Anytime Algorithm for QoS Web Service Composition *

Hyunyoung Kil
KAIST, Daejeon 305-701, Korea
hkil@kaist.ac.kr

Wonhong Nam
Konkuk University, Seoul 143-701, Korea
wnam@konkuk.ac.kr

ABSTRACT

The *QoS-aware* web service composition (WSC) problem aims at the automatic construction of a composite web service with the optimal accumulated QoS value. It is, however, intractable to solve the QoS-aware WSC problem for large scale instances, since the problem corresponds to a global optimization problem. In this paper, we propose a novel *anytime algorithm* for the QoS-aware WSC problem to identify composite web services with high quality much earlier than an optimal algorithm and the *beam stack search* [3].

Categories and Subject Descriptors
H.3.5 [Information Storage and Retrieval]: Online Information Services—Web-based services
General Terms: Algorithms, Performance
Keywords: Anytime algorithm, Quality of Services (QoS), Web service composition

1. Introduction

In the *QoS-aware web service composition (WSC) problem*, one indeed desires, as a solution, *not* the shortest sequence of web services *but* a composite web service with the optimal accumulated QoS value. Since it is computationally hard to identify such a composite web service, the problem is intractable for large scale problem instances, as web services in the real world. To resolve this challenge, we propose a novel *anytime algorithm* for the QoS-aware WSC problem. While traditional algorithms cannot provide an answer until completely terminating a fixed number of computations, anytime algorithms always provide best-so-far answers and improve the quality of answers along with execution time. If enough time is allowed, anytime algorithms will provide the optimal answer eventually. By using our anytime algorithm, we can identify composite web services with high quality much earlier than an optimal algorithm as well as the *beam stack search* [3] which is a state-of-the-art anytime algorithm in AI literature. To the best of our knowledge, there is no work to employ an anytime algorithm to WSC problems.

2. QoS-aware Web Service Composition

Suppose that clients want to reserve a trip including *all* of flights, ground transportation and a hotel with the *fastest response time*. However, there does not exist a single web service that provides all the services, but there are a number of reservation web services for some of the services with different response time. Figure 1 illustrates this example. The TR service deals with the reservation for both of flights and ground transportation. On the other hand, FR and GTR treat a request only for flights and ground transportation, respectively. HR1 and HR2 make a reservation for hotel rooms. The response time for each web service is presented in Figure 1. If considering the length of composite web services as our aim, TR-HR1 and TR-HR2 would be the best compositions. However, since the minimal response time is our goal, FR-GTR-HR2 is the optimal composition (i.e., $R_{FR\text{-}GTR\text{-}HR2} = 180\,msec$).

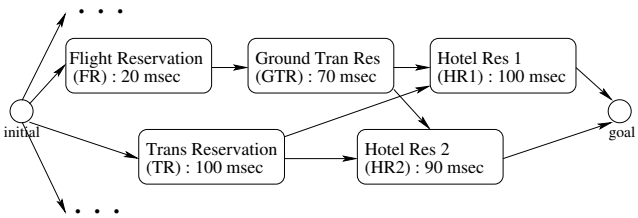

Figure 1: Travel agency system

Now, we formalize the notion of web services with QoS criteria and the QoS-aware WSC problem. A *web service* w is a tuple (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.

In the QoS-aware WSC problem, we identify a sequence $w_1 \cdots w_n$ of web services such that we can invoke the next web service in turn and achieve the desired requirement eventually. Given a web service w and a sequence $w_1 \cdots w_n$ of web services, we denote $w \sqsupseteq_I w_1 \cdots w_n$ if calling $w_1 \cdots w_n$ in turn requires less inputs than w does, and $w \sqsubseteq_O w_1 \cdots w_n$ if invoking $w_1 \cdots w_n$ produces more outputs than w_1 does. Then, given a set W of web services and a service request w_r, the *QoS-aware WSC problem* $\langle W, w_r \rangle$ is to find a sequence $w_1 \cdots w_n$ of web services (where each $w_i \in W$) such that $w_r \sqsupseteq_I w_1 \cdots w_n$ and $w_r \sqsubseteq_O w_1 \cdots w_n$, and the aggregate QoS value $Q(w_1 \cdots w_n)$ is minimal. Given a sequence $\sigma = w_1 \cdots w_n$, $Q(\sigma)$ is computed as follows (assume that there are m QoS criteria):

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

*This research was supported by the MAKE, Korea, under the ITRC support program supervised by the NIPA (NIPA-2010-C1090-1001-0008).

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

3. Anytime Algorithm for QoS-aware WSC

We propose an anytime algorithm for the QoS-aware WSC problem, which is based on the beam stack search [3]. The beam stack search is an anytime algorithm that has shown an excellent performance in AI literature. In the beam stack search, due to its backtracking mechanism, a bad decision in early phases requires more searching space than a bad selection in late phases does. However, the algorithm assigns a fixed value to the beam width for each search level, and thus it considers the same number of candidates in all the steps. To resolve this problem, we propose to *dynamically* assign larger beam widths to early phases to consider more qualified candidates. Then, we decrease its size gradually as going down to a search graph. This idea is influenced by a following natural intuition. When finding out an unknown path from a source to a destination, we do not know whether we would north, south, east or west, especially at the beginning. We hence consider more candidates in the early phases, and a good choice at early stages will lead to a good solution more quickly.

To determine the beam width values, the number of nodes in each level of a search graph is a major factor. We first compute the number of successors of the root node (i.e., level 0) called n_{succ}. We approximate the number of successors of each node as n_{succ}. Based on the approximation, we decide the beam width $BW(i)$ for each level i as following:

- $BW(1) = n_{succ}$. We consider all nodes for the most careful choice at the highest level.
- $BW(2) = R_{inc} \cdot BW(1)$, where R_{inc} is the increasing rate for the level 2 ($1 < R_{inc} < n_{succ}$); in our experiment, we take 1.5 as R_{inc}.
- For $i \geq 3$, $BW(i) = R_{dec} \cdot BW(i-1)$, where R_{dec} is the decreasing rate ($0 < R_{dec} < 1$); in our experiment, we take 0.9 as the decreasing rate. Since choices at bottom levels are not relatively significant, we decrease the beam width by a user provided constant R_{dec}.
- wd_{min} is the minimum value of a beam width where $0 < wd_{min} < n_{succ}$; we take $0.3 \cdot n_{succ}$ as wd_{min}. This value is needed, since if a beam width is too small, it induces excessive backtracking and retrial. Finally, for every i, if $BW(i) < wd_{min}$, then $BW(i) = wd_{min}$.

In addition to the dynamic beam width, we propose two heuristics to improve the beam stack search: *short backtracking* and *upper bound propagation*. By avoiding the duplicated state expansion and unnecessary state inclusion, these techniques help to efficiently find approximate solutions with better QoS values. We omit the detail for our algorithm and the heuristics for the sake of space.

To demonstrate that our proposed algorithm quickly identify good solutions, we compare it with an optimal algorithm, *uniform cost search algorithm* [2] and a state-of-the-art anytime algorithm, *the beam stack search* [3]. We have experimented on five WSC problems produced by the TestSetGenerator which the Web Service Challenge 2009 competition [1] provides. All experiments have been performed on a PC using a 2.33GHz Core2 Duo processor, 1GB memory and a Linux operating system.

Figure 2 presents how our anytime algorithm works in solving a QoS-WSC problem (i.e., P3 in Table 1) by comparing an optimal algorithm, where Y-axis is the solution quality ratio α (i.e., the QoS value of the optimal solution vs. the QoS value of the anytime algorithm's solution). While

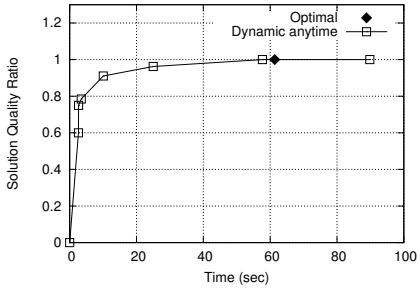

Figure 2: Optimal algo. vs. dynamic anytime algo.

Table 1: Experiment result

	Timeout (sec)	Opt. algo	Beam stack	Qual. ratio	**Dyn. anytime**	**Qual. ratio**
P1	60	–	I	0.84	**I**	**1**
P2	60	–	I	0.81	**I**	**1**
P3	60	–	I	0.96	**I**	**1**
P4	60	–	I	0.68	**I**	**0.81**
P5	60	–	**I**	**0.85**	**I**	**0.85**
P1	120	**S**	I	1	**C**	**1**
P2	120	**S**	I	0.94	I	**1**
P3	120	**S**	I	0.96	**C**	**1**
P4	120	–	I	0.68	**I**	**0.81**
P5	120	–	I	0.85	**I**	**0.96**
P1	300	**S**	**C**	**1**	**C**	**1**
P2	300	**S**	**C**	**1**	**C**	**1**
P3	300	**S**	**C**	**1**	**C**	**1**
P4	300	–	I	0.81	**I**	**0.87**
P5	300	–	**I**	**0.96**	**I**	**0.96**

the optimal algorithm finds the optimal solution at 61.4 seconds (see ◆ in Figure 2), our anytime algorithm identifies several solutions with high quality (i.e., $0.9 < \alpha$) at 10.1, 25.0, and 57.7 seconds much earlier than the optimal algorithm. Even, it finds solutions with moderate quality (i.e., $0.6 < \alpha \leq 0.9$) very quickly (i.e., at 2.6, 2.7, and 3.5 seconds). Table 1 presents the comparison of three algorithms with specific timeouts. In the case of the optimal algorithm, since it returns the optimal answer only, we report only whether a given problem is solved ('S') or not ('–'). For the beam stack search and our anytime algorithm, to declare that the solution found is optimal, sometimes they need to explore more search space even after they have found it. 'C' and 'I' indicate if the algorithms complete the confirmation step. Within 60 seconds, while the optimal algorithm does not provide any answer at all, our algorithm produces good solutions of which QoS quality ratios are from 0.81 to 1. In 120 seconds, while the optimal algorithm solves 3 problems, our algorithm completes 2 problems and returns 3 solutions with high quality from 0.81 to 1. In 300 seconds, the optimal algorithm still cannot solve 2 problems, but our algorithm keeps improving the solution quality. In all the cases, our dynamic anytime algorithm outperforms the beam stack search.

4. Conclusion

We have proposed a novel anytime algorithm for the QoS-aware WSC problem. Our preliminary experiment has shown promising results. As future work, we plan ample experiment to validate our algorithm for various problem instances. Adaptive control of the beam width for a given problem instance is also an interesting issue.

5. REFERENCES

[1] The web service challenge. `http://ws-challenge.org/`.
[2] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.
[3] R. Zhou and E. A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, pages 90–98, 2005.