Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/081L18

On-the-fly Learning-based Search for QoS-aware Web Service Composition

Hyunyoung Kil[1] *and Wonhong Nam*[2,*]

[1] Institute for Ubiquitous Information Technology and Applications, Konkuk University, Seoul 143-701, Korea
[2] Department of Internet & Multimedia Engineering, Konkuk University, Seoul 143-701, Korea

Received: 24 Apr. 2013, Revised: 19 Aug. 2013, Accepted: 20 Aug. 2013
Published online: 1 Apr. 2014

Abstract: The *Web Service Composition (WSC)* is a prominent technique to help software developers to easily build applications on top of the *Service Oriented Architecture*. Given a set of web services and a user request, the aim of the WSC problem is to construct an optimal composite web service to satisfy the request. In this paper, in particular, we study the *Quality of Services (QoS)-aware WSC problem* to optimize the quality criteria of the composite service, e.g., throughput, availability, response time, capacity and accuracy. However, since the QoS-aware WSC problem corresponds to a global optimization problem, it is a hard problem to solve the problem for large scale instances. To resolve this challenge, we propose a novel solution using *on-the-fly learning-based search*. Our technique employs the *Learning Depth-First Search (LDFS)* as an underlying search algorithm, which performs iterated depth-first searches enhanced with learning. Moreover, the on-the-fly searching technique partially constructs a search graph only when the part of the graph is required. We empirically show, with a number of examples, that the proposed on-the-fly technique is able to find the optimal composite services much earlier than our previous work, the off-line LDFS method.

Keywords: Web Service Composition, Quality of Services (QoS), On-the-fly Construction, Learning

1 Introduction

Web services are methods for supporting interoperable machine-to-machine interaction over the Internet. Recently, abundant study has been performed to improve the flexible and dynamic functionalities of the *service oriented architectures (SOA)*, and this effort promotes to define the web service standards. However, various research challenges still remain [1]; for instance, web service discovery/recommendation, service selection and composition, and validation and testing of composed web services. In general, given a set of web services and a user requirement, the *web service composition (WSC) problem* is to find the shortest sequence of web services which meets the user requirement. Recently, one asks service providers to satisfy functional requirements as well as nonfunctional requirements, i.e., *Quality of Services (QoS)* constraints. In this case, users indeed want the composite web service with the optimal QoS value *rather than* the shortest sequence of web services as a solution. This problem is called the *QoS-aware WSC problem*. However, it is computationally hard to identify such a composite web service since it is a global optimization problem. Therefore, as the number of given web services gets increased like real web services on the Web, the problem becomes intractable.

In this paper, we propose a novel solution for the QoS-aware WSC problem, which establishes an optimal composition with respect to given QoS criteria. That is, given a set of web service descriptions including the QoS information of each service and a requirement web service, we construct a sequence of web services such that users are able to call legally the next web service in each step, the execution of the sequence satisfies the desired requirement, and an aggregated QoS value of the sequence is optimal. Our technique first recasts the composition problem into a *graph search problem* on a *weighted state-transition system* where the optimal sequence to a goal state is exactly correspondent with the optimal composite web service with respect to the QoS value. To identify the optimal sequence of the graph search problem, we apply an efficient algorithm using *Learning Depth First Search (LDFS)* [2,3]. However, we have observed a drawback of the LDFS implementation,

* Corresponding author e-mail: wnam@konkuk.ac.kr

in which the method constructs a whole search graph at the beginning even though some part of the graph will not be visited. In addition, for large scale problem instances, the ratio of not-visited space can be large and the whole graph construction needs a significant amount of time. In this paper, therefore, we propose an on-the-fly searching technique to partially build a search graph only when it is needed. The on-the-fly LDFS algorithm for the QoS-aware WSC problem can find optimal composite web services earlier than our previous work [4], the off-line version of LDFS. With experiment on a number of examples by the test set generator employed in Web Services Challenge 2009 [5,6], we show our method's efficiency. In the experiment, the proposed on-the-fly LDFS method always generates the optimal solutions earlier than the offline LDFS.

2 QoS-Aware Web Service Composition

Quality of Services (QoS) includes various non-functional properties of services, e.g., throughput, response time, availability, accuracy and capacity. The QoS value of each web service helps clients select an appropriate service provider for them from a number of candidate providers with same functionality. In this section, we formally define the QoS-aware web service composition (WSC) problem that we study in this paper.

2.1 Example: Movie Theater Reservation System

Consider that a client wants to find and make a reservation for a movie theater, and to get a map to the theater. He searches a movie and a theater by a movie genre (e.g., 'drama') and a city name (e.g., 'Seoul'), and his QoS requirement is the fastest response time. Assume that there is no single web service which finds/reserves a movie theater and provides a map, and there are many different services for movie theaters or maps with various response times. In this case, he wants to combine several web services to achieve his goal and also minimize aggregated response time. Figure 1 illustrates this example. Initially, he sets a movie genre preference (*genre preference*) and a location preference (*city name*) for a movie theater. Given a city name and a genre preference, the Recommend Movie Theater (RMT) service returns a theater address and a movie title. When the Reserve Theater (RT) service receives a theater address and a movie title, it makes a reservation for the theater. On the other hand, the Search Movie Theater (SMT) service can find and reserve a theater for a given city and genre preference. Once the Get Map 1 (GM1) and Get Map 2 (GM2) services receive an address, they both provide a map for a given address. The response time for each web service is as follows: R_{RMT} = 10 msec,

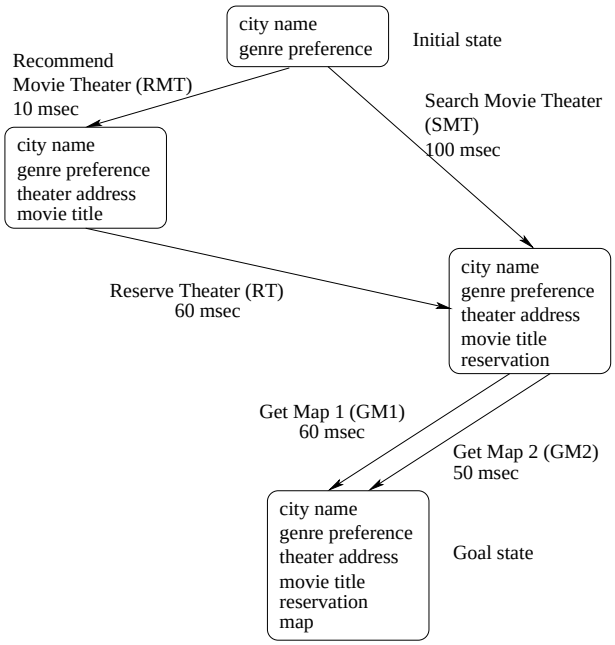


Fig. 1: Movie theater reservation system

R_{RT} = 60 msec, R_{SMT} = 100 msec, R_{GM1} = 60 msec, and R_{GM2} = 50 msec, respectively. In this example, we have four sequences to find/reserve a movie theater and provide a map: i.e., RMT-RT-GM1, RMT-RT-GM2, SMT-GM1, and SMT-GM2. If considering the length of composite web services as our aim, SMT-GM1, and SMT-GM2 would be the best solutions. However, since we want the minimal response time, RMT-RT-GM2 is the best composition (i.e., $R_{RMT-RT-GM2}$ = 120msec).

2.2 QoS-aware Web Service Composition Problem

Now, we formalize the definition of web services with QoS criteria and the QoS-aware composition problem which we consider in this paper. A *web service* is represented as a tuple $w(I,O,Q)$ where:

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

When a web service $w(I,O,Q)$ is called with all the input parameters $i \in I$ with the type t_i, it provides all output parameters $o \in O$ with the type t_o. The invocation of the web service w corresponds to each service quality criterion $q \in Q$, e. g., throughput, response time,

availability and capacity. 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 or equal to t_2 so that t_1 can be replaced with t_2 everywhere. In this case, t_2 is a *supertype* of t_1. Note that 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 $t_1 <: t_3$). Given two web services $w_1(I_1, O_1, Q_1)$ and $w_2(I_2, O_2, Q_2)$, we denote $w_1 \sqsubseteq_I w_2$ if w_1 needs less informative inputs than or the same inputs with w_2; that is, for each $i_1 \in I_1$, there exists $i_2 \in I_2$ such that $t_{i_2} <: t_{i_1}$. 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_2 generates more informative outputs than or the same outputs with w_1; that is, for each $o_1 \in O_1$, there exists $o_2 \in O_2$ such that $t_{o_2} <: t_{o_1}$. These relationships, $\sqsubseteq_I$ and $\sqsubseteq_O$, are both reflexive. Given a set W of web services and a request web service w_r, a *web service discovery problem* is to identify a web service $w \in W$ such that $w \sqsubseteq_I w_r$ and $w_r \sqsubseteq_O w$. Intuitively, the solution web service w asks less inputs than or the same inputs with w_r, and generates more outputs than or the same outputs with w_r.

However, it is sometimes possible that there does not exist a single web service satisfying the user request. In this case, needless to say, users want to search a sequence $w_1 \cdots w_n$ of web services such that they can legally call the next web service in each step and achieve the desired goal eventually. Formally, we extend the relations, $\sqsubseteq_I$ and $\sqsubseteq_O$, to a sequence of web services as follows.

- $w_1 \cdots w_n \sqsubseteq_I w$ (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 $t_{i_1} <: t_{i_2}$. Intuitively, calling $w_1 \cdots w_n$ requires less input parameters than calling w.
- $w \sqsubseteq_O w_1 \cdots w_n$ (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 $t_{o_2} <: t_{o_1}$. In other words, calling $w_1 \cdots w_n$ provides more outputs than calling w.

Finally, given a set W of available web services and a service request w_r, the *QoS-aware WSC problem* $\langle W, w_r \rangle$ is to identify a sequence $w_1 \cdots w_n$ (every $w_j \in W$) of web services such that $w_1 \cdots w_n \sqsubseteq_I w_r$ and $w_r \sqsubseteq_O w_1 \cdots w_n$. Intuitively, calling the sequence $w_1 \cdots w_n$ of web services requires less inputs than or the same inputs with w_r, and generates more outputs than or the same outputs with w_r. The optimal solution for this problem is such a sequence $\sigma = w_1 \cdots w_n$ with the minimal aggregate QoS value $Q(\sigma)$ where the aggregate QoS value $Q(\sigma)$ is inductively computed as follows:

- $Q(\sigma) = c_1 \cdot Q_1(\sigma) + \cdots + c_m \cdot Q_m(\sigma)$ where each c_j is a given weight for the j-th QoS criterion.
- Each function Q_j depends on the corresponding quality criterion. For example, let us consider throughput as the QoS criterion. If $\sigma = w_1$, then $Q_j(\sigma) = th_{w_1}$ where th_{w_1} is the throughput of w_1. Otherwise (i.e., $|\sigma| > 1$), $Q_j(\sigma) = \mathrm{Min}(th_{w_1}, Q_j(w_2 \cdots w_n))$. On the other hand, let us consider response time. If $\sigma = w_1$ (i.e., $|\sigma| = 1$), then $Q_j(\sigma) = rt_{w_1}$ where rt_{w_1} is the response time of w_1. Otherwise (i.e., $\sigma = w_1 \cdots w_n, n > 1$), $Q_j(\sigma) = rt_{w_1} + Q_i(w_2 \cdots w_n)$.

3 Reduction to Graph Search Problem

In this section, we explain our reduction method for the QoS-aware WSC problem into a graph search problem. Given a QoS-aware WSC problem $\langle W, w_r \rangle$, we first recast into a graph search problem on a *weighted state-transition system* $\mathscr{S} = (S, s_0, G, A, T, C)$ where:

- S is a finite set of *states*, where a state corresponds to a node in search graphs.
- $s_0 \in S$ is the *initial state*.
- $G \subseteq S$ is a finite set of *goal states*.
- A is a finite set of *actions*.
- $T : S \times A \to S$ is a *transition function* which maps a current state and an action into a next state.
- For each state $s \in S$ and each action $a \in A$, $C(s, a)$ is the *action cost*.

The graph search problem on the above weighted state-transition system $\mathscr{S} = (S, s_0, G, A, T, C)$ is to identify a sequence of actions $a_1, \cdots, a_n$ from the initial state to a goal state, which has the minimal total cost.

Given a set $W = \{w_1, \cdots, w_n\}$ of available web services where for each j, $w_j = (I_j, O_j, Q_j)$, we denote as TP a set of all types included in W, i.e., $TP = \{t \mid \text{for each } p \in \bigcup_j (I_j \cup O_j), t = type(p)\}$. Now, given a set $W = \{w_1, \cdots, w_n\}$ of available web services and a requirement web service $w_r(I_{w_r}, O_{w_r}, Q_{w_r})$, we can build a weighted state-transition system $\mathscr{S} = (S, s_0, G, A, T, C)$ as follows:

- $S = \{(x_1, \cdots, x_m) \mid x_j = true \text{ or } false, m = |TP|\}$. Each boolean variable x_j means if we possess an instance with the type t_j at a state s.
- $s_0 = (x_1, \cdots, x_m)$, where each x_j is *true* if and only if there is an input parameter $i \in I_{w_r}$ such that t_i is a subtype of its corresponding type t_{x_j} (i.e., $t_i <: t_{x_j}$). If not, x_j is *false*. Remark that since the supertype and subtype relations are reflexive, every type t is its supertype as well as its subtype.
- $G = \{(x_1, \cdots, x_m) \mid$ every x_j is *true* if and only if there exists $o \in O_{w_r}$ such that $t_{x_j} <: t_o\}$.
- $A = W$.
- For $s = (x_1, \cdots, x_m)$, $s' = (x'_1, \cdots, x'_m)$, and $w = (I, O, Q)$, $T(s, w) = s'$ if and only if (1) for all $i \in I$, there exists x_j in s such that x_j is *true* and its corresponding type t_{x_j} is a subtype of the type of i (i.e., $t_{x_j} <: t_i$), (2) if x_k is *true*, x'_k is also *true*, and (3) $\forall o \in O_j$: for every variable x'_j in s', if its corresponding type t_{x_j} is a supertype of t_o, x'_j is *true*. Intuitively, a web service w can be called at a state s if we have data instances that are more informative than inputs of w at the state s. If we call a web service w at such a state s, then we

proceed to a state s' where we preserve all the data instances from the state s and acquire outputs of w and their supertypes.

–For each state s and each web service w, $C(s,w) = c_1 \cdot q_1 + \cdots + c_k \cdot q_k$, where each q_j is j-th QoS criterion value of w.

Intuitively, we have the initial state in which we have all the instances correspondent with the input of w_r and instances correspondent with their supertypes. A state s is a goal state if s includes more informative data instances than or equal to the outputs of w_r. In addition, the optimal path for the graph search problem instance on the above weighted state-transition system exactly corresponds to the optimal composite web service for a given QoS-aware WSC problem instance. Finally, we modify the LDFS algorithm to be appropriate for the QoS-aware WSC problem. That is, we construct only the initial state of a search graph at the beginning. Then, as we explore the rest part of the search graph, we incrementally expand successor states only when they are needed.

4 Learning Depth First Search with On-the-fly Construction

The *Learning Depth First Search (LDFS)* [2,3] is designed to attain the benefits of both a general dynamic programming technique and the effectiveness of heuristic search methods. Its basic idea is to search solutions by applying a learning technique to combining iterative, bound depth first searches [7,8]. In a searching iteration step, if the algorithm finds a solution with a cost which does not exceed a lower bound, then it returns the solution. Otherwise, it repeats the searching process with the updated value function and the lower bound. By generalizing *learning process* and *lower bounds*, the LDFS algorithm can reduce the searching space effectively. In the recent works [2,3], the LDFS technique has shown better execution time in many cases of experiments than other searching techniques, e.g., AO* algorithm, value iteration, and Min-Max LRTA* [9]. However, we observe a drawback of the LDFS algorithm; even though the LDFS method can find a solution after traversing a small portion of the search graph in many cases, it should explicitly construct a whole search graph before searching. Moreover, the graph construction can take a significant amount of time, especially for large sized problem instances. To resolve this challenge, in this paper, we propose on-the-fly construction of a search graph. At the beginning, we build only the initial state. Then, during searching process, our on-the-fly method partially expands a search graph as much as it needs in each searching iteration step.

We assume that there exists an initial value function V which is a *lower bound*, i.e., $V(s) \leq V^*(s)$). Also, this function is *monotonic*, i.e., $V(s) \leq \min_{a \in A} Q_V(s,a)$ for every non-terminal state. Therefore, *learning process* [7, 8] updates this function by an operation in the form of a Bellman update $V(s) := \min_{a \in A} Q_V(s,a)$. The underlying idea of LDFS algorithm is to search a state reachable from a given initial state with the strategy σ such that $V(s) < \min_{a \in A} Q_V(s,a)$, and update $V(s)$. We note that if $V(s)$ is equal to $\min_{a \in A} Q_V(s,a)$ for every state s reachable from an initial state, s_0, with the corresponding strategy, $V(s)$ is the optimal. We will say that a state s of which value function $V(s)$ is equal to $\min_{a \in A} Q_V(s,a)$ is *consistent*. Otherwise (i.e., $V(s) < \min_{a \in A} Q_V(s,a)$ for every action a), a state s is *inconsistent*. To efficiently discover a consistent state, the LDFS algorithm uses the depth-first search, and repeats the searching iteration until no consistent state exist.

```
WS_seq QoS_WSC(WS_Set W, Req_WS w_r){
1:   ConstructPartially(W, w_r);
2:   do done := On_the_fly_LDFS(s_0) while (done = false);
3:   Translate_to_WS_Sequence(σ, solution_seq);
4:   return solution_seq;
}

Boolean On_the_fly_LDFS(state s){
5:   if s.flag = terminal then {
6:       V(s) = 0;
7:       s.flag := solved;
8:   }
9:   if s.flag = solved then return true;
10:  done = false;
11:  foreach a ∈ A do {
12:      if Q_V(s,a) > V(s) then continue;
13:      done := true;
14:      Successors := ExpandSuccessors(s, a);
15:      foreach t ∈ Successors do {
16:          done := LDFS(t) ∧ (Q_V(s,a) ≤ V(s));
17:          if ¬ done then break;
18:      }
19:      if done then break;
20:  }
21:  if done = true then {
22:      σ(s) := a;
23:      s.flag := solved;
24:  } else V(s) := min_{a∈A} Q_V(s,a);
25:  return done;
}
```

Fig. 2: QoS-driven WSC algorithm using LDFS

Figure 2 presents our on-the-fly algorithm for the QoS-aware WSC problem. Given a set W of web services and a requirement web service w_r, we construct a partial weighted state-transition system as much as it needs, i.e., *not* the whole search graph *but* the initial state s_0 with the necessary information such as a set of goal states, and a transition function (line 1). After this partial construction, we repeat the on-the-fly LDFS procedure with s_0 until identifying the optimal strategy σ (line 2) which is translated to the sequence of the web services (line 3). The on-the-fly LDFS procedure considers all possible

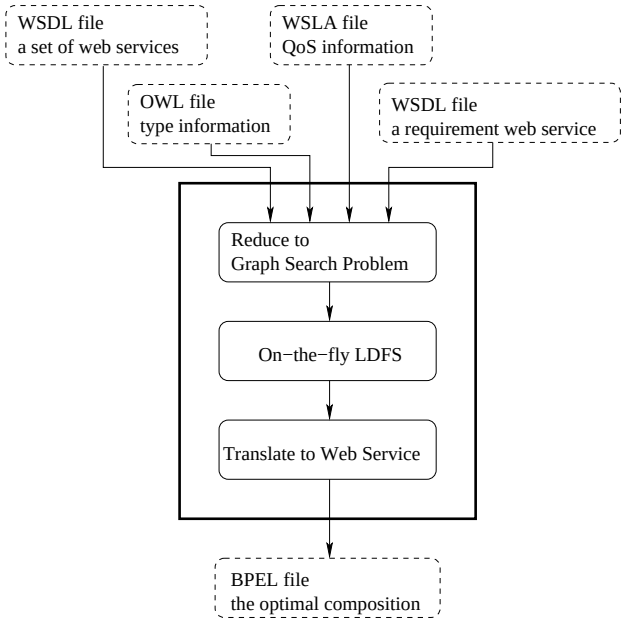


Fig. 3: Architecture diagram

actions in a given state, backtracks on unsolved states, and updates both the unsolved states and their predecessors. During this process, the LDFS procedure includes two loops; for all possible actions a on the given state s (line 11) and for all successor states t which is generated by $T(s,a)$ (line 15). Again, our on-the-fly method expands a set of successors only when they are needed (line 14). When the algorithm reaches to an inconsistent state, a terminal state, or a state labeled as solved, it terminates. Therefore, 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* (line 23). In this case, the algorithm sets the strategy σ for the state s to a, and skips the other actions. Then, LDFS procedure returns *true*. Otherwise, it continues searching with another action until there is no action left. If there exists no action which satisfies the above condition, the state s is updated. Then, the LDFS procedure returns *false*. Similarly, if the search on a successor state $t \in T(s,a)$ returns false or no action a satisfies the condition $(Q_V(s,a) \leq V(s))$, the algorithm skips searching in the rest of successors.

As a summary, Figure 3 presents the architecture of our tool. As inputs, our tool takes a set of web services and a requirement web service in WSDL, parameter type information in OWL, and QoS specification in WSLA. It reduces the QoS-driven WSC problem into a weighted graph search problem, and then tries to find an optimal strategy by the on-the-fly LDFS method. Finally, if our tool finds the solution, it returns a BPEL file for the optimal composition to which the strategy is translated.

5 Experiment

We have implemented an efficient tool for the QoS-aware WSC problem employing the LDFS algorithm and the on-the-fly construction. Given a set of web service descriptions and a requirement web service description in WSDL, their QoS information in WSLA, and their parameter type information in OWL, our tool automatically identifies the composite web service with the minimal QoS value. To validate that our tool efficiently identifies the optimal solutions, we have compared the on-the-fly LDFS method with the offline version of LDFS method which is our previous work [4]. The experiment has been performed on a number of examples which are generated by the test set generator employed in Web Services Challenge 2009 [5,6].

We have performed all experiments on a PC using a 2.93GHz Core i7 processor, 8GB memory and a Linux OS. Table 1 describes the total number of web services and their parameters, and the length of the optimal composite web service for each problem instance. It also presents the execution time in seconds for the offline LDFS method and the on-the-fly LDFS method. In the execution time column, '–' means that the corresponding method cannot complete in 1,200 seconds. For the on-the-fly method, the table presents the search graph construction ratio, i.e., the number of states that the on-the-fly method constructs over the number of states that the offline method constructs. If any method cannot terminate within the given time-out, we mark '–' for the construction ratio since we cannot compare them. The experiment result has shown that our on-the-fly LDFS technique always outperforms the offline LDFS method in terms of execution time. In the cases where the graph construction ratio is very small (e.g., P_3: 5%, P_9: 1%, P_{12}: 5%, P_{14}: 0.3%), the on-the-fly method has completed earlier than the offline method by one to three orders of magnitude. Even for the cases where the graph construction ratio is relatively large (e.g., P_5: 69%, P_7: 96%, P_{14}: 50%), the on-the-fly method still has shown better performance than the offline method. For P_6, P_{16}, P_{17}, P_{18}, P_{19}, and P_{20}, our on-the-fly method has converted infeasible problems into feasible ones.

6 Related Work

In recent years, a number of efforts have been carried to standardize web service specifications, especially, non-functional description such as QoS. WS-Policy [10] represents a set of specifications that allows web services to describe their service policies on functionality, requirements, quality of service, and other constraints. On the other hand, Web Service Level Agreement (WSLA) framework [11] defines assertions of a service provider based on agreed QoS parameters and exception-dealing methods to meet the asserted service guarantees. WS-Agreement standard [12] that the grid resource

Table 1: Experiment result

Problem	WS	Parameter	Solution length	Offline LDFS	On-the-fly LDFS	Const. ratio
P_1	25	300	8	203.33	54.54	49%
P_2	25	300	10	14.70	1.69	28%
P_3	25	500	6	92.57	1.15	5%
P_4	25	500	7	16.18	4.80	43%
P_5	25	500	9	189.80	92.64	69%
P_6	50	500	7	–	42.27	–
P_7	50	500	9	4.89	4.37	96%
P_8	30	1,000	6	10.67	0.21	7%
P_9	40	1,000	6	824.12	1.57	1%
P_{10}	50	1,000	6	167.24	6.04	12%
P_{11}	50	1,000	8	470.56	21.33	13%
P_{12}	50	1,000	10	1121.17	16.94	5%
P_{13}	30	2,000	4	6.83	0.14	9%
P_{14}	50	2,000	5	806.79	0.34	0.3%
P_{15}	50	2,000	8	41.41	13.35	50%
P_{16}	30	5,000	7	–	3.55	–
P_{17}	50	5,000	9	–	176.54	–
P_{18}	80	5,000	5	–	2.19	–
P_{19}	100	5,000	3	–	22.68	–
P_{20}	100	5,000	5	–	238.21	–

allocation agreement protocol working group has developed is targeted to negotiate and manage services based on QoS attributes. Based on these standards, the problem to find the composite web service with the optimal QoS value also has received the attention. Zeng et al. [13] suggest a linear programming method for this problem. Lin et al. [14] reduce the service selection problem to a fuzzy constraint satisfaction problem, and then search the solution by using deep-first branch-and-bound method. As related work to the LDFS algorithm, Korf [7] has presented a Real-Time-A* algorithm by applying the two-player game assumptions, and then LRTA* algorithm which shows better performance over successive problem by learning more accurate heuristic values from previous trials. Koenig [9] suggests a Min-Max LRTA* algorithm to construct an optimal solution for a planning problem. In addition, other studies on the LDFS algorithm exist. However, to the best of our knowledge, there is no research to apply the on-the-fly construction of LDFS algorithm for the QoS-aware WSC problem. While Bertoli et al. [15] suggest an on-the-fly belief space search method for the behavior-description based WSC problem, since this technique does not consider the QoS property, it cannot be directly applied to the QoS-aware WSC problem in this paper. Kil et al. [17,18] devised efficient methods by using various techniques, but they consider only the service functionality. Therefore, this work is the first trial for QoS specification using the on-the-fly depth first search with the learning technique. Recently, Kil and Nam [16] propose to apply an anytime algorithm to the QoS-driven web service composition problem which can identify composite services with high quality much earlier than optimal algorithms.

7 Conclusions

We have proposed a novel algorithm which finds the web service composition with the optimal QoS value. To identify the optimal solution, our proposal employs LDFS with the on-the-fly construction. Still there are various directions for future work. First, while we use a state-of-the-art searching algorithm, i.e., LDFS, in this work, we can apply other planning methods, e.g., *search with value iteration* or *AO* search algorithm*. Therefore, we plan to apply other techniques and compare our proposal with them. Second, we can extend this proposal to consider ontologies such as OWL as well as parameter types. Then, we can devise other alternatives by using reasoning therein. Last, we can consider approximate methods for a large-sized QoS-aware WSC problem since it can be hard to find the optimal solution in large-sized optimization problem instances.

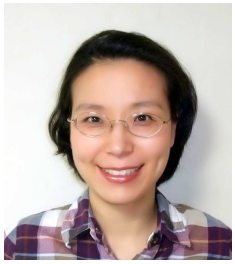
Acknowledgement

This paper was supported by Konkuk University in 2012.

References

[1] R. Hull and J. Su, SIGMOD Record, **34**, 86-95 (2005).

[2] B. Bonet and H. Geffner, Proceedings of the National Conference on Artificial Intelligence, **20**, 1343-1348 (2005).
[3] B. Bonet and H. Geffner, Proceedings of the International Conference on Automated Planning and Scheduling, **16**, 172-151 (2006).
[4] W. Nam, H. Kil, J. Lee, Proceedings of the IEEE Conference on Commerce and Enterprise Computing, **11**, 507-510 (2009).
[5] The Web Service Challenge (2009), http://ws-challenge.georgetown.edu/wsc09/.
[6] S. Kona, A. Bansal, M. Blake, S. Bleul, T. Weise, Proceedings of IEEE Conference on Commerce and Enterprise Computing, **11**, 487-490 (2009).
[7] R. E. Korf, Artificial Intelligence, **42**, 189-211 (1990).
[8] A. G. Barto, S. J. Bradtke, S. P. Singh, Artificial Intelligence, **72**, 81-138 (1995).
[9] S. Koenig, Artificial Intelligence, **159**, 165-197 (2001).
[10] W3C, Web services policy (WS-Policy) version 1.2 (2006), http://www.w3.org/Submission/WS-Policy/.
[11] IBM Corporation, WSLA (Web Service Level Agreements) project (2004), http://www.research.ibm.com/wsla/documents.html.
[12] OGF, Web services agreement specification (WS-Agreement) (2007), http://www.ogf.org/documents/GFD.107.pdf.
[13] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Z. Sheng, Proceedings of the International World Wide Web Conference, **12**, 411-421 (2003).
[14] M. Lin, J. Xie, H. Guo, and H. Wang. Proceedings of IEEE International Conference on E-Technology, E-Commerce, and E-Services, 9-14 (2005).
[15] P. Bertoli, M. Pistore and P. Traverso. Proceedings of the International Conference on Automated Planning and Scheduling, **16**, 358-361 (2006).
[16] H. Kil and W. Nam, International Journal of Web and Grid Services, **9**, 82-106 (2013).
[17] W. Nam, H. Kil, D. Lee, Proceedings of IEEE Joint Conference on E-Commerce Technology. and Enterprise Computing, E-Commerce and E-Services, 331-334 (2008).
[18] H. Kil, W. Nam, D. Lee. International Journal of Web and Grid Services, **9**, 54-81 (2013).



Hyunyoung Kil received the B.S. and M.Sc. degrees from Korea University, Seoul, Korea, in 1998 and 2001, respectively. She received the M.S.E. degree from the University of Pennsylvania, Philadelphia, PA, USA in 2003, and the Ph.D. degree from the Pennsylvania State University, State College, PA, USA in 2010. She is currently a researcher professor of Konkuk University, Seoul, Korea. Her research interests include automated planning, web services composition, SOA and web sciences.

Wonhong Nam received the B.S. and M.Sc. degrees from Korea University, Seoul, Korea, in 1998 and in 2001, respectively, and the Ph.D. degree from the University of Pennsylvania, Philadelphia, PA, USA in 2007. From 2007 to 2009, he was a postdoctoral researcher with the College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA. He is currently an associate professor of the Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea. His research interests include formal methods, formal verification, model checking, automated planning, and web services composition.