tool for Web services composition and discovery, which provides a set of functions to simplify the generation of Web service test datasets and builds test environments including the testing requests [14]. A complex graph network with nontrivial topological features does not occur in a simple form such as lattices or random graphs but often occur in real applications. Many systems in nature can be described by models of complex networks, which are structures consisting of nodes or vertices connected by links or edges [15]. Such as a Web service can be assumed to be a transformation between two difference domain nodes, this could be regarded as clusters of parameters. The development of WSBen is based on the above assumption. The complex

network is most commonly used in three concepts—"random," "small-world," and "scale-free" network [15]. Therefore, the tool provides these three different types of network models to generate Web services test datasets. It also can generate diverse sizes of datasets based on the complex network type specified.

The network topology will be constructed as a directed graph based on graph theory. Each node is represented as a parameter cluster, and each edge is represented as a connecting between two different clusters, that will be regarded as generation template of Web services. The development of WSBen is based on the above assumption. This test environment also includes five feasible and correct requests ($r1$, $r2$, $r3$, $r4$, and $r5$) generated by WSBen. We put them into all experimental cases as the test requests in three network models for evaluation. There is an input framework that users can specify the generated Web service and the characteristics of network topology in WSBen. The input framework xTS = $\langle |J|, G_r, \eta, M_p, |W| \rangle$ is described as below.

(I) $|J|$ is the total number of parameter cluster.

(II) G_r donates a graph model to specify the topology of parameter cluster network. The three types of network, which are "random," "small-world," and "scale-free" complex networks, can be simulated by the three network model, as follows.

(i) *Erdo-Renyi* $(|J|, p)$. The model has such a simple generation approach that it creates $|J|$ nodes in graph and assign each edge in the graph with probability p.

(ii) *Newman-Watts-Strogatz* $(|J|, k, p)$. The initialization is a ring graph with k nodes. Each node adds to graph and constructs edge by connecting to others with probability p. The process will iterate until there are $|J|$ nodes in the graph.

(iii) *Barabasi-Albert* $(|J|, m)$. There are m nodes with no edge in the initial graph. Each node adds with m edges, which are preferentially attached to existing nodes with high degrees.

(III) η donates the parameter condense rate. Users can control the density of partial matching cases in generated Web services.

(IV) M_p donates the minimum number of parameters in a cluster. In other words, each cluster has at least M_p parameters.

(V) $|W|$ donates the total number of Web services in a test dataset.

The assumptions in datasets for simulation have to be described in advance. There are three types of network in the simulation, which are random, small-world, and scale-free types. Each network is assumed as a parameterized cluster network, and a Web service is a transformation between two clusters. Each cluster contains its parameters, and it is also called node. In other words, the input and output parameters of a Web service can be generated and selected from each two domain cluster nodes according to argument η, M_p, and the generation rules of WSBen, and a Web service generated in the network could be regarded as an edge. WSBen provides a set of functions to simplify the generation of test data for WSC algorithm. It generates Web services according to the parameter cluster network which users specify. In our simulation, we assume that there are 100 clusters in network and the parameter condense rate is 0.8. The configurations for three types of network model in our experiment platform are as follows.

(I) Random Network: Barabasi-Albert (100, 0.06). The model creates 100 nodes in the graph and each edge in the graph is with probability 0.06 to be chosen.

(II) Small-World Network: Newman-Watts-Strogztz (100, 6, 0.1). The initialization is a ring graph with 6 nodes. Each node adds to the graph and constructs edges connected to each other with probability 0.1, until there are 100 nodes in this graph

(III) Scale-Free Network: Erdo-Reyi (100, 6). There are 6 nodes with no edge in the initial graph. Each node adds 6 edges until reach 100 nodes. Each added edge is preferentially attached to existing nodes with high degrees.

For each network, there are 10 different sizes in each of test data types, which sizes are 10,000 to 100,000, respectively. Thus, there are 30 test sets (three frameworks multiplied by ten different test sizes) in our large-scale WSC simulation.

4.3. Evaluation Criteria

Effectiveness, efficiency, and feasibility are three evaluations, which are used to test our proposed approach. We use diverse sizes of Web services and three types of Web service networks to measure the scalability and robustness of our approach. The evaluation metrics are as follows.

(I) #T: it measures the time that is required by the algorithm to find a fully matched or approximate solution. In other words, it is a measure of computational efficiency.

(II) #C: the number of Web services in a solution of WSC problem, which also stands for composite cost. It is a measure of cost and effectiveness.

(III) #L: the number of nested levels of composite Web services in a solution of WSC problem. It is also a measure of cost and effectiveness.

(IV) #P: it measures the difference of input parameters between the final solution and user request. It means that there exists a completely correct solution if the #P rate is 100%, otherwise, the solution is an approximate answer with some missing input parameters, and user can add those missing input parameters to achieve the goal. Precision (described in (3.1)) is a measure of effectiveness.

4.4. Simulation Results

In this section, we show the efficiency, effectiveness, and feasibility of the proposed algorithm in three cases. We utilize diverse sizes of Web services and different type of network topology to observe the scalability and robustness of our proposed algorithm. Some related results are illustrated in the below sections. The three test datasets of our experiments deal with the networks of random, small, and scale-free type. We compare the proposed backward strategy with the forward strategy in the previous study [7] to observe the results. The evaluation criteria are four indexes: #T, #C, #L, and #P which are described in Section 4.3.

Case 1 (Random Network). Table 2 shows the results of five requests of random network with $|W| = 10,000$ Web services in a test dataset. The results of criterion #P mean that both Backward and Forward strategies can find solutions in all cases (#$P = 1$). Regarding #L, Backward and Forward strategies have no difference in finding solutions. In the criterion of #T, the computational efficiency of Forward is better than Backward, because of generating

Table 2: Results of random network with $|W| = 10000$.

Test request	Backward				Forward			
	#L	#C	#T	#P	#L	#C	#T	#P
r1	8	18	2.725	1	8	262	1.033	1
r2	8	14	3.028	1	8	237	1.05	1
r3	7	9	1.916	1	7	220	1.066	1
r4	7	10	2.191	1	7	245	1.072	1
r5	9	16	4.141	1	9	241	1.062	1
Avg.	**7.8**	**13.4**	**2.8**	**1**	**7.8**	**241**	**1.056**	**1**

Table 3: Results of small world network with $|W| = 10000$.

Test request	Backward				Forward			
	#L	#C	#T	#P	#L	#C	#T	#P
r1	14	14	1.599	1	14	183	0.863	1
r2	11	11	1.016	1	11	175	0.769	1
r3	12	12	1.985	1	12	156	0.746	1
r4	10	10	2.419	1	10	174	0.716	1
r5	16	16	1.854	1	16	181	0.903	1
Avg.	**12.6**	**12.6**	**1.776**	**1**	**12.6**	**173.8**	**0.8**	**1**

final solutions by a backward search to expand graph layers is very time consuming. It still falls within an acceptable time frame in our proposed approach even in the large-scale (10000 Web services) environment. In the experiment with #*C* Web services, our proposed Backward approach outperforms the Forward. Backward strategy uses less 20 services to fulfill the request in all cases, but the Forward strategy requires more than 200 Web services to find a solution.

Case 2 (Small World Network). Table 3 shows the results of five test requests of a small world network with $|W| = 10,000$ Web services in a test dataset. The results of criterion #*P* mean that both our proposed Backward and the Forward strategies still can find solutions in all cases ($\#P = 1$). Regarding #*L*, the result of Backward strategy is as good as the Forward strategy. Our proposed Backward strategy takes more a little time than the Forward strategy in #*T*, but processing time is still acceptable. The reason is to generate final solutions by a backward search to expand graph layers is very time consuming. Finally, in the experiment with #*C*, it shows it has produced much better performance than the Forward strategy.

Case 3 (Scale-Free Network). Table 4 shows the result of five test requests of a scale-free network which contains $|W| = 10,000$ Web services in a test dataset. The results of criterion #*P* mean that Forward strategy still can satisfy all requests ($\#P = 1$), but it requires more than 200 Web services to obtain the solution in service cost #C. In the more complex scale-free network, although Backward strategy finds the fully matched solution is impossible in some cases, an approximate solution can be found and replaced, which use much less services to satisfy the request. Regarding #*T* and #*L*, the result shows that Backward strategy are a bit better than Forward in #*T*, but both produce almost the same performance in these two criteria.

Table 4: Results of scale-free network with $|W| = 10000$.

Test request	Backward				Forward			
	#L	#C	#T	#P	#L	#C	#T	#P
$r1$	4	11	1.232	0.933	4	244	2.48	1
$r2$	4	6	3.151	1	4	343	1.654	1
$r3$	5	11	2.886	0.778	5	356	2.824	1
$r4$	—	—	—	—	4	313	1.414	1
$r5$	4	10	0.203	0.814	4	281	2.122	1
Avg.	**4.25**	**9.5**	**1.868**	**0.8812**	**4.4**	**316**	**2.3808**	**1**

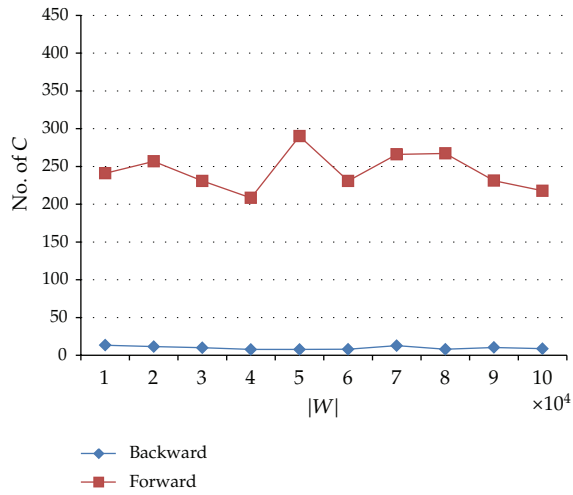

Figure 8: The average cost of finding solution in random network.

As shown in Figures 8, 9, and 10, the proposed algorithm has much better usage of Web services in all cases in terms of obtaining the solution. Because of the aim of the forward algorithm, which is to reach the goal as quick as possible, it expands the search space in planning graph for services composition problem no matter how many redundant Web services are produced. However, the proposed backward algorithm has no redundant Web services existed in the solution, because its backward strategy searches what it needs to reach the initial state. Figure 10 shows average cost of searching solution in Scale-Free Network. It can be observed that the forward algorithm represents an unstable circumstance in Scale-Free Network, when the size of a test dataset becomes large. Nevertheless, our backward algorithm is still to appear stable and effective results. On average, our algorithm reduces to 94% service cost for finding the solutions. From the above experiment results, we have a simple deduction that the backward search will continue to display the stable and smooth results in different types of network topology, even if the sizes of Web services continue to increase.

In experiments, three types of network topology and diverse sizes of Web services are utilized to evaluate these two algorithms. The main findings about the proposed algorithm from the experiments are given as follows. In effectiveness, the experiment results show our proposed backward algorithm has 94% better effectiveness compared to the forward algorithm in most cases. In few cases, it uses little more levels to obtain the solution, but it

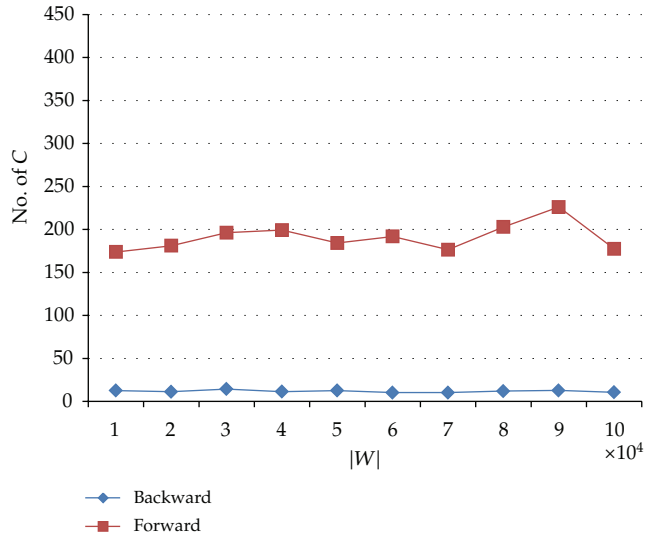

Figure 9: The average cost of finding solution in small world network.

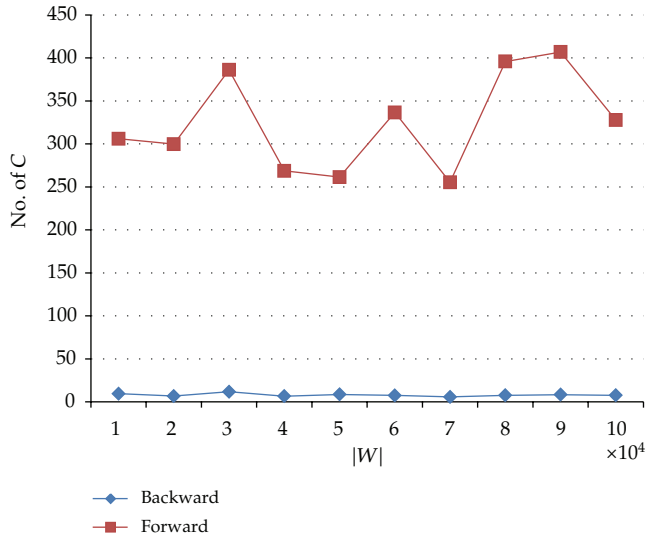

Figure 10: The average cost of finding solution in scale-free network.

can get low-cost solutions. As confirmed by the experiment results, our proposed algorithm can also get very high-precise solutions in most cases. In efficiency, although the forward algorithm has better performance than the backward algorithm, the cost of solution is very high. The proposed backward algorithm is efficient when the sizes of Web service are less than 50,000. Therefore, the effect of this cost-effective approach for large-scale WSC problem is exhibited.

5. Conclusion

In this study, we proposed a backward planning graph mechanism for Web service composition on cloud environment. It utilizes a planning graph based on a backward search to find multiple feasible solutions and recommends the best composite solution based on their service costs. We also validated that the proposed algorithm can improve the error-prone problem of service composition and the redundant Web service involved in large-scale service composition problem. Therefore, the algorithm based on the backward planning graph search, which is capability of recommending multiple service composition and remove the redundant services. As the experiment results in this paper, we proved that our proposed backward algorithm had a better cost-effectiveness than the forward search algorithm in terms of service cost. The proposed algorithm is able to recommend approximate solutions of service composition using very few Web services, because it has higher quality of relationships between services. In other words, we can decrease the amount of cost of Web services and remain acceptable planning graph levels and execution time.

In the future, we will study how to improve a greedy algorithm in order to expand the solution tree. To obtain right combinations is a very important issue for the design of algorithm. It cannot only help to decrease wrong combinations, but also to improve the effectiveness and efficiency of algorithm. Moreover, we can add more predictable restrictions to prune the huge combination tree nodes for our algorithm efficiency. If there are some more predictable restrictions and composition information, then that will help us to make more appropriate decisions to find the solutions. Moreover, because the Semantic Similarity Module in the proposed approach is optional, this module has been not analyzed in the experiments. There is a need to have an environment that can help to validate semantic association of service compositions. To design a semantic experiment environment should be undertaken determining how the semantic can influence the effectiveness of service composition.

Acknowledgment

This research work was supported by the National Science Council of the Republic of China under the Grant NSC-99-2410-H-009-036-MY3. This research is carried out as a part of GREENet which was supported by a Marie Curie International Research Staff Exchange Scheme Fellowship within the 7th European Community Framework Programme under grant agreement no. 269122.

References

[1] H. Haas and A. Brown, "Web Services Glossary," http://www.w3.org/TR/ws-gloss/.
[2] D. Booth, H. Haas, F. McCabe et al., "Web Services Architecture," http://www.w3.org/TR/ws-arch/.
[3] M. Kuzu and N. K. Cicekli, "Dynamic planning approach to automated web service composition," *Applied Intelligence*, vol. 36, no. 1, pp. 1–28, 2010.
[4] S. Wang, W. Shen, and Q. Hao, "Agent based workflow ontology for dynamic business process composition," in *Proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design (CSCWD '05)*, pp. 452–457, Coventry, UK, 2005.
[5] L. Qiu, L. Chang, F. Lin, and Z. Shi, "Context optimization of AI planning for semantic Web services composition," *Service Oriented Computing and Applications*, vol. 1, no. 2, pp. 117–128, 2007.
[6] A. Alves, A. Arkin, S. Askary et al., "Web Services Business Process Execution Language Version 2.0," OASIS WSBPEL TC, 2007.

[7] X. Zheng and Y. Yan, "An efficient syntactic web service composition algorithm based on the planning graph model," in *Proceedings of the IEEE International Conference on Web Services (ICWS '08)*, pp. 691–699, September 2008.
[8] Y. Yan and X. Zheng, "A planning graph based algorithm for semantic web service composition," in *Proceedings of the IEEE Joint Conference on E-Commerce Technology (CEC '08) and Enterprise Computing, E-Commerce and E-Services (EEE '08)*, pp. 339–342, July 2008.
[9] J. Peer, *Web Service Composition as AI Planning—a Survey*, University of St.Gallen, St. Gallen, Switzerland, 2005.
[10] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.
[11] S. Kambhampati and M. B. Do, "Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP," *Artificial Intelligence*, vol. 132, no. 2, pp. 151–182, 2001.
[12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.
[13] Y. Dimopoulos, B. Nebel, and J. Koehler, "Encoding planning problems in nonmonotonic logic programs," in *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning (ECP '97)*, pp. 169–181, Toulouse, France, 1997.
[14] S. C. Oh and D. Lee, "WSBen: A web services discovery and composition benchmark toolkit," *International Journal of Web Services Research*, vol. 6, no. 1, pp. 1–19, 2009.
[15] X. F. Wang and G. Chen, "Complex networks: Small-world, scale-free and beyond," *IEEE Circuits and Systems Magazine*, vol. 3, no. 1, pp. 6–20, 2003.