Supervised Web Service Composition Integrating Multi-objective QoS Optimization and Service Quantity Minimization

Shi-Liang Fan[1,2], Feng Ding[2], Cheng-Hao Guo[2], and Yu-Bin Yang[1(✉)]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
dyyslfan@smail.nju.edu.cn, yangyubin@nju.edu.cn
[2] Science and Technology on Information System Engineering Laboratory, Nanjing 210007, China

Abstract. The QoS of web service has been increasingly crucial due to the escalating number of services with similar or identical functionality, which leads to intensive researches on QoS-aware web service composition. Correspondingly, to optimize not only QoS but also service quantity in a composition has also been increasingly challenging. Currently, there are already many researches on service composition addressing the optimization of multiple QoS attributes, but it is still rare to take service quantity as an optimization objective as well. To address this issue, this paper proposes a novel supervised web service composition mechanism integrating multi-objective QoS optimization and the minimization of service quantity. Firstly a memory-based search algorithm is proposed to compute each single-objective optimal QoS, after which a knapsack-variant algorithm is applied to minimize the number of services without considering the QoS. Finally, a supervised multi-objective optimization is performed based on the above single-objective optimization results. Experimental results on both Web Service Challenge 2009's datasets and substantial datasets randomly generated show that the proposed service composition method outperforms the state-of-the-arts by achieving a much better tradeoff among all the objectives.

Keywords: Web service composition · Multi-objective · Supervised

1 Introduction

QoS-aware web service composition problem has been widely studied in recent years [1–4]. A survey shows that the majority of studies aim at optimizing a single global QoS [5]. If there are more than two objectives, i.e., QoS attributes, involved, it usually fails to achieve the satisfactory performance [6]. Moreover, since the QoS attributes are usually in conflict to each other, it is often impossible

H. Jin et al. (Eds.): ICWS 2018, LNCS 10966, pp. 215–230, 2018.
https://doi.org/10.1007/978-3-319-94289-6_14

to find a solution that maximizes/minimizes all of them. Consequently, multi-objective service composition starts to attract more attention in the research community, the goal of which is to find the composition(s) capable of achieving better tradeoffs among all QoS objectives.

For example, Zeng et al. simply transformed the multi-objective service composition into single-objective optimization by defining a special objective function [7], after which the traditional techniques were applied to solve it. At the same time, there are also other approaches proposed based on Pareto set model [8–11], which aimed at searching for a set of Pareto optimal compositions rather than a single solution, which exhibited different tradeoffs among all QoS objectives.

Besides QoS objectives, minimizing service quantity in the resulting composition has important benefits as well for brokers, customers and service providers [12]. Therefore, it is necessary to set the number of services as one of the optimization objectives when conducting service composition. However, the current available approaches to multi-objective service composition are all based on the same prerequisite that there is only one composition workflow with a fixed set of abstract tasks, where each abstract task can be implemented by a concrete service. Both the composition workflow and the candidate services for each abstract task are predefined beforehand, which makes them impossible to generate compositions with variable sizes. Up to now, only a few studies on the problem of multi-objective service composition started to take the service quantity into consideration [13–16], by which the generated solutions were still far from satisfactory.

Supposing that the optimal result of each single-objective has been solved, there is hardly a perfect composition that integrates all the results of single-objective optimization. Considering that, this paper aims to find a web service composition in which all attributes (including service quantity) approximate to the perfect composition as close as possible under the supervision of all single-objective optimization results. In this regard, the finally generated composition is able to perform satisfactorily in each attribute, which achieves an ideal tradeoff among all objectives. Accordingly, a multi-objective service composition mechanism is proposed in this paper to effectively and efficiently find such a composition. The main contributions are as follows:

- A memory-based search algorithm is proposed, which efficiently generates each single-objective optimal QoS.
- An efficient approach integrating greedy strategy and knapsack-variant algorithm is proposed, which efficiently minimize service quantity in resulted compositions without consideration of QoS attributes.
- A supervised multi-objective optimization algorithm is proposed, which transforms multi-objective service composition into single-objective optimization based on the results of the above two algorithms.

Furthermore, to validate the proposed methods, extensive experiments have also been carried out on both WSC-2009's datasets and randomly generated datasets.

The rest of this paper is organized as follows. Section 2 describes the background and reviews some related work. Section 3 illustrates the motivation of this research. Section 4 presents the proposed mechanism in detail. Section 5 shows and analyzes the experimental results, and Sect. 6 provides final remarks.

2 Background and Related Work

2.1 Background

Web services are the foundation of this paper. The formal definition of a *web service* is given as follows.

Definition 1. *A Web Service ("service" for short) is defined as a tuple* $s = \{In_s, Out_s, Q_s\}$*, where* $In_s = \{in_s^1, \ldots, in_s^n\}$ *is the set of inputs required to invoke the service s, and* $Out_s = \{out_s^1, \ldots, out_s^n\}$ *is the set of outputs generated by executing s. Each input and output is related to a semantic concept from the set Con defined in an ontology, namely,* $In_s \subseteq Con$ *and* $Out_s \subseteq Con$*.* $Q_s = \{q_s^1, \ldots, q_s^n\}$ *is the set of nonfunctional attributes which are the measures for how well the service s serves the user.*

Obviously, services aren't independent to each other. Relevant services can be combined by connecting matched inputs and outputs to construct compositions.

Lemma 1. *Given an output* out_s *of a service s, and an input* $in_{s'}$ *of another service* s'*, if* out_s *and* $in_{s'}$ *are equivalent concepts or* out_s *is a sub-concept of* $in_{s'}$*,* out_s *matches* $in_{s'}$ *(i.e.,* $in_{s'}$ *is matched by* out_s*).*

Each individual service has its own QoS, which contributes to the global QoS of a composition. The computation of the global QoS depends on the structure of the composition. There are mainly two kinds of structures, namely *sequential structure* and *parallel structure*. The services organized as a sequential structure are invoked in order, while those in parallel structure are invoked synchronously.

Definition 2. *A Composition containing the set of services* $S = \{s_1, \ldots, s_n\}$ *is represented as* Ω*. If the services are chained in sequence, the composition is expressed as* $\Omega^{\rightarrow} = s_1 \rightarrow \ldots \rightarrow s_n$*; if in parallel,* $\Omega^{\parallel} = s_1 \parallel \ldots \parallel s_n$*. The set of services involved in* Ω *is defined as* $Servs(\Omega) = S$*. Moreover, the length of a composition* Ω *is defined as* $Len(\Omega) = |S|$*, namely the number of services in* Ω*. Taking the response time as an example, the global QoS of* Ω *is computed as:*

$$\left.\begin{aligned} RT(\Omega^{\rightarrow}) &= \sum_{i=1}^{n} RT(s_i),\ s_i \in S \\ RT(\Omega^{\parallel}) &= \max_{1 \le i \le n} RT(s_i),\ s_i \in S \end{aligned}\right\}. \tag{1}$$

where $RT(\Omega)$ represents the global response time of the composition Ω, and $RT(s)$ represents the response time of the service s. Similarly, the global throughput $TP(\Omega)$ of the composition lies on the throughput $TP(s)$ of each service $s \in S$.

$$\left.\begin{aligned} TP(\Omega^{\rightarrow}) &= \min_{1 \le i \le n} TP(s_i),\ s_i \in S \\ TP(\Omega^{\parallel}) &= \min_{1 \le i \le n} TP(s_i),\ s_i \in S \end{aligned}\right\}. \tag{2}$$

Based on the above concepts, the precise definition of the *multi-objective web service composition* in this paper is provided as follows.

Definition 3. *Multi-Objective Web Service Composition is defined as, for a given composition request $R = \{In_R, Out_R\}$, to seek for a composition Ω that achieves an ideal tradeoff among $Len(\Omega)$, $RT(\Omega)$, $TP(\Omega)$ and etc.*

2.2 Related Work

Multi-objective service composition is a fundamental research topic in the field of service computing, which has been approached from many perspectives.

To deal with multiple QoS attributes, a mechanism based on Simple Additive Weighting (SAW) was proposed in [7]. Firstly, the multiple objectives were aggregated to a single one via a linear weight sum. Then, an objective function was defined as the optimization formula that can be solved by traditional techniques. The optimal composition was the one with the best function value. The method is easy to apply, however, it is based on the assumption that there is a predefined composition workflow with a fixed set of abstract tasks which can be implemented by concrete services. Therefore, the mechanism fails to take the number of services into consideration.

Pareto set model is widely used in multi-objective optimization and multi-criteria decision making. Many composition methods based on Pareto set model have been investigated recently. Yu and Bouguettaya [10] presented a Bottom-Up algorithm to compute the Pareto optimal compositions representing the tradeoffs related to different QoS attributes. Moustafa and Zhang used multi-objective reinforcement learning to enable web service composition considering multiple QoS objectives [11]. Similar to the approach in [7], these methods depend on predefined composition workflow, which makes the number of services in the final compositions invariable.

Xia and Yang addressed the issue of service quantity in web service composition by proposing a novel composition algorithm integrating both QoS optimization and redundancy removal [15]. Only two objectives were involved in the algorithm, which failed to guarantee the satisfactory performance on other QoS attributes.

A redundant service removal mechanism was presented by Chen and Yan [14]. This method firstly modeled the composition problem as an integer programming problem (IP), and then obtained a composition with an optimal global QoS by solving it. The next step was to remove redundant services in the composition while keeping the optimal QoS. During the process of redundancy removal,

another QoS attribute was to some extent optimized simultaneously. Though three objectives, including the number of services, are considered when conducting compositions, the mechanism took very long time to generate solutions.

Chattopadhyay et al. presented an approximate mechanism to obtain the solutions against time [16]. The authors proposed an on-the-fly strategy to construct only a path of the auxiliary graph instead of the complete graph. Then, the path selection algorithm considering multiple objectives was discussed, after which a greedy strategy was adopted to transform multi-objective service composition into single-objective one. The method has a superior execution time compared to the others, whereas there is still much room for improvement in the solutions generated by it.

3 Motivation

Graph is a natural and intuitive way to express the complex interaction relations between entities. As shown in Fig. 1, a service composition problem whose request is $R = \{\{in_1, in_2\}, \{out_1, out_2\}\}$ is described as a layered directed graph. Each rectangle in the graph represents a web service (associated to a response time and a throughput), while each circle is an input or output of a service. In addition, the edges connecting circles represents the matching relations among services.

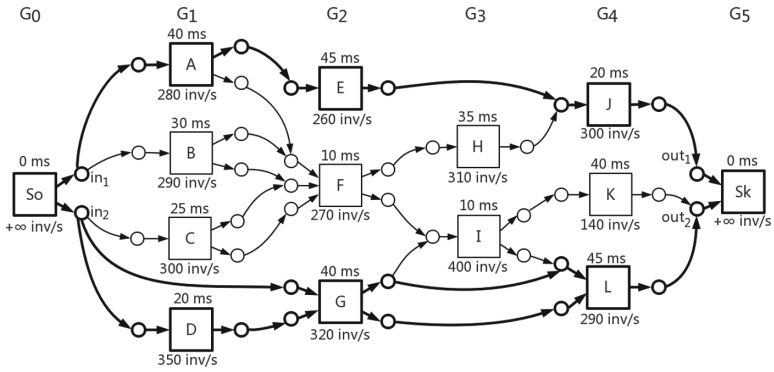


Fig. 1. An example of service dependency graph. The ideal composition is highlighted.

As can be seen from the graph, there are quantities of feasible compositions satisfying R with different QoS and different number of services. The composition $\Omega = s_o \rightarrow (B \parallel C) \rightarrow F \rightarrow ((H \rightarrow J) \parallel (I \rightarrow K)) \rightarrow s_k$ is the one with the optimal global response time of 95 ms. Meanwhile, the throughput of Ω is $TP(\Omega) = 140\ inv/s$ and the length (including the s_o and the s_k) is $Len(\Omega) = 9$. Moreover, the highlighted composition $\Omega' = s_o \rightarrow ((A \rightarrow E \rightarrow J) \parallel (D \rightarrow G \rightarrow L)) \rightarrow s_k$ has the response time of 105 ms, the throughput of 260 inv/s, and

the length of 8. Both Ω and Ω' are Pareto optimal solutions, but we prefer the latter because $RT(\Omega')$ changes little from $RT(\Omega)$, while the throughput of Ω' has been greatly improved (260 versus 140). Moreover, $Len(\Omega')$ is also smaller than $Len(\Omega)$.

Given a request of composition like R, the paper aims at generating a digraph similar to the one shown in Fig. 1 and finding the composition with an ideal tradeoff among all objectives, just like the highlighted Ω'.

4 Composition Mechanism

In this section, an efficient mechanism is proposed for the multi-objective service composition. Given a request $R = \{In_R, Out_R\}$ and a service repository S_{all}, a service dependency graph is firstly constructed with the relevant services for the request. Then, a memory-based search algorithm is proposed to compute each single-objective optimal QoS, and a knapsack-variant algorithm is applied to minimize the number of services without considering the QoS. Finally, a supervised algorithm is proposed to transform the problem into single-objective one on the basis of the pre-computed single-objective optimization results.

4.1 Generation of the Service Dependency Graph

For the given user request $R = \{In_R, Out_R\}$, a service dependency graph similar to the one shown in Fig. 1 is constructed to show the input-output dependency among services. There is only a dummy service $s_o = \{\varnothing, In_R, \{0\ ms, +\infty\ inv/s\}\}$ in the first layer, and another dummy service $s_k = \{Out_R, \varnothing, \{0\ ms, +\infty\ inv/s\}\}$ is also the only one contained in the last layer. The specific services in the other layers are selected from an external repository S_{all} and each layer contains the services whose inputs are all matched by the outputs generated by previous layers. After constructing the service dependency graph G, we have:

Definition 4. *The set of precursors of a service $s \in G_i$ is defined as $Pre(s) = \{s' \mid s' \in G_j(\forall j < i) \wedge In_s \cap Out_{s'} \neq \varnothing\}$. In particular, $Pre(s_o) = \varnothing$.*

Hereafter, the precursors of a service s is expressed as $Pre(s)$ which will be used frequently in the following algorithms.

4.2 Computation of the Optimal QoS

Based on the service dependency graph G, a memory-based search algorithm is proposed to efficiently compute each single-objective optimal QoS. The algorithm is applicable to diverse QoS, and here the response time and the throughput are selected as the representatives of all QoS. For each service $s \in G$, there are many possible compositions that starts from the service s_o and ends with the service s, among which the one with the optimal response time is expressed as Ω_s^R and the one with the optimal throughput is represented as Ω_s^T. Therefore, for each

Algorithm 1. Memory-Based Search Algorithm for Response Time

```
Input: G
Output: OptR
OptR ← {s_o : 0}
for i = 1; i < |G|; i++ do
    for service s ∈ G_i do
        tmpR ← 0
        for concept c ∈ In_s do
            Rin[c] ← +∞
        for concept c ∈ In_s do
            for service s' ∈ Pre(s) do
                if c ∈ Out_s' and OptR[s'] < Rin[c] then
                    Rin[c] ← OptR[s']
            tmpR ← max(tmpR, Rin[c])
        OptR[s] ← tmpR + RT(s)
return OptR
```

$s \in G$, the memory-based search algorithm is used to compute the $RT(\Omega_s^R)$ and $TP(\Omega_s^T)$.

For each input $in_s \in In_s$, let $Rin[in_s]$ represent the shortest response time to obtain in_s. Then, the decision-making process of $RT(\Omega_s^R)$ is

$$\begin{aligned} RT(\Omega_s^R) &= \max_{in_s \in In_s} \{Rin[in_s]\} + RT(s). \\ \text{where} \quad & Rin[in_s] = \min_{s' \in Pre(s) \ \wedge \ in_s \in Out_{s'}} RT(\Omega_{s'}^R). \end{aligned} \tag{3}$$

Similarly, let $Tin[in_s]$ represent the highest throughput to obtain in_s. Then,

$$\begin{aligned} TP(\Omega_s^T) &= \min \{ \min_{in_s \in In_s} \{Tin[in_s]\}, TP(s)\}. \\ \text{where} \quad & Tin[in_s] = \max_{s' \in Pre(s) \ \wedge \ in_s \in Out_{s'}} TP(\Omega_{s'}^T). \end{aligned} \tag{4}$$

Taking the computation of each $RT(\Omega_s^R)$ as an instance, the memory-based search algorithm is shown in Algorithm 1. As can be seen from (3), the computation of the service s depends on the computation results of the precursors of s, hence the computation processes over the graph G are performed layer by layer. All the results are cached in the returned $OptR$ where $OptR[s] = RT(\Omega_s^R)$.

In the same way, the computation results of the optimal throughput can be obtained as $OptT$ where $OptT[s] = TP(\Omega_s^T)$ on the ground of the model in (4).

4.3 Computation of the Minimal Number of Services

For each service s in the dependency graph G, there are many possible compositions starting from service s_o and ending with service s, among which the one

with the minimal number of services is expressed as Ω_s^L in this paper. Seeing that the memory-based search algorithm isn't applicable to compute $Len(\Omega_s^L)$ for each s, an efficient mechanism integrating greedy strategy and knapsack-variant algorithm is proposed to solve the problem [17].

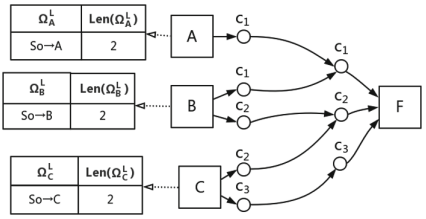


Fig. 2. Search step on the graph.

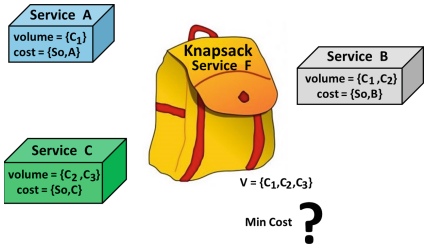


Fig. 3. Dynamic knapsack problem.

Firstly, the search step of Ω_s^L is defined as determining the optimal precursors of the service s according to a greedy strategy. As shown in Fig. 2, supposing that $\{\Omega_A^L, \Omega_B^L, \Omega_C^L\}$ have been determined in advance, the search step of Ω_F^L is defined as selecting the optimal subset of $\{\Omega_A^L, \Omega_B^L, \Omega_C^L\}$ to compose Ω_F^L, which is in fact a greedy strategy. Therefore, search steps on the graph should be executed layer by layer because each search step depends on the optimization results of the search steps in previous layers.

Then, for each service s, the search step of Ω_s^L can be transformed into a dynamic knapsack problem. As can be seen in Fig. 3, the service s is abstracted into a knapsack with a *capacity*, and each precursors of s is spontaneously regarded as an item with a *volume* and a *cost*. The objective is to minimize the sum of the cost of the items in the knapsack so that the sum of the volume is equal to the knapsack's capacity. The capacity of the knapsack s is In_s (the set of inputs of service s), while the volume of an item $s' \in Pre(s)$ is relevant to $Out_{s'}$ (the set of outputs of the service s') and the cost is measured with $Servs(\Omega_{s'}^L)$ (the set of services involved in the composition $\Omega_{s'}^L$). Set operations are too inconvenient to be applied to the following composition algorithm. Therefore, an approach is presented to quantify the capacity of the knapsack, as well as the volume and cost of each item.

All the subsets of In_s is obtained by Algorithm 2 in a certain order. According to the returned $Subs$ whose indices start at 0, the quantization approach can be described as follows.

- The capacity of knapsack s is quantified as $V_{cap} = |Subs| - 1$.
- Assuming that service s' provides the set of outputs $Out \subseteq Out_{s'}$ for service s, the volume of s' is quantified as the value of the $index$ that satisfies the condition that $Subs[index] = Out$.
- Assuming that Ser represents the set of services that belong to $Servs(\Omega_{s'}^L)$ and have not yet been selected, the volume of the item s' is quantified as the size of the $Ser \cup \{s'\}$.

Algorithm 2. Generation of Subsets

Input: In_s
Output: $Subs$
$Subs \leftarrow \{\varnothing\}$, $upper\_bound \leftarrow 2^{|In_s|}$
for $index = 0; index < upper\_bound; index++$ **do**
  $i \leftarrow 0$, $tmp \leftarrow index$, $subset \leftarrow \{\}$
  while $tmp > 0$ **do**
    if $(tmp \bmod 2) > 0$ **then**
      $subset \leftarrow subset \cup \{In_s[i]\}$
    $tmp \leftarrow tmp \text{ div } 2$, $i \leftarrow i + 1$
  $Subs[index] \leftarrow subset$
return $Subs$

Owning to the fact that Out and Ser are uncertain before decision-making, both the volume and the cost of s' cannot be determined in advance, which leads to the inapplicability of the 0-1 knapsack algorithm. A knapsack-variant algorithm is proposed to solve the problem by determining the volume and cost of each item dynamically.

Given a knapsack s, the capacity of which is V_{cap}, and a set of items $Pre(s) = \{s_1, s_2, \ldots, s_N\}$ where $N = |Pre(s)|$ represents the number of items, each with an uncertain volume $volume_i$ and an uncertain cost $cost_i$, let $C[i][v]$ represent the minimal cost of selecting items from $\{s_1, s_2, \ldots, s_i\}$ $(1 \leq i \leq N)$ to fill a temporary knapsack, the capacity of which is v $(1 \leq v \leq V_{cap})$, and $I[i][v]$ represents the set of items selected to minimize $C[i][v]$. Then,

$$\begin{aligned} C[i][v] &= \mathbf{min}\ \{C[i-1][v], C[i-1][v - volume_i] + cost_i\} \\ \text{where} \quad & volume_i = \mathbf{DV}(s_i, In_s, Subs, v), \\ & cost_i = \mathbf{DC}(s_i, I, i, v, volume_i). \end{aligned} \tag{5}$$

The function DV in Algorithm 3 is used to dynamically calculate the volume of an item. For the given temporary knapsack with capacity v, the outputs provided by service s_i for the knapsack are determined as $Out_{s_i} \cap Subs[v]$. Thus, the volume of item s_i can be quantified by the approach proposed above.

Moreover, the function DC shown in Algorithm 4 is applied to determine the cost of an item s_i drawing support from I.

According to the optimization model in (5), by systematically increasing the values of i (from 1 to N) and v (from 1 to V_{cap}), composition Ω_s^L with the minimal number of services will be finally obtained when $i = N$ and $v = V_{cap}$:

$$OptL[s] = Len(\Omega_s^L) = C[N][V_{cap}] + 1. \tag{6}$$

4.4 Supervised Multi-objective Service Composition

We have obtained the composition $\Omega_{s_k}^R$ with the shortest response time, the composition $\Omega_{s_k}^T$ with the highest throughput, and the composition $\Omega_{s_k}^L$ with the

Algorithm 3. Determination of Volume of Items

Input: s_i, In_s, $Subs$, v
Output: $DV(s_i, In_s, Subs, v)$
$map \leftarrow \{\}$, $volume \leftarrow 0$, $Out \leftarrow Out_{s_i} \cap Subs[v]$
for $index = 0; index < |In_s|; index++$ **do**
  $c \leftarrow In_s[index]$, $map[c] \leftarrow index$
for concept $c \in Out$ **do**
  $index \leftarrow map[c]$, $volume \leftarrow volume + 2^{index}$
return $volume$

Algorithm 4. Determination of Cost of Items

Input: s_i, I, i, v, $volume_i$
Output: $DC(s_i, I, i, v, volume_i)$
$Union \leftarrow \{\}$
for service $s \in I[i-1][v - volume_i]$ **do**
  $Union \leftarrow Union \cup Servs(\Omega^s)$
$Inter \leftarrow Servs(\Omega^{s_i}) \cap Union$, $Ser \leftarrow Servs(\Omega^{s_i}) - Inter$
return $|Ser| + 1$

minimal number of services. However, in most cases, these compositions are not the same one. For instance, $\Omega^R_{s_k}$ may hold a low throughput or a long length. To reach a compromise among all these attributes, a supervised algorithm is applied to aggregate multiple objectives to a single one drawing support from the pre-computed optimization results, i.e., $OptR$, $OptT$, and $OptL$.

Assuming that Ω_s represents a composition which starts from s_o and ends with the service s, to measure the overall quality of Ω_s, the *loss* is defined as

$$Loss(\Omega_s) = \frac{RT(\Omega_s) - RT(\Omega^R_s)}{RT(\Omega^R_s)} + \frac{TP(\Omega^T_s) - TP(\Omega_s)}{TP(\Omega^T_s)} + \frac{Len(\Omega_s) - Len(\Omega^L_s)}{Len(\Omega^L_s)}. \tag{7}$$

Note that, the division operations in (7) are used to eliminate the effects brought by the different dimensions of different attributes, which plays the role of normalization. For a composition Ω_s, $Loss(\Omega_s)$ can be explained as the degree of deviation from the perfect composition of current search step. A composition that performs poorly in certain attribute brings a great loss, while a composition whose attributes are all approximate to the pre-computed optimization results brings a minor loss. Therefore, to achieve an ideal compromise among all the objectives, we aim at searching a composition Ω_s with the minimal $Loss(\Omega_s)$. There are many possible compositions starting from the service s_o and ending with the service s, among which the one with the minimal loss is expressed as Ω^M_s. Then, for each service $s \in G$, the search step of Ω^M_s is to find a composition Ω_s with the objective of **min** $Loss(\Omega_s)$.

Assuming that the precursors of the service s is $Pre(s) = \{s_1, s_2, \ldots, s_N\}$ where $N = |Pre(s)|$ and the set of compositions $\{\Omega_{s'}^M \mid s' \in Pre(s)\}$ has been determined in advance, inspired by the methods in Sect. 4.3, the decision-making process of Ω_s^M is regarded as an approximate knapsack problem. Given In_s, the $Subs$ can be obtained according to Algorithm 2. Then, the service s is abstracted into a knapsack whose capacity is $V_{cap} = |Subs| - 1$, and each s_i $(1 \leq i \leq N)$ is regarded as an item with a volume of $volume_i$. A composition Ω_s is generated when the knapsack s is filled with items. Therefore, the loss of the filled knapsack s is defined as the value of $Loss(\Omega_s)$. The problem is to select items from $Pre(s)$ to fill the knapsack s with the objective of minimizing the loss of the knapsack. Since the optimal substructure can't be guaranteed, an approximate algorithm is proposed to solve the problem against time.

Algorithm 5. Determination of Temporary Loss

Input: s, s_i, $I[i-1][v - volume_i]$, $OptR$, $OptT$, $OptL$
Output: $DL(s, s_i, I[i-1][v - volume_i], OptR, OptT, OptL)$

$RT_{tmp} \leftarrow RT(\Omega_{s_i}^M)$, $TP_{tmp} \leftarrow TP(\Omega_{s_i}^M)$, $Union \leftarrow Servs(\Omega_{s_i}^M)$
for service $s' \in I[i-1][v - volume_i]$ **do**
  $RT_{tmp} \leftarrow max(RT_{tmp}, RT(\Omega_{s'}^M))$, $TP_{tmp} \leftarrow min(TP_{tmp}, TP(\Omega_{s'}^M))$
  $Union \leftarrow Union \cup Servs(\Omega_{s'}^M)$
$RT_{tmp} \leftarrow RT_{tmp} + RT(s)$, $TP_{tmp} \leftarrow min(TP_{tmp}, TP(s))$, $Len_{tmp} \leftarrow |Union| + 1$
$loss \leftarrow \frac{RT_{tmp} - OptR[s]}{OptR[s]} + \frac{OptT[s] - TP_{tmp}}{OptT[s]} + \frac{Len_{tmp} - OptL[s]}{OptL[s]}$
return $loss$

Let $L[i][v]$ represent the minimal loss of selecting items from $\{s_1, s_2, \ldots, s_i\}$ $(1 \leq i \leq N)$ to fill a temporary knapsack whose capacity is v $(1 \leq v \leq V_{cap})$, and $I[i][v]$ represents the set of items selected to minimize $L[i][v]$. Then,

$$\begin{aligned} L[i][v] \quad &= \quad \mathbf{min}\ \{L[i-1][v], loss_i\} \\ \text{where} \quad &\quad loss_i = \mathbf{DL}(s, s_i, I[i-1][v - volume_i], OptR, OptT, OptL), \\ &\quad volume_i = \mathbf{DV}(s_i, In_s, Subs, v). \end{aligned} \tag{8}$$

The function DV is used to dynamically calculate the volume of an item, which is shown in Algorithm 3. Moreover, the function DL shown in Algorithm 5 is applied to compute the loss of a temporary knapsack. The items which lead to the reduction of the loss will be selected to put into the knapsack.

By systematically increasing the values of i (from 1 to N) and v (from 1 to V_{cap}), the composition Ω_s^M will be finally obtained when $i = N$ and $v = V_{cap}$. Each search step is performed in this way layer by layer, and when the last search step is completed, the final composition $\Omega_{s_k}^M$ which reaches an ideal tradeoff among multiple objectives is obtained.

Table 1. The characteristics of datasets

Datasets	D-01	D-02	D-03	D-04	D-05	R-01	R-02	R-03	R-04	R-05
#Services	572	4129	8138	8301	15211	1000	3000	5000	7000	9000

5 Experimental Results

Extensive experiments have been carried out to evaluate the performance of the proposed method. To make the conclusion more convincing, experimental evaluations are carried out on two different groups of datasets, the datasets of the Web Service Challenge (WSC) 2009 and the datasets generated randomly.

5.1 Datasets

As shown in Table 1, the group of datasets of the WSC 2009 ranges from 572 to 15211 services. Considering that all the 5 datasets are generated by the same model, another group of random datasets ranging from 1000 to 9000 services are used to further evaluate the performance of our algorithm, which are available at https://wiki.citius.usc.es/inv:downloadable_results:ws-random-qos.

5.2 Validation of the Pre-computation Algorithms

The pre-computation algorithms are the foundation of our composition mechanism. Only by obtaining the optimization results of each single objective can we adopt the supervised algorithm to accomplish the multi-objective composition. Table 2 shows the optimization results of each single objective. *MS* is short for the memory-based search algorithm which is applied to generate compositions with the optimal global response time or throughput. Row *RT.sgl* shows the shortest response time for each dataset and *TP.sgl* the highest throughput. *KV* is short for the knapsack-variant algorithm that is proposed for compositions with the minimal number of services, and the row *Len.sgl* shows the results of *KV*. Moreover, the rows *Time* represent the execution times of every algorithm. For each dataset, the total time spent by the pre-computation is no more than 300 ms, which indicates the efficiency of the proposed *MS* and *KV*.

Table 2. The optimization results of single objectives

Datasets		D-01	D-02	D-03	D-04	D-05	R-01	R-02	R-03	R-04	R-05
MS	RT.sgl (ms)	500	1690	760	1470	4070	1430	975	805	1225	1420
	Time (ms)	0.6	1.2	0.9	1.9	1.4	0.5	0.9	2.1	1.9	2.9
	TP.sgl (inv/s)	15000	6000	4000	4000	4000	1000	2500	1500	2000	2500
	Time (ms)	0.4	0.9	0.7	1.6	1.5	0.3	0.7	1.6	1.9	2.8
KV	Len.sgl	5	20	10	40	30	7	15	12	14	16
	Time (ms)	102.1	57.4	20.2	81.1	53.3	19.6	121.3	204.7	256.9	297.1

5.3 Validation of the Supervised Algorithm

To validate our supervised algorithm, we compare it with three different state-of-the-arts in the same experimental setting. Table 3 shows the comparisons.

Table 3. Detailed comparisons with other methods

Datasets		D-01	D-02	D-03	D-04	D-05	R-01	R-02	R-03	R-04	R-05
Method in [15]	RT.mult (ms)	500	1690	760	1470	4070	1430	975	805	1225	1420
	TP.mult (inv/s)	3000	3000	2000	2000	1000	1000	1000	500	1000	500
	Len.mult	10	20	10	42	33	8	19	18	21	19
	Loss.comp	1.80	0.50	0.50	0.55	0.85	0.14	0.87	1.17	1.00	0.99
	RT.mult (ms)	840	2200	2450	4150	4990	1430	1305	1520	2095	1975
	TP.mult (inv/s)	15000	6000	4000	2000	4000	1000	2500	1500	2000	2500
	Len.mult	5	20	10	44	32	13	18	20	30	19
	Loss.comp	0.68	0.30	2.22	2.42	0.29	0.86	0.54	1.55	1.85	0.58
	Time.comp (ms)	11.2	18.3	11.8	41.9	16.1	6.7	23.1	51.7	87.6	76.5
Method in [14]	RT.mult (ms)	500	1690	760	1470	4070	1430	975	805	1225	1420
	TP.mult (inv/s)	3000	3000	2000	2000	1000	1000	2000	500	1000	1000
	Len.mult	8	21	10	42	33	9	18	15	20	18
	Loss.comp	1.40	0.55	0.50	0.55	0.85	0.29	0.40	0.92	0.93	0.73
	RT.mult (ms)	760	2270	1950	4080	4990	1430	1305	1520	1785	2165
	TP.mult (inv/s)	15000	6000	4000	2000	4000	1000	2500	1500	2000	2500
	Len.mult	5	20	21	40	30	11	18	15	23	17
	Loss.comp	0.52	0.34	2.67	2.28	0.23	0.57	0.54	1.09	1.10	0.59
	Time.comp (ms)	33.5	1491.3	1465.2	54351.7	936.9	45.6	493.7	767.1	941.6	375.3
Method in [16]	RT.mult (ms)	760	2270	1300	2140	5340	1580	1815	1640	1840	2300
	TP.mult (inv/s)	10000	6000	3000	1000	4000	1000	2000	1000	2000	1500
	Len.mult	6	21	12	47	36	9	18	17	19	20
	Loss.comp	1.05	0.39	1.16	1.38	0.51	0.39	1.26	1.73	0.86	1.27
	Time.comp (ms)	1.1	1.6	1.6	15.4	2.6	0.7	3.9	5.6	7.8	9.9
Our method	RT.mult (ms)	680	1800	760	1600	4260	1430	975	1090	1225	1605
	TP.mult (inv/s)	14000	6000	4000	3500	4000	1000	2000	1500	2000	2500
	Len.mult	5	20	10	43	33	8	16	15	17	18
	Loss.comp	**0.43**	**0.07**	**0.00**	**0.29**	**0.15**	**0.14**	**0.27**	**0.60**	**0.21**	**0.26**
	Time.comp (ms)	121.2	69.3	20.1	81.9	59.0	21.4	115.5	211.7	328.8	386.1

For each dataset, we mainly show solicitude for the global QoS of the generated composition (*RT.mult* for response time and *TR.mult* for throughput), the number of services in the composition (*Len.mult*), the loss of the composition (*Loss.comp*), and the execution time to extract the composition (*Time.comp*). Note that, for each dataset, both [14,15] can generate two different solutions (one with the optimal response time and another with the optimal throughput).

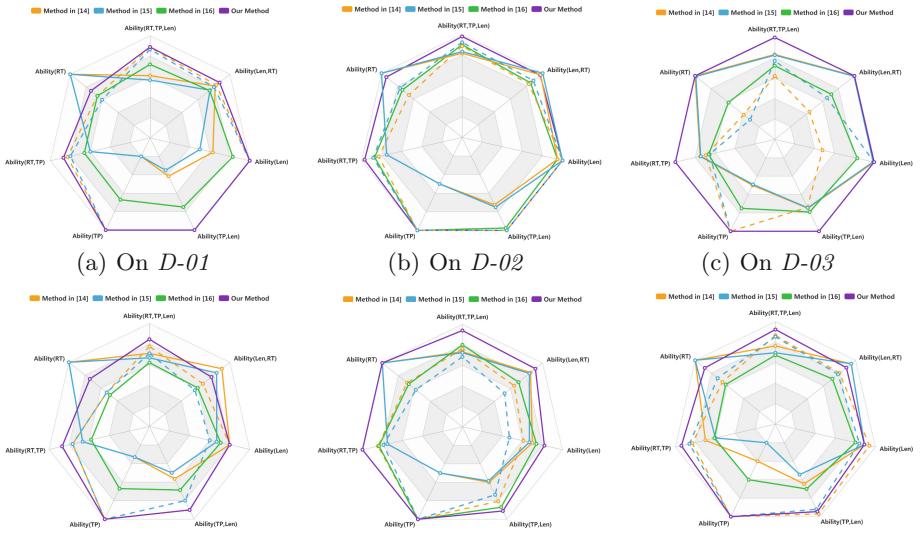
As can be seen in Table 3, the methods in [14,15] can generate compositions with the optimal response time (throughput) and the near-optimal length, however, the throughput (response time) of the compositions generated by these two methods are both exceedingly low (long) in contrast with the pre-computed *TP.sgl* (*RT.sgl*), which leads to an unsatisfactory loss. In [16], the optimization objective is to minimize the value of $(RT\text{-}TP + Len)$, therefore the generated compositions achieve a degree of tradeoff among the three attributes, but there

is still much room for improvement. Moreover, for each dataset, all the three attributes generated by our method change little from the optimal ones shown in Table 2. Therefore, our method outperforms the other methods by generating compositions with smaller loss, and it makes a better tradeoff among all the attributes. Even better, the solution on *D-03* owns the shortest response time, the highest throughput and the minimal number of services simultaneously.

To measure the performance of each method in terms of the response time, throughput, and length, we define $Ability(RT) = \frac{RT.sgl}{RT.mult}$, $Ability(TP) = \frac{TP.mult}{TP.sgl}$, and $Ability(Len) = \frac{Len.sgl}{Len.mult}$ respectively. In addition, we have *Ability(RT,TP) = Ability(RT) + Ability(TP)*. By analogy, *Ability(TP,Len)*, *Ability(Len,RT)*, and *Ability(RT,TP,Len)* are defined in the same manner to evaluate the performance of each method on several attributes simultaneously.

The greater the value of the measurements defined above, the better the performance of a method. On the basis of this, a series of radar charts shown in Fig. 4 are plotted to make the comparisons more intuitive. Note that, to differentiate the two solutions generated by the same method ([14] or [15]), the one with the optimal response time is drawn by a solid line while another with the optimal throughput is drawn by a dashed line. As can be seen from the figures, for each dataset, our method achieves a better tradeoff among the three objectives than the others owing to the fact that it has the greater value of *Ability(RT,TP,Len)* and it covers the lager area of each radar chart.

We further compare the efficiency of those methods. As shown in Fig. 5, our method isn't as efficient as the methods in [15,16], however it is, on average, over 40 times faster than [14]. For each dataset, the composition time of our method



(a) On *D-01* (b) On *D-02* (c) On *D-03*

(d) On *R-03* (e) On *R-04* (f) On *R-05*

Fig. 4. Radar charts to compare the performance of four methods on several datasets.

is no more than 400 ms, which proves that the supervised method can generate solutions within a reasonable execution time.

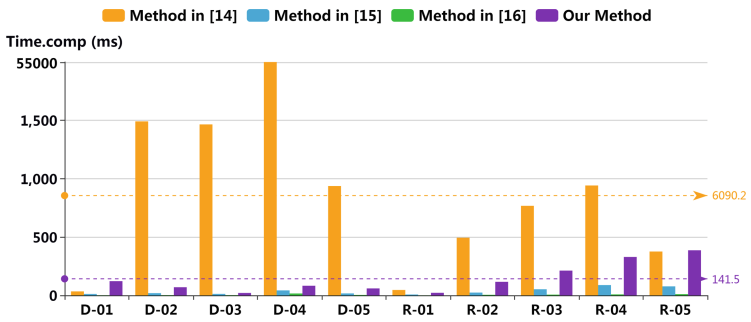


Fig. 5. Further comparisons in terms of the execution time.

6 Conclusions

In this paper, we propose an effective and efficient mechanism to solve the problem of multi-objective service composition taking service quantity into account. The mechanism combines a memory-based search algorithm and a knapsack-variant algorithm to optimize each single objective, after which a supervised algorithm is applied to solve the multi-objective composition on the basis of the pre-computed optimization results. A large number of experiments on two different groups of datasets show that our mechanism performs better than the state-of-the-arts, as it can generate compositions that reach an ideal compromise among multiple objectives including the number of services with high efficiency.

Acknowledgment. This work is funded by the Natural Science Foundation of China (No. 61673204), National Key R&D Program of China (No. 2018YFB1003800), State Grid Corporation of Science and Technology Projects (Funded No. SGLNXT00DKJS1700166), and the Program for Distinguished Talents of Jiangsu Province, China (No. 2013-XXRJ-018).

References

1. Wagner, F., Ishikawa, F., Honiden, S.: QoS-aware automatic service composition by applying functional clustering. In: 2011 IEEE International Conference on Web Services (ICWS), pp. 89–96. IEEE (2011)
2. Jiang, W., Zhang, C., Huang, Z., Chen, M., Hu, S., Liu, Z.: QSynth: a tool for QoS-aware automatic service composition. In: 2010 IEEE International Conference on Web Services (ICWS), pp. 42–49. IEEE (2010)
3. Rodriguez-Mier, P., Mucientes, M., Lama, M.: A dynamic QoS-aware semantic web service composition algorithm. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 623–630. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34321-6_48

4. Zou, G., Lu, Q., Chen, Y., Huang, R., Xu, Y., Xiang, Y.: QoS-aware dynamic composition of web services using numerical temporal planning. IEEE Trans. Serv. Comput. **7**(1), 18–31 (2014)
5. Strunk, A.: QoS-aware service composition: a survey. In: 2010 IEEE 8th European Conference on Web Services (ECOWS), pp. 67–74. IEEE (2010)
6. de Campos Jr., A., Pozo, A.T., Vergilio, S.R., Savegnago, T.: Many-objective evolutionary algorithms in the composition of web services. In: 2010 Eleventh Brazilian Symposium on Neural Networks (SBRN), pp. 152–157. IEEE (2010)
7. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng. **30**(5), 311–327 (2004)
8. Zhang, F., Hwang, K., Khan, S.U., Malluhi, Q.M.: Skyline discovery and composition of multi-cloud mashup services. IEEE Trans. Serv. Comput. **9**(1), 72–83 (2016)
9. Niu, S., Zou, G., Gan, Y., Xiang, Y., Zhang, B.: Towards uncertain QoS-aware service composition via multi-objective optimization. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 894–897. IEEE (2017)
10. Yu, Q., Bouguettaya, A.: Efficient service skyline computation for composite service selection. IEEE Trans. Knowl. Data Eng. **25**(4), 776–789 (2013)
11. Moustafa, A., Zhang, M.: Multi-objective service composition using reinforcement learning. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 298–312. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_21
12. Rodriguez-Mier, P., Mucientes, M., Lama, M.: Hybrid optimization algorithm for large-scale QoS-aware service composition. IEEE Trans. Serv. Comput. **10**(4), 547–559 (2017)
13. Rodriguez-Mier, P., Mucientes, M., Lama, M.: A hybrid local-global optimization strategy for QoS-aware service composition. In: 2015 IEEE International Conference on Web Services (ICWS), pp. 735–738. IEEE (2015)
14. Chen, M., Yan, Y.: Redundant service removal in QoS-aware service composition. In: 2012 IEEE 19th International Conference on Web Services (ICWS), pp. 431–439. IEEE (2012)
15. Xia, Y.M., Yang, Y.B.: Web service composition integrating QoS optimization and redundancy removal. In: 2013 IEEE 20th International Conference on Web Services (ICWS), pp. 203–210. IEEE (2013)
16. Chattopadhyay, S., Banerjee, A., Banerjee, N.: A scalable and approximate mechanism for web service composition. In: 2015 IEEE International Conference on Web Services (ICWS), pp. 9–16. IEEE (2015)
17. Fan, S., Yang, Y.: Efficient web service composition via knapsack-variant algorithm. arXiv preprint arXiv:1801.09102 (2018)