

QoS-Driven Web Service Composition Using Learning-based Depth First Search

Wonhong Nam Hyunyoung Kil
Pennsylvania State University
University Park, PA 16802, USA.
Email: {wnam, hykil}@psu.edu

Jungjae Lee
Korea University
Seoul, 136-701, Korea.
Email: jjlee@formal.korea.ac.kr

Abstract—The goal of the Web Service Composition (WSC) problem is to find an *optimal composition* of web services to satisfy a given request using their syntactic and/or semantic features. In this paper, in particular, we study the *Quality of Services (QoS)*-driven WSC problem to optimize service quality criteria, e.g., response time and/or throughput. We propose a novel solution based on Learning-based Depth First Search (LDFS). Given a set of web service descriptions including QoS information and a requirement web service, we reduce the QoS-driven WSC problem into a planning problem on a state-transition system. We then find the optimal solution for the problem using a dynamic programming based on LDFS which recently has shown a promising result.

I. INTRODUCTION

Web services are software systems designed to support machine to machine interoperations over the Internet. Many researches have been carried out for the web service standard, and these efforts significantly have improved flexible and dynamic functionality of *Service Oriented Architectures (SOA)* in the current semantic web services. However, a number of research challenges still remain [1]; e.g., automatic web service discovery, web service composition and formal verification for composed web services. Recently, the *web service composition (WSC)* [2] requires to discover service providers that satisfy not only functional requirements but also nonfunctional ones, including *Quality of Services (QoS)* constraints. In this case, one desires the composite web service, which is not a simple shortest path from the initial state to a given goal but the optimal composition to maximize the QoS value of the result composition.

In this paper, we propose a novel technique to identify an *optimal composition* with respect to QoS value. Given a set of web service descriptions including QoS information (e.g., the response time and/or throughput of each web service) and a requirement web service, our algorithm identifies the *composite web service* such that we can legally invoke the next web service in each step, achieve the desired requirement, and maximize QoS. We first reduce the composition problem into a *planning problem* on a state-transition system where the optimal cost path to a goal state corresponds to the optimal composition of web services with respect to QoS. To solve the optimal planning problem, we employ a state-of-the-art technique using *Learning Depth First Search*

(LDFS) [3], [4] which has recently shown a promising result. To the best of our knowledge, there is no work to employ the LDFS technique for this problem setting.

II. QOS-DRIVEN WEB SERVICE COMPOSITION

In this section, we formalize the notion of web services and their QoS-driven composition that we consider in this paper. A *web service* is a tuple $w(I, O, Q)$ with the following components:

- I is a finite set of *input parameters* for w .
- O is a finite set of *output parameters* for w ; each input/output parameter $p \in I \cup O$ has a type t_p .
- Q is a finite set of *quality criteria* for w .

When a web service $w(I, O, Q)$ is invoked with all the input parameters $i \in I$ with the type t_i , it returns all the output parameters $o \in O$ with the type t_o . The invocation of the web service w corresponds to each service quality criterion $q \in Q$, for instance, a response time q_r or a throughput q_t . We assume that the signature description for each web service is given in *Web Services Description Language (WSDL)* and the quality criteria of a web service is given in *Web Service Level Agreements (WSLA)*. Given two types t_1 and t_2 , t_1 is a *subtype* of t_2 (denoted by $t_1 <: t_2$) if t_1 is more informative than t_2 so that t_1 can substitute for t_2 everywhere. In this case, t_2 is a *supertype* of t_1 . This relation is reflexive (i.e., $t <: t$ for any type t) and transitive (i.e., if $t_1 <: t_2$ and $t_2 <: t_3$ then $t_1 <: t_3$). We assume that the type hierarchy is given; e.g., specified in OWL. Given two web services $w_1(I_1, O_1, Q_1)$ and $w_2(I_2, O_2, Q_2)$, we denote $w_1 \sqsupseteq_I w_2$ if w_1 requires more informative inputs than w_2 ; i.e., for every $i_2 \in I_2$, there exists $i_1 \in I_1$ such that $t_{i_1} <: t_{i_2}$. Given two web services $w_1(I_1, O_1, Q_1)$ and $w_2(I_2, O_2, Q_2)$, we denote $w_1 \sqsubseteq_O w_2$ if w_1 provides less informative outputs than w_2 ; i.e., for every $o_1 \in O_1$, there exists $o_2 \in O_2$ such that $t_{o_2} <: t_{o_1}$. A *web service discovery problem* is, given a set W of available web services and a request web service w_r , to find a web service $w \in W$ such that $w_r \sqsupseteq_I w$ and $w_r \sqsubseteq_O w$.

However, it might happen that there is no single web service satisfying the requirement. In that case, we want to find a sequence $w_1 \cdots w_n$ of web services such that we

can invoke the next web service in each step and achieve the desired requirement eventually. Formally, we extend the relations, \sqsupseteq_I and \sqsubseteq_O , to a sequence of web services as follows.

- $w \sqsupseteq_I w_1 \cdots w_n$ (where $w = (I, O, Q)$ and each $w_j = (I_j, O_j, Q_j)$) if $\forall 1 \leq j \leq n$: for every $i_2 \in I_j$ there exists $i_1 \in I \cup \bigcup_{k < j} O_k$ such that $t_{i_1} <: t_{i_2}$.
- $w \sqsubseteq_O w_1 \cdots w_n$ (where $w = (I, O, Q)$ and each $w_j = (I_j, O_j, Q_j)$) if for every $o_1 \in O$ there exists $o_2 \in \bigcup_{1 \leq j \leq n} O_j$ such that $t_{o_2} <: t_{o_1}$.

Finally, given a set W of available web services and a service request w_r , the *QoS-driven web service composition problem* $\langle W, w_r \rangle$ which we focus on in this paper is to find a sequence $w_1 \cdots w_n$ (every $w_j \in W$) of web services such that $w_r \sqsupseteq_I w_1 \cdots w_n$ and $w_r \sqsubseteq_O w_1 \cdots w_n$. The optimal solution for this problem is to find a sequence to maximize the target value $v = c_1 \cdot q_1 + \cdots + c_m \cdot q_m$ where each q_i is a quality criterion in Q and c_i is a weight for q_i .

III. REDUCTION TO A PLANNING PROBLEM

Given a QoS-driven web service composition problem $\langle W, w_r \rangle$, the problem can be reduced into a planning problem on a state-transition system. A *state-transition system* is a 6-tuple $\mathcal{S} = (S, s_0, F, A, T, C)$ with the following components:

- S is a finite set of *states*.
- $s_0 \in S$ is an *initial states*.
- $F \subseteq S$ is a set of *final states*.
- A is a set of *actions*.
- $T \subseteq S \times \Sigma \times S$ is a *transition relation*.
- For each state s and each action a , $C(s, a)$ is the action cost.

The planning problem is to find a sequence of actions from the initial state to a final state which minimizes the total cost. The solution can be represented in terms of Bellman equation which characterizes the *optimal cost function* [5]:

$$V(s) = \begin{cases} 0 & \text{if } s \text{ is a final state} \\ \min_{a \in A} Q_V(s, a) & \text{otherwise} \end{cases}$$

where $Q_V(s, a)$ is the *cost-to-go* which for Max and Additive graphs takes the form:

$$Q_V(s, a) = \begin{cases} C(s, a) + \max_{s' \in T(s, a)} V(s') & \text{(Max)} \\ C(s, a) + \sum_{s' \in T(s, a)} V(s') & \text{(Additive)} \end{cases}$$

There exists a unique optimal value function $V^*(s)$ that solves the Bellman equation, and the optimal solution can be represented as a strategy σ . The strategy σ is a function mapping a state $s \in S$ into an action $a \in A$.

Given a set $W = \{w_1, \cdots, w_n\}$ of web services where for each j , $w_j = (I_j, O_j, Q_j)$, we denote as TP a set of all types t such that there exists $p \in \bigcup (I_j \cup O_j)$ and t is the type of p . Then, given a set $W = \{w_1, \cdots, w_n\}$ of web services

and a requirement web service $w_r(I_{w_r}, O_{w_r}, Q_{w_r})$, we can construct a state-transition system $\mathcal{S} = (S, s_0, F, A, T, C)$ as follows:

- $S = \{(x_1, \cdots, x_m) \mid x_j = \text{true or false}, m = |TP|\}$; each boolean variable x_j represents whether we have an instance with the type $t_j \in T$ at a state s .
- $s_0 = (x_1, \cdots, x_m)$ where each x_j is *true* if there exists an input parameter $i \in I_{w_r}$ such that its type t_{x_j} is a supertype of t_i (i.e., $t_i <: t_{x_j}$); otherwise x_j is *false*.
- $F = \{(x_1, \cdots, x_m) \mid \text{each } x_j \text{ is true iff there exists an output parameter } o \in O_{w_r} \text{ such that its type } t_{x_j} \text{ is a subtype of } t_o \text{ (i.e., } t_{x_j} <: t_o)\}$.
- $A = W$.
- For each j , $T(s, w_j, s') = \text{true}$ where $s = (x_1, \cdots, x_m)$, $s' = (x'_1, \cdots, x'_m)$ (each x_k and x'_k are *true* or *false*), and $w_j = (I_j, O_j, Q_j)$ iff (1) for every $i \in I_j$, there exists x_k in s such that x_k is *true* and its corresponding type t_{x_k} is a subtype of the type of i (i.e., $t_{x_k} <: t_i$), (2) if x_l is *true*, x'_l is also *true*, and (3) $\forall o \in O_j$: for every variable x'_k in s' , if its corresponding type $t_{x'_k}$ is a supertype of t_o , x'_k is *true*. Intuitively, if a web service w_j is invoked at a state s where we have data instances being more informative than inputs of w_j , we proceed to a state s' where we retain all the data instances from s and acquire outputs of w_j as well as their supertypes.
- For every state s , $C(s, a) = -(c_1 \cdot q_1 + \cdots + c_m \cdot q_m)$ where each q_j is given from w which is corresponding to a . Since while we want to maximize the target value in WSC problem, this planning problem minimizes the total cost, we take the negative value as the cost. In addition, we assume that each weight constant c_j is given.

Intuitively, we have an initial state where we possess all the data instances corresponding to the input of w_r as well as ones corresponding to their supertypes. As goal states, if a state is more informative than the outputs of w_r , it is a goal state. Finally, given a QoS-driven web service composition problem $\langle W, w_r \rangle$, we can reduce it into a planning problem on $\mathcal{S} = (S, s_0, F, A, T, C)$ where the optimal cost path from an initial state to a goal state corresponds to the optimal composition of web services. For the sake of space, we omit a formal proof for our reduction.

IV. LEARNING DEPTH FIRST SEARCH

Learning Depth First Search (LDFS) [3], [4] is a dynamic programming technique with the effectiveness of heuristic search methods. The basic idea of LDFS is to search solutions by combining iterative, bound depth first searches with learning [6], [7]. In each iteration, if there is a solution with cost not exceeding a lower bound, then the solution is accepted as the optimal and the algorithm terminates.

Otherwise, the process is repeated with the lower bound and the value function updated. The LDFS generalizes two key characteristics underlying effective search algorithms for various models—*learning* and *lower bounds*. Moreover, the LDFS technique has recently shown a promising result [3], [4]; it shows better execution times for a number of experiments than state-of-the-art techniques such as AO* algorithm, value iteration, and another learning algorithm called Min-Max LRTA* [8].

In this paper, we assume that we have an initial value function V which is a *lower bound* (i.e., $V(s) \leq V^*(s)$) and *monotonic* (i.e., $V(s) \leq \min_{a \in A} Q_V(s, a)$) for every non-terminal state. Then, *learning* [6], [7] modifies this value function V by Bellman update $V(s) := \min_{a \in A} Q_V(s, a)$. It is obvious that V is the optimal if $V(s)$ is equal to $\min_{a \in A} Q_V(s, a)$ for every state. However, given a fixed initial state s_0 , it is not necessary that the above condition is true in all the states. Indeed, V is the optimal if $V(s)$ is equal to $\min_{a \in A} Q_V(s, a)$ for every state s which is reachable from s_0 and the corresponding strategy σ . Therefore, the basic idea of LDFS is to search a state s which is reachable from s_0 and σ such that $V(s) < \min_{a \in A} Q_V(s, a)$, and update $V(s)$. To find such a state, LDFS employs depth first search. It repeats this iteration until there is no such state. In what follows, we will say that a state s is *consistent* if $V(s) = \min_{a \in A} Q_V(s, a)$.

Figure 1 presents our QoS-driven WSC algorithm using LDFS. Given a set W of web services and a requirement web service w_r , we first reduce the QoS-driven WSC problem into a planning problem on a state-transition system \mathcal{S} (line 1). Then, we repeat LDFS with the initial state s_0 until obtaining the optimal strategy σ (line 2), and then translate σ to the optimal composition sequence of web services (line 3). The procedure LDFS that we invoke considers all the actions in a state, backtracks on unsolved states, and updates the unsolved states as well as their ancestors. For this process, LDFS includes two loops—one over the all actions $a \in A$ on the given state s (line 11) and the other over all the possible successor states $s' \in T(s, a)$ (line 14). This search process terminates on an inconsistent state s (i.e., $V(s) < \min_{a \in A} Q_V(s, a)$ for every action a), a terminal state, or a state labeled as *solved*. If the search finds an action $a \in A$ on a state s such that there does not exist any inconsistent state below s , then s is labeled as *solved*. In this case, the strategy σ for the state s is set to a , and other actions are skipped. Then, the procedure LDFS returns *true*. Otherwise, the search process tries another action until no one is left. If there is no action satisfying the above condition, s is updated. In this case, LDFS returns *false*. Similarly, if the search below a successor state $s' \in T(s, a)$ returns *false* or there is no action a satisfying ($Q_V(s, a) \leq V(s)$), the rest of successors are skipped.

As a summary, Figure 2 illustrates the architecture of our technique. Our technique takes a WSDL file for a set of

```

WS_seq QoS_WSC(WS_Set W, Req_WS w_r){
1: Reduce(W, w_r, S);
2: repeat flag := LDFS(s_0) until flag = true;
3: Translate(σ, sol_seq);
4: return sol_seq;
}

Boolean LDFS(state s){
5: if s.tag = terminal then {
6:   V(s) = 0;
7:   s.tag := solved;
8: }
9: if s.tag = solved then return true;
10: flag = false;
11: foreach a ∈ A do {
12:   if Q_V(s, a) > V(s) then continue;
13:   flag := true;
14:   foreach s' ∈ T(s, a) do {
15:     flag := LDFS(s') ∧ (Q_V(s, a) ≤ V(s));
16:     if ¬ flag then break;
17:   }
18:   if flag then break;
19: }
20: if flag = true then {
21:   σ(s) := a;
22:   s.tag := solved;
23: } else V(s) := min_{a ∈ A} Q_V(s, a);
24: return flag;
}

```

Figure 1. QoS-driven WSC algorithm using LDFS

web services, an OWL file for type information, a WSLA file for QoS information, and a WSDL file for a requirement web service. It reduces the QoS-driven WSC problem into a planning problem, and then compute an optimal strategy. Finally, our technique translates the optimal strategy into a BPEL file for the optimal composition.

V. RELATED WORK

Quality driven web service compositions have been studied by several researches. Zeng et al. [9] employ an efficient linear programming method for this problem. Lin et al. [10] formalize the service selection problem as a fuzzy constraint satisfaction problem, and they use deep-first branch-and-bound method to search a solution. However, to the best of our knowledge, there is no research to apply the LDFS technique for the QoS-driven WSC problem. There are a number of related studies to LDFS. By applying the two-player game assumptions, Korf [6] has presented an efficient algorithm, Real-Time-A*, and he also has proposed its learning version LRTA* which improves its performance over successive problem solving trials by learning more

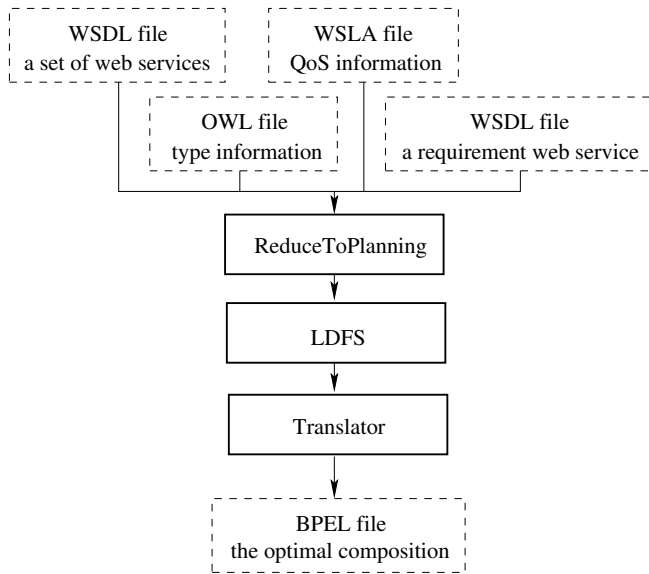


Figure 2. Architecture diagram

accurate heuristic values. Koenig [8] proposes a learning algorithm, called Min-Max LRTA*, to construct an optimal solution for a planning problem, which is a labeled version of LRTA*.

For the functionality composition of web services, we have previous works with a SAT-based approach [11] and abstraction/refinement technique on behavioral description based web services [12]. However, this work is a new approach for QoS requirements using a dynamic programming with learning depth first search.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a novel solution that finds the optimal QoS-driven composition of web services. To identify the optimal solution, our proposal employs Learning-based Depth First Search (LDFS).

Ample directions are ahead for future work. First, we plans to implement efficiently our algorithm and experiment with a number of WSC examples. Second, while our technique uses a state-of-the-art technique, LDFS, note that it permits to use other optimal planning methods such as *value iteration* and *AO* algorithm*. Thus, we plan to compare our proposal with other techniques. Finally, our proposal needs to be extended further to consider not only types of parameters but also full ontologies (in OWL) and reasoning therein.

REFERENCES

[1] R. Hull and J. Su, "Tools for composite web services: A short overview," *SIGMOD Record*, vol. 34, no. 2, pp. 86–95, 2005.

[2] "The web service challenge," <http://ws-challenge.org/>.

[3] B. Bonet and H. Geffner, "An algorithm better than AO*?" in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, 2005, pp. 1343–1348.

[4] B. Bonet and H. Geffner, "Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs," in *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 2006, pp. 142–151.

[5] R. Bellman, *Dynamic Programming*. Princeton University, 1957.

[6] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189–211, 1990.

[7] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81–138, 1995.

[8] S. Koenig, "Minimax real-time heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 165–197, 2001.

[9] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *Proceedings of the 12th International World Wide Web Conference (WWW'03)*, 2003, pp. 411–421.

[10] M. Lin, J. Xie, H. Guo, and H. Wang, "Solving QoS-driven web service dynamic composition as fuzzy constraint satisfaction," in *Proceedings of IEEE International Conference on E-Technology, E-Commerce, and E-Services (EEE'05)*, 2005, pp. 9–14.

[11] W. Nam, H. Kil, and D. Lee, "Type-aware web service composition using boolean satisfiability solver," in *Proceedings of IEEE Joint Conference on E-Commerce Technology (CEC'08) and Enterprise Computing, E-Commerce and E-Services (EEE'08)*, 2008, pp. 331–334.

[12] H. Kil, W. Nam, and D. Lee, "Efficient abstraction and refinement for behavioral description based web service composition," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, 2009.