Redundant Service Removal in QoS-Aware Service Composition

Min Chen
Dept. of Computer Science and Software Engineering
Concordia University
Montreal, Canada
Email: minchen2008halifax@yahoo.com

Yuhong Yan
Dept. of Computer Science and Software Engineering
Concordia University
Montreal, Canada
Email: yuhong@encs.concordia.ca

***Abstract*—QoS-aware service composition is the generation of a business process to fulfill functional goals and optimize the QoS criteria at the same time. People may focus on the optimization of a single QoS criterion or a set of QoS criteria. We find that though many composition algorithms can get the optimal QoS values, the solutions obtained can possibly contain redundant services, the removal of which does not worsen the QoS value of the solution. In the literatures using WSC open data sets, the removable services can be over 30% of the services in the solutions. This common problem has been ignored so far. The fundamental reason is that execution cost is not one of the criteria to optimize. Even in the cases when execution costs are not explicitly given for each service, we are still motivated to reduce the number of services in the final solution by assuming each service takes unit cost. In this paper, we study the redundancy removal problem to further optimize the QoS optimal solutions obtained by other algrithms. We model the redundancy removal problem as an integer programming problem. Though solvable using a standard solver, we present an algorithm to solve the problem in this specific context and it proves to have better performance than a standard integer programming solver. We also present the results of our data experiments.**

***Keywords*-Redundant service removal; QoS-aware service composition;**

I. Introduction

A *web service* is a self-describing software module designed to complete tasks on behalf of a user or application. Available web services are posted across the Internet using a set of open standards such as SOAP [14], WSDL [15], and UDDI [9]. With these open standards, web services are invokable and interoperable. When a single service cannot satisfy functional requirements, a composite service to fulfill the functional requirements is necessary. Automated Service Composition (ASC) is the generation of a business process to fulfill functional goals that cannot be fulfilled by individual services. Quality of Service (QoS) is the non-functional property of services. In addition to satisfying functional goals, recent research moves towards the optimization of QoS. This triggers the research area of QoS-aware service composition, whose objective is to find a composite service that satisfies the functional goals and optimizes QoS criteria.

The QoS-aware service composition problems can be classified into single QoS criterion problems or multiple QoS criteria problems. To solve multiple QoS criteria problems, people use a preemptive model where multiple single criterion problems are solved in a sequence of priorities or a non-preemptive model where an aggregated score is calculated as a single criterion to optimize. Sometimes, execution cost is not in the list to optimize. In this case, many composition algorithms can get the solutions with optimal QoS values. However, the solutions obtained can possibly contain redundant services, the removal of which does not worsen the QoS value of the solution. For example, in the literatures using WSC open data sets, *e.g.,* [8], [16], and [18], we find that all the algorithms generate redundant services with almost every data set. The removable services can be over 30% of the services in a solution in some cases. Surprisingly, this very common problem is ignored by all the WSC participants and is not considered in the evaluation of the WSC results. In reality, even if execution cost is not under consideration, to reduce the number of services included in the solution without worsening the QoS performance is a reasonable requirement. This is our motivation to do this research.

In this paper, we study the redundancy removal problem to further optimize the QoS optimal solutions obtained by other algorithms. For simplicity, we consider response time or throughput as the first optimal criterion in this paper. Then execution cost can be considered as the second criterion in the preemptive model. We model the redundancy removal problem as an integer programming problem. Though solvable using a standard solver, we present an algorithm to solve the problem in this specific context and it proves to have better performance than a standard integer programming solver. We also present the results of our data experiments. Experimental results show that our method can find the cost-optimized solution by removing redundant services and the optimal response time (or throughput) of the solution is guaranteed.

The organization of this paper is as follows: Section II introduces the preliminary knowledge and a motivation example to explain redundant services. In Section III, we propose a redundant service removal algorithm. First, we analyze the redundant service removal problem in Section III-A. Based on the analysis, the redundant service removal problem is

modelled as an integer programming problem and we propose a redundant service removal algorithm in Section III-B. The experimental results of our proposed algorithm are shown in Section IV. Section V reviews some related work. Finally, Section VI concludes the article.

II. Preliminary knowledge and motivation

A. The QoS-aware service composition problem

Automatic Service Composition is studied under different assumptions [10], [11]. The most useful and practical problem is to connect SOAP services into a network by matching their parameters, so that this network of services can produce a set of required output parameters given a set of input parameters. In this paper, we take the services as stateless black boxes (no conversations). The services expose themselves in WSDL descriptions which do not include state information. We can also associate semantic information to inputs and outputs using SAWSDL [13] or OWL [12].

Definition 1: Given a set $\mathcal{C}$ of concepts, a **service** w is a tuple $(w_{in}, w_{out}, Q(w))$, where $w_{in} \subseteq \mathcal{C}$ (resp. $w_{out} \subseteq \mathcal{C}$) denotes the inputs (resp. the outputs) of w, and $Q(w) = \{w_{q_i} | 1 \leq i \leq k$ and k is the number of quality criterion$\}$ is a finite set of quality criteria for w.

The above definition assumes each service has one operation. For a service w with n operations $o_1, \ldots, o_n$ we may do as if we had n services $w : o_1, \ldots, w : o_n$.

One output of one service could become one input of another service, if they are exactly the same concept, or the input concept subsumes the output concept. The network of web services can be built based on WSDL and OWL ontology. We use $\sigma = w_1, w_2, \ldots, w_n$ to represent a network of connected services. If they are connected in sequence, $\sigma = w_1; w_2; \ldots; w_n$, or in parallel, $\sigma = w_1 || w_2 || \ldots || w_n$.

We consider response time, throughput and execution cost as QoS criteria in this paper, because they are the most frequently used ones. For a service network σ, they can be computed as following [6], [2]:

- **Response time** $Q_1(w)$: the time interval between the receipt of the end of transmission of an inquiry message and the beginning of the transmission of a response message to the station originating the inquiry.

$$Q_1(w_1; \ldots; w_n) = \sum Q_1(w_i) \quad (1)$$

$$Q_1(w_1 || \ldots || w_n) = \max Q_1(w_i) \quad (2)$$

- **Throughput** $Q_2(w)$: the average rate of successful message delivery over a communication channel, *e.g.,* 10 successful invocations per second.

$$Q_2(w_1; \ldots; w_n) = \min Q_2(w_i) \quad (3)$$

$$Q_2(w_1 || \ldots || w_n) = \min Q_2(w_i) \quad (4)$$

- **Execution cost** $Q_3(w)$: the fee to invoke w.

$$Q_3(w_1, \ldots, w_n) = \sum Q_3(w_i) \quad (5)$$

Response time and execution cost are so called negative criteria, *i.e.,* the higher the value, the lower the quality. Throughput is a positive criterion, *i.e.,* the higher the value, the higher the quality.

When the objective of composition problem is to satisfy business goals and optimize the QoS as well, it is called QoS-aware service composition problem.

Definition 2: **A service composition problem** is a tuple $(W, \mathcal{D}_{in}, \mathcal{D}_{out}, Q)$, where W is a set of services, $\mathcal{D}_{in}$ are provided inputs, $\mathcal{D}_{out}$ are expected outputs, and Q is a finite set of quality criteria.

B. A motivating example

Our previous work [18] combined a planning algorithm, namely the GraphPlan [3], with Dijkstra's algorithm to solve the single QoS criterion service composition problem. GraphPlan is used to satisfy functional goals, while Dijkstra's algorithm systematically searches the Planning Graph to find a QoS optimal solution. We extended Dijkstra's algorithm from handling a single source graph to handling a multiple sources graph as the Planning Graph. We use the following example to briefly introduce this method.

Table I
Services in the solution shown in Figure 2

Service	inputs	outputs	response time	cost
w_1	A	D, E	40	20
w_2	B	F	20	30
w_3	C	G, I	120	30
w_4	D	N, H	30	20
w_5	E, F	H, I	70	30
w_6	G	M, J	100	50
w_7	G	J	120	20
w_8	H	K	50	30
w_9	I	L	20	30
w_{10}	J	Q	30	30

Example 1: A set of available services are presented in Table I. A composition query is $(\mathcal{D}_{in}, \mathcal{D}_{out}) = (\{A, B, C\}, \{K, L, J\})$, and its objective is to optimize response time. Figure 1 shows the Tagged Planning Graph (TPG) for this service composition problem. For clarity, we do not draw the duplicated services if they have appeared in the previous action layers. In Figure 1, each service is labelled with its response time, and each proposition is labelled with the optimal response time to obtain it. The computation of the optimal response time of each proposition is similar to the classic Dijkstra's algorithm, except now the optimal value depends on several inputs. The solution with optimal response time $\{(w_1 || w_2 || w_3); (w_4 || w_5 || w_6); (w_8 || w_9)\}$ is generated by retrieving the recorded best parents for each goal. The TPG for the solution is shown in Figure 2. The response time of the solution is 220 because it is the maximum response time to obtain the individual goals. The execution cost of the solution is 240 by adding up all the costs of the services in the solution. The bold arrows in

Figure 1 and 2 are the optimal paths to get $\mathcal{D}_{out}$. *no-op* is a dummy activity to keep the service layers and the parameter layers interleaving. *no-op* takes 0 time and 0 cost to execute.

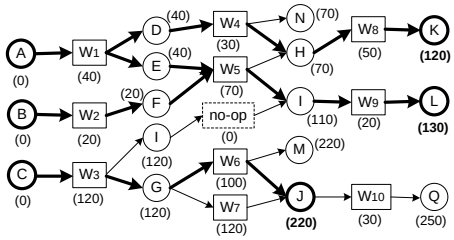

Figure 1. The tagged planning graph for example 1

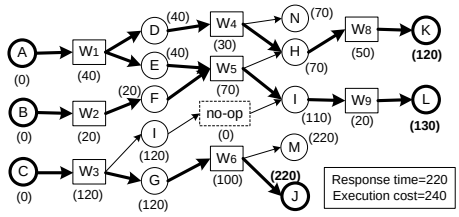

Figure 2. The TPG for the solution for example 1

The service composition problem is an AI planning problem without negative effects. It is known that it is a polynomial time problem [7]. It is also known that it takes polynomial time to construct a Planning Graph [3]. In [18], our algorithm can solve a single QoS criterion, *i.e.,* throughput or response time, service composition problem with throughput or response time in $O(|w|^2)$ time complexity[1], where w is the set of services.

Example 2: Let us check the solution in Figure 2. When the total execution cost for the solution is taken into consideration, we find it is possible to reduce the total cost without worsening the optimal response time by removing some services. Please check the following two cases:

Case 1: w_4 is removed. The final solution becomes $\{(w_1||w_2||w_3);(w_5||w_6);(w_8||w_9)\}$. The total cost of the solution is reduced to 220 while response time of the solution remains 220.

Case 2: w_5 and w_2 are removed. The final solution becomes $\{(w_1||w_3);(w_4||w_6);(w_8||w_9)\}$. The total cost of the solution is reduced to 180 while response time of the solution remains 220.

Our previous work [18] detected the existence of redundant services in an optimal solution. The optimal solution is valid, because it satisfies the functional goals, and has an optimal QoS value. However, it is necessary to remove the redundant services, because the removal can reduce the execution cost. After checking the papers using the same data sets, e.g. [8] and [16], we found this is a common problem. Surprisingly, people seem satisfied after obtaining a solution with the optimal QoS value, without further studying this problem. We consider the fundamental reason to this problem is that execution cost is not one of the criteria to optimize. In reality, even if execution cost is not under consideration, to reduce the number of services included in the solution without worsening the QoS performance is a reasonable requirement.

Intuitively, the redundant services are due to the reproduction of the outputs by multiple services. For example, output H is produced by both w_4 and w_5, which brings the possibility to remove one of them. However, due to the complexity of how the outputs of w_4 and w_5 are used, it is difficult to make a decision without checking the whole solution carefully.

In this paper, we want to find a method to use the execution cost as a second criterion to further optimize the solution obtained from optimizing the first criterion[2]. Our method can be extended to solve more general multiple criteria optimization problems with a preemptive model.

III. Redundant service removal in QoS-aware service composition

A. Analysis of redundant service removal

We extend the definition of a Direct Acyclic Graph (DAG) $G = (V, E)$ to represent a solution to a QoS-aware composition query.

Definition 3: An Extended Direct Acyclic Graph (EDAG) $EG = (V, E)$ is a direct acyclic graph with alternating levels of vertices V_p and vertices V_s, where $V = V_p \cup V_s$ is the vertex set and $E = (V_p \times V_s) \cup (V_s \times V_p)$ is the edge set.

We can label the levels starting from level 0 for the EDAG, where P_i is a level of vertices V_p and S_i is a level of vertices V_s.

We map a solution to an EDAG in the following way:

- The vertices V_p are the parameter vertices.
- The vertices V_s are the service vertices.
- The edges $V_p \times V_s$ connect the input parameters with the services.
- The edges $V_s \times V_p$ connect the services with their output parameters.
- The initial input of the solution is considered to be a dummy service with $\emptyset$ as its inputs and $\mathcal{D}_{in}$ as its outputs.
- The expected output of the solution is considered to be a dummy service with $\mathcal{D}_{out}$ as its inputs and $\emptyset$ as its outputs.

Example 3: Figure 3 shows the EDAG for the solution in Example 1. D_I and D_G are the two dummy services.

In an EDAG, if a parameter is not used as an input of another service, *e.g.,* N in Figure 3, the parameter is not

[1]Other constraints apply for the other single criteria.

[2]Limit to throughput and response time in this paper

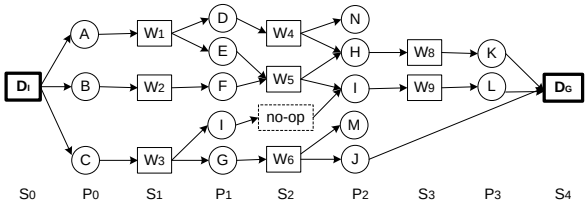

Figure 3. An EDAG with labelled levels for Example 1

important and possibly removed. Otherwise, we regard a parameter as a key parameter.

Definition 4: A parameter is a key parameter, if it is used as an input parameter of a service in the EDAG .

One condition of a redundant service is as following:

Proposition 1: **A service is redundant** if all its key outputs are reproduced by some other services.

Example 4: w_4 has one key output H, and H is reproduced by w_5. Therefore w_4 is a redundant service. w_5 has two key outputs H and I, both of which are reproduced by some other services. Therefore, w_5 is a redundant service.

Redundant services can be possibly removed without worsening the optimal QoS value. The removal can cause some other services to become useless and can be removed too. Please check the following example.

Example 5: After removing service w_5 from Figure 3, the new EDAG is shown in Figure 4. Obviously, w_4 is not a redundant service anymore. And w_2 can be further removed, because its only output is not a key output.

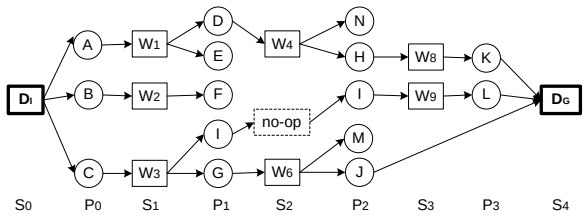

Figure 4. The EDAG after w_5 is removed from Figure 3

Therefore, we have the following proposition:

Proposition 2: A service is useless and removable if none of its outputs is a key parameter.

Removing services from a solution may worsen the optimal QoS value for a solution. Please check the following example.

Example 6: **Change the response time of w_6 in Example 1 to** 25. Table II shows response time before and after redundancy removal. Response time of the solution increases after service w_4 is removed. Hence, w_4 cannot be removed from the solution. After removing service w_2 and service w_5, response time of the solution remains optimal. Since removing services implies reducing cost, the solution after removing w_2 and w_5 becomes the optimal solution.

Table II
RESPONSE TIME BEFORE AND AFTER REDUNDANCY REMOVAL FOR EXAMPLE 6

Solution	Goal	Execution path	Response time	
Before removing service	K	$\{w_1; w_4; w_8\}$	120	145 (solution)
	L	$\{w_1 \|\| w_2; w_5; w_9\}$	130	
	J	$\{w_3; w_6\}$	145	
After removing w_4	K	$\{w_1 \|\| w_2; w_5; w_8\}$	160	160 (solution)
	L	$\{w_1 \|\| w_2; w_5; w_9\}$	130	
	J	$\{w_3; w_6\}$	145	
After removing w_2, w_5	K	$\{w_1; w_4; w_8\}$	120	145 (solution)
	L	$\{w_3; w_9\}$	140	
	J	$\{w_3; w_6\}$	145	

B. Model redundant service removal problem

Assume a solution sol with optimal response time contains n services $\{w_1, \ldots, w_n\}$. The total number of concepts $m = |\mathcal{C}|$, where $\mathcal{C}$ is the concept set. We model the redundant service removal problem as an integer programming problem $(X, D, C, f(sol))$, where X is the variable set, D is the domain set, C is the constraint set and $f(sol)$ is the objective function on sol.

1) Definition of variables and domains of the model: Variable set $X = \{X_0, X_1, \ldots, X_n, X_{n+1}\}$ corresponding to the set of services in sol , where

- X_i $(1 \leq i \leq n)$ is regarded as a real services w_i $(1 \leq i \leq n)$ in sol
- X_0 is a dummy service corresponding to $\mathcal{D}_{in}$
- X_{n+1} is a dummy service corresponding to $\mathcal{D}_{out}$
- Each variable X_i has a tuple $(X_{i.v}, L_i, I_i, O_i, X_{i.r}, X_{i.c})$, where:
  - $X_{i.v}$ is the variable to identify whether service w_i is included in the final solution. When $X_{i.v} = 1$, service w_i remains in sol; when $X_{i.v} = 0$, service w_i is removed from sol.
  - L_i is the variable corresponding to the level where service w_i is located
  - I_i is the $1 \times m$ array representing the input attributes of w_i. $I_{ij} = 1$, when the j^{th} attribute is an input of w_i, otherwise 0.
  - O_i is the $1 \times m$ array representing the output attributes of w_i. $O_{ij} = 1$, when the j^{th} attribute is an output of w_i, otherwise 0.
  - $X_{i.r}$ is the response time of service w_i.
  - $X_{i.c}$ is the execution cost of service w_i.

2) Objective function: The objective function f is the minimum total cost of the solution.

$$f = \min\{\sum_{i=1}^{i=n} X_{i.v} \cdot X_{i.c}\} \tag{6}$$

3) Constraints: C is the constraint set that contains all the constraints sol needs to satisfy after redundancy removal.

- Initial inputs constraint: X_0 should always be included in sol because this dummy service provides the initial

inputs of the composition query.

$$X_{0.v} = 1 \tag{7}$$

- Goal constraint: X_{n+1} should always be included in sol, because this dummy service's inputs are the goals and should always be satisfied.

$$X_{n+1.v} = 1 \tag{8}$$

- Service invokable constraint: Each service except the dummy service X_0 needs to be invokable in sol.

$$X_{k.v} \cdot I_{kj} \leq \sum_{L_i < L_k} X_{i.v} \cdot O_{ij} \tag{9}$$

where $k = 1, \ldots, n+1$, $j = 1, \ldots, m$, and $i = 0, \ldots, n$.

- Constraint on the key output parameter: Each real service in the final sol has to at least produce one key parameter.

$$X_{k.v} \cdot \sum_{j=1}^{m} \left[O_{kj} \cdot \sum_{L_i > L_k} X_{i.v} \cdot I_{ij} \right] \geq X_{k.v} \tag{10}$$

where $k = 1, \ldots, n$ and $i = 2, \ldots, n+1$.
This constraint needs a little explanation. When $X_{k.v} = 0$, service w_k is removed from sol. We have $X_{k.v} \cdot \sum_{j=1}^{m} \left[O_{kj} \cdot \sum_{L_i > L_k} X_{i.v} \cdot I_{ij} \right] = 0$ because none of the outputs of w_k is produced.
When $X_{k.v} = 1$, service w_k is not removed from sol. We check every output of w_k. For the j^{th} attribute of w_k,

$$O_{kj} \cdot \textstyle\sum_{L_i > L_k} X_{i.v} \cdot I_{ij} \tag{11}$$

$$\begin{cases} \geq 1 & \text{if } O_{kj} \text{ at least matches an input of} \\ & \text{services at an upper level } L_i \ (L_i > L_k) \\ = 0 & \text{otherwise} \end{cases}$$

Therefore, if w_k at the level L_k is able to at least produce one parameter which matches an input of services at an upper level L_i ($L_i > L_k$), we get the following equation

$$X_{k.v} \cdot \sum_{j=1}^{m} \left[O_{kj} \cdot \sum_{L_i > L_k} X_{i.v} \cdot I_{ij} \right] \geq 1$$

- Constraint on response time: The optimal response time of the solution needs to be guaranteed after redundancy removal.

$$\sum_{h=1}^{L_{max}} \max_{h} \{ X_{i.v} \cdot X_{i.r} | L_i = h \} = Q_r \tag{12}$$

where Q_r is the optimal response time of the solution, h denotes a level in the solution and $L_{max} = \max\{L_i | 1 \leq i \leq n\}$.

Based on the constraints and objective function defined, the redundant service removal model can be formulated:

$$f(sol) = \min\{\sum_{i=1}^{n} X_{i.v} \cdot X_{i.c}\}$$

subject to the constraints (7) through (12).

When throughput is the first QoS criterion, the variable $X_{i.r}$ is changed into $X_{i.p}$ which corresponds to the throughput of service w_i. Equation 12 is replaced with the constraint on the throughput:

$$\min_{h=1}^{L_{max}} \min_{h} \{ X_{i.v} \cdot X_{i.p} | L_i = h \} = Q_p \tag{13}$$

where Q_r is the optimal throughput of the solution.

C. Redundant service removal in QoS-aware service composition

According to the model in the previous subsection, we can use a standard integer programming solver to find a solution. Under the worst case, the time complexity of the standard solver would be the same as that of the exhaustive search. If the size of the service set in sol is n, the complexity is 2^n. In the context of redundancy removal, we will present an algorithm with less time complexity in this subsection.

The intuition behind our algorithm is that we probe the removal of all the combinations of redundant services. For each combination of redundant services, we remove redundant services in the combination and useless services afterwards. We will pick a solution which keeps the same response time (or throughput) and has minimal execution cost. Since the redundant services are normally just part of the total services, our algorithm has less time complexity.

Suppose we are removing redundant services from a solution sol with optimal response time, where sol contains n real services and $\mathcal{D}_{in}$ (resp. $\mathcal{D}_{out}$) corresponds to service w_0 (resp. service w_{n+1}) located at the lowest (resp. highest) level of sol. Algorithm 1 $RedundancyRemoval$ is the main algorithm. Initially, we have the optimal response time Q_{opt} (line 1) and the cost-minimized solution $optSol$ is supposed to be the original solution sol (line 2). Algorithm 2 $FindReduntS$ (line 3) searches for all redundant services in sol. If there are redundant services, Algorithm 3 $ReduntSolver$ (line 5) is called to remove redundant services and useless services in order to find a cost-minimized solution $optSol$ whose response time is still optimal.

Algorithm 2 $FindReduntS$ searches for all redundant services $reduntS$ satisfying Proposition 1. L_i denotes the level where service w_i is located at solution sol. If all key outputs of service w_i are contained in the union of outputs of other services at level L_j ($L_j \leq L_i$)(line 3), w_i is a redundant service (line 4).

Algorithm 3 $ReduntSolver$ removes redundant services and finds a cost-minimized solution with optimal response time. For each possible combination of redundant services

Algorithm 1 $RedundancyRemoval(sol)$

1: Q_{opt} is the optimal response time of sol;
2: $optSol \leftarrow sol$;
3: $reduntS \leftarrow FindReduntS(sol)$;
4: **if** $reduntS \neq \emptyset$ **then**
5: $\quad optSol \leftarrow ReduntSolver(sol, reduntS, Q_{opt})$;
6: **end if**
7: **return** $optSol$;

Algorithm 2 $FindReduntS(sol)$

1: $reduntS \leftarrow \emptyset$;
2: **for** $i = 1$ to n **do**
3: $\quad$ **if** $|w_{i.out} \cap (\cup_{L_j > L_i} w_{j.in}) - \cup_{w_j \neq w_i \wedge L_j \leq L_i} w_{j.out}| = 0$ **then**
4: $\quad\quad reduntS \leftarrow reduntS \cup \{w_i\}$
5: $\quad$ **end if**
6: **end for**
7: **return** $reduntS$;

(line 3), a new solution $newSol$ is generated by removing redundant services in the combination from sol (line 5). Algorithm $RemoveUselessServices$ (line 6) searches for useless services in current solution $newSol$ and removes these useless services from $newSol$. If each service in $newSol$ is invokable, it means $newSol$ can achieve the goals. Thus, $newSol$ is also a solution (line 7). Line 8 checks whether the response time of $newSol$ is optimal. If functional invokable solution $newSol$ with optimal response time (line 13) has lower execution cost (line 14), $newSol$ becomes the current cost-minimized solution $optSol$ (line 16).

Algorithm 4 $RemoveUselessServices$ removes useless services from current solution $newSol$. We search for useless services from level $maxL$ that is the highest level where real services are located in $newSol$ (line 1). If none of the outputs of service w_i is an input of services at an upper level L_j ($L_j > L_i$) or a goal (line 6), w_i is a useless service (line 7). Line 2 - line 11 removes useless services from $newSol$ level by level.

Algorithm 5 $CheckInvokability$ checks whether each service is able to be invoked and the solution $newSol$ can achieve the goals (line 2).

The complexity of the $RedundancyRemoval$ is as Theorem 1. The proof is omitted.

Theorem 1: The complexity of $RedundancyRemoval$ is 2^k, where k is the number of redundant services.

As the number of redundant services k is normally less than the number of the services in sol, our algorithm is normally faster than a standard integer programming solver.

Example 7: We apply Algorithm 1 to Example 1. The objective is to minimize the cost of the solution on the condition that optimal response time is guaranteed.

Initially, optimal response time Q_{opt} of the solution is 220

Algorithm 3 $ReduntSolver(sol, reduntS, Q_{opt})$

1: $optSol \leftarrow sol$;
2: $minCost$ is the total cost of sol;
3: **for** $cmbReduntS \in 2^{|reduntS|} - \{\emptyset\}$ **do**
4: $\quad newSol \leftarrow sol$;
5: $\quad newSol \leftarrow newSol - cmbReduntS$;
6: $\quad RemoveUselessServices(newSol)$;
7: $\quad isInvokable \leftarrow CheckInvokability(newSol)$;
8: $\quad$ **if** Response time of $newSol = Q_{opt}$ **then**
9: $\quad\quad isOpt \leftarrow true$;
10: $\quad$ **else**
11: $\quad\quad isOpt \leftarrow false$;
12: $\quad$ **end if**
13: $\quad$ **if** $isInvokable = true$ and $isOpt = true$ **then**
14: $\quad\quad$ **if** The total cost of $newSol < minCost$ **then**
15: $\quad\quad\quad minCost \leftarrow$ The total cost of newSol;
16: $\quad\quad\quad optSol \leftarrow newSol$;
17: $\quad\quad$ **end if**
18: $\quad$ **end if**
19: **end for**
20: **return** $optSol$;

Algorithm 4 $RemoveUselessServices(newSol)$

1: $maxL \leftarrow \max\{L_i | w_i \in newSol - \{w_0, w_{n+1}\}\}$;
2: **for** $l = maxL, \ldots, 1$ **do**
3: $\quad setS \leftarrow \{w_i | w_i$ locates at level l of $newSol\}$;
4: $\quad uselessS \leftarrow \emptyset$;
5: $\quad$ **for** $w_i \in setS$ **do**
6: $\quad\quad$ **if** $|w_{i.out} \cap (\cup_{L_j > L_i} w_{j.in})| = \emptyset$ **then**
7: $\quad\quad\quad uselessS \leftarrow uselessS \cup \{w_i\}$;
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad newSol \leftarrow newSol - uselessS$;
11: **end for**
12: **return**

Algorithm 5 $CheckInvokability(newSol)$

1: **for** $i = 1$ to $n + 1$ **do**
2: $\quad$ **if** $|w_{i.in} - \cup_{L_j < L_i} w_{j.out}| > 0$ **then**
3: $\quad\quad$ **return** $false$;
4: $\quad$ **end if**
5: **end for**
6: **return** $true$;

and $minCost = 240$.

Redundant service set $reduntS = \{w_4, w_5\}$, because all key parameters generated by w_4 and w_5 are reproduced. And $2^{|reduntS|} - \{\emptyset\} = \{\{w_4\}, \{w_5\}, \{w_4, w_5\}\}$ contains all the possible combinations of redundant services in $reduntS$.

Table III shows the probes to remove each of the combinations. For example, if $\{w_4\}$ is removed (see row 1),

no useless services are generated (row 2). The services in $newSol$ is shown in row 3. Each service in $newSol$ is invokable and $newSol$ can achieve the goals (row 4). The new solution has the same response time 220 (row 5). Execution cost is reduced to 220 (row 6). When $\{w_5\}$ is removed, w_2 becomes useless. After removal, we can see the new solution has the same response time 220. Execution cost is reduced to 180. We pick this solution $\{(w_1||w_3); (w_4||w_6); (w_8||w_9)\}$ as the new optimal solution.

Table III
REDUNDANCY REMOVAL FOR EXAMPLE 1

$cmbReduntS$	$\{w_4\}$	$\{w_5\}$	$\{w_4, w_5\}$
$uselessS$	$\emptyset$	$\{w_2\}$	$\{w_2\}$
Services in $newSol$	$\{w_1, w_2, w_3, w_5, w_6, w_8, w_9\}$	$\{w_1, w_3, w_4, w_6, w_8, w_9\}$	$\{w_1, w_3, w_6, w_8, w_9\}$
$checkInvokable$	true	true	false
$optQ(newSol)$	220	220	-
$cost(newSol)$	220	180	-

IV. EXPERIMENTAL RESULT

A. Data set

The experiments are carried out on the WSC09 [2] data sets. Each data set contains a WSDL file for web services, an OWL file for ontology concepts and a WSLA file for QoS values (response time and throughput) of web services. Each web service in WSC09 data sets is taken as a black box and the functionality of each service is represented by multiple inputs/output parameters. The Web service input/output parameters are instances ("things" in an OWL file) of the semantic concepts in OWL files. Web services are described in a WSDL file annotated with a simple extension mechanism to link to the ontology definition in the OWL file. The WSLA file defines a response time and a throughput for a web service.

Five data sets from WSC09 are used in our experiment. Dataset 1 has 500 services and 5,000 concepts. Dataset 5 has 15,000 services and 100,000 concepts. The other datasets have 4,000-8,000 services and 40,000-60,000 concepts.

B. Experimental results

A service in the data sets is provided with response time and throughput. Its execution cost is not available. To deal with this situation, we use the value of throughput as the value of execution cost for a service if removing redundant services from a solution optimized with response time. Conversely, when removing redundant services from a solution with optimal throughput, we take the value of response time as the value of execution cost for a service.

After analyzing the composition results posted at [2], especially the composition results of the first place winner [8] and the second place winner [16], We find redundant services exist in almost all the results for all the data sets. Though all the presented algorithms can find a solution with the correct optimal response time (or throughput), service redundancy is a quite common problem in their solutions. We use the results of the second place winner [16] as the original solutions to test our method. This is because these results contain more redundant services than any other solutions posted at [2].

We compare our method with the method presented in our previous work [18]. [18] also proposed a redundancy removal method. This method randomly selects a redundant service to remove and further removes any useless services, as long as the removal does not worsen the QoS value or renders the solution invalid. This method cannot guarantee to find a cost-optimized solution after redundancy removal. Table IV shows the results over the solutions with optimal response time. The check marks on the row "Keep Resp. Time" mean that the redundancy removal maintains the optimal value. The row "Redunt. # Services" shows the number of redundant services when starting our algorithm. "Redunt. # Services" determines the time complexity. "# Services" shows the number of services in the solutions. Comparing the "# Services" before and after the removal, we can see the removed services count from 9% to 38% of the total services in the original solutions. "Exec. Cost" shows the execution cost of the solution. Noticeably, our method can give better execution cost than the method in [18]. Similarly, Table V shows the results over the solutions with optimal throughput.

V. RELATED WORK

The QoS-aware service composition problem is studied under different assumptions. The first type of problems is called service selection problem, in which a business process template is pre-defined. The objective is to select the services for the tasks in the business process so that the resulting business process can have optimal QoS values. Since the activities in the business process are pre-defined, there is no service redundancy problem in this type of problems. When facing optimization of multi-criteria QoS values, this combinatorial optimization problem can be modelled as a multi-dimension multi-choice 0-1 knapsack problem. Integer programming is a powerful tool to solve it [19]. As this is an NP-complete problem, heuristic search can be applied to search the problem space only partially [1], [17]. Genetic Algorithm (GA) is another way to partially search the problem space [4]. And the advantage of GA compared to integer programming is that GA can deal with nonlinear constraints of QoS requirements.

The second type of problems is to determine the business process and optimize the QoS criteria at the same time. This kind of QoS-aware service composition problem is widely studied in WSC09 [2]. Using WSC09's data, several systematic search algorithms are developed for single QoS criterion, e.g. [8], [16], and [18]. Since [8], [16], and [18] normally find an optimal path for each of the outputs and

		Dataset1	Dataset2	Dataset3	Dataset4	Dataset5
Original solution produced by [16]	Resp. Time	500	1690	760	1470	4070
	Exec. Cost	126000	322000	80000	562000	432000
	#Services	13	27	11	52	41
Our method	Keep Resp. Time	✓	✓	✓	✓	✓
	Exec. Cost	73000	248000	75000	453000	342000
	#Services	8	21	10	42	33
	Redunt. #Services	4	8	1	13	4
Random removal method in [18]	Keep Resp. Time	✓	✓	✓	✓	✓
	Exec. Cost	112000	301000	75000	556000	414000
	#Services	5	20	10	40	32

Table IV
REDUNDANCY REMOVAL RESULTS FOR THE SOLUTIONS WITH OPTIMAL RESPONSE TIME: OUR METHOD /RANDOM REMOVAL METHOD IN [18]

		Dataset1	Dataset2	Dataset3	Dataset4	Dataset5
Original solution produced by [16]	Throughput	15000	6000	4000	2000	4000
	Exec. Cost	1810	6300	7490	13600	9720
	#Services	7	24	31	53	41
Our method	Keep Throughput	✓	✓	✓	✓	✓
	Exec. Cost	1200	5190	4840	10960	7500
	#Services	5	20	21	40	30
	Redunt. #Services	2	4	8	12	5
Random removal method in [18]	Keep Throughput	✓	✓	✓	✓	✓
	Exec. Cost	1730	5350	7290	12880	9330
	#Services	5	20	15	42	32

Table V
REDUNDANCY REMOVAL RESULTS FOR THE SOLUTIONS WITH OPTIMAL THROUGHPUT: OUR METHOD /RANDOM REMOVAL METHOD IN [18] METHOD

then combine the paths to be the optimal solution. The optimal solutions obtained by these algorithms always contain redundant services. But these algorithms have polynomial time complexity. When optimizing multiple QoS values, this kind of problem can also be modelled as an optimization problem. Integer Programming (IP) is used to find a QoS optimized solution [5]. [5] models multiple objectives as multiple QoS criteria to be optimized and constrains as functional and non-functional requirements. The multi-criteria programming model proposed by [5] falls into two categories: One is a preemptive method, and the other a non-preemptive method (weighted average method). The preemptive method gives the priority of the objectives in order such that optimization of multiple QoS criteria is achieved one by one. However, modelling QoS-aware service composition into an IP problem is not so straightforward and the model loses the business logic. Theoretically, if execution cost is not used to optimize the solution, the solution may include redundant services.

VI. CONCLUSION

In this paper, we study the redundant service removal problem in QoS-aware service composition. When execution cost is not used to optimize the solutions, it is possible that the solutions may include redundant services. Even if execution cost is not the criterion to optimize, it is preferable if we can remove some removable services from the solution. Therefore, we recommend to use our method as a last step to further optimize the solution obtained by other methods. We try to solve this problem in the context of redundancy removal, though we can model this problem as yet-another IP problem and use a standard IP solver to solve it. An algorithm with lower time complexity than IP is presented.

REFERENCES

[1] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. Heuristics for qos-aware web service composition. In *Proc. ICWS*, pages 72–82, 2006.

[2] Steffen Bleul. Web service challenge rules. http://ws-challenge.georgetown.edu/wsc09/downloads/WSC2009Rules-1.1.pdf, 2009.

[3] A. L. Blum and M. L Furst. Fast Planning through Planning Graph Analysis. *Artificial Intelligence Journal*, 90(1-2):225–279, 1997.

[4] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proc. GECCO*, pages 1069–1075, 2005.

[5] LiYing Cui, Soundar Kumara, and Dongwon Lee. Scenario analysis of web service composition based on multi-criteria mathematical programming. *INFORMS Service Science*, 3(3), 2011.

[6] Joyce El Haddad, Maude Manouvrier, and Marta Rukoz. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *IEEE T. Services Computing*, 3(1):73–85, 2010.

[7] J. Hoffman and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[8] Zhenqiu Huang, Wei Jiang, Songlin Hu, and Zhiyong Liu. Effective pruning algorithm for qos-aware service composition. In *Proc. CEC*, pages 519–522, 2009.

[9] OASIS. Uddi version 2.04 api specification. http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm, 2007.

[10] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[11] J. Rao and X. Su. A survey of automated web service composition methods. In *Proc. of 1st Int. WS on Semantic Web Services and Web Process Composition, SWSWPC*, 2004.

[12] W3C. Owl web ontology language overview. http://www.w3.org/TR/owl-features/. Retrieved 2011-06-30, 2004.

[13] W3C. Semantic annotations for wsdl and xml schema (sawsdl). http://www.w3.org/TR/sawsdl/. Retrieved 2011-06-30, 2007.

[14] W3C. Soap version 1.2 part 1: Messaging framework (second edition). http://www.w3.org/TR/soap12-part1/#intro. Retrieved 2011-06-30, 2007.

[15] W3C. Web services description language (wsdl) version 2.0. http://www.w3.org/TR/wsdl20/. Retrieved 2011-06-30, 2007.

[16] Yixin Yan, Bin Xu, Zhifeng Gu, and Sen Luo. A qos-driven approach for semantic service composition. In *Proc. CEC*, pages 523–526, 2009.

[17] Tao Yu, Yue Zhang 0001, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1), 2007.

[18] Min Chen Yuhong Yan and Yubin Yang. Anytime qos optimization over the plangraph for web service composition. In *ACM SAC 2012*, 2012.

[19] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.