Large-Scale QoS-Aware Service Composition Integrating Chained Dynamic Programming and Hybrid Pruning

Shi-Liang Fan, Kai-Yu Peng, and Yu-Bin Yang(✉)

State Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210023, China
dyyslfan@smail.nju.edu.cn, yangyubin@nju.edu.cn

Abstract. Providing both optimal QoS and a minimum number of services simultaneously is a promising perspective of QoS-aware service composition, whereas most existing research studies are still unfavorable toward making an ideal trade-off between quality and efficiency, particularly in large-scale scenarios. To address this issue, this paper proposes a composition mechanism that effectively and efficiently minimizes the number of services in the composition result while achieving the optimal global QoS. We first transform the composition task into an equivalent one with decreased computing complexity, after which a chained dynamic programming algorithm, *Chain-DP*, is proposed to extract the optimal QoS with the minimum number of services. Finally, we further optimize the efficiency of the algorithm by adopting a global-local strategy of pruning. Experimental results on Web Service Challenge 2010's datasets show that the proposed method outperforms the state-of-the-art approach by generating solutions containing fewer services for the optimal QoS with higher efficiency and better generalization on large-scale datasets.

Keywords: Web service composition · QoS-aware · Large-scale

1 Introduction

The problem of QoS-Aware Web Service Composition [1–4] aims at obtaining a combination result with an optimal, single-criterion, end-to-end QoS when fulfilling a user's request. In large-scale scenarios, for a given request, the composition of a substantial number of services may generate numerous possible solutions with the same optimal QoS but different numbers of services [5].

Minimizing the number of services of the composition while satisfying the optimal QoS is a significant challenge because it has important benefits for brokers, customers, and service providers [6,7]. From the brokers' viewpoint, a composition result with fewer services can facilitate maintenance and management work; from the customers' point of view, a smaller composition ordinarily means

© Springer International Publishing AG, part of Springer Nature 2018
H. Jin et al. (Eds.): ICWS 2018, LNCS 10966, pp. 196–211, 2018.
https://doi.org/10.1007/978-3-319-94289-6_13

that a lower payment is demanded for the services. Thus, a decrease in the number of services included in the composition may greatly increase the success rate of achieving the desired responses to the requests of customers. From the service providers' viewpoint, solutions with fewer services can save resources and costs for the same task. However, a survey on QoS-aware service composition shows that the majority of studies aimed at optimizing the global QoS but rarely at minimizing the total number of services when the optimal QoS is guaranteed. To date, only a few studies have taken the optimization of both QoS and the number of services into consideration simultaneously. These existing methods are mainly divided into exact algorithms and approximate algorithms. Exact algorithms can generate compositions with a minimum number of services subject to the optimal QoS at the expense of a long running time, while approximate algorithms can only achieve near-optimal results.

As a matter of fact, minimizing the number of services while maintaining the optimal QoS leads to an NP-hard problem [8]. This will have a huge search space for optimal solutions in large-scale environments. In this paper, to make a good trade-off between quality and efficiency in large-scale scenarios, we propose a complete web service composition mechanism that effectively and efficiently minimizes the number of services in the composition result while achieving the optimal global QoS. The main contributions of this paper are as follows:

- An equivalent transformation approach is proposed, which transforms the problem of QoS-aware web service composition into a tractable one with decreased computing complexity.
- An optimal dynamic programming algorithm called Chain-DP is proposed, which guarantees to obtain the minimum number of services while holding the optimal global QoS based on the tractable problem after transformation.
- A global-local strategy of pruning is proposed, which greatly improves the efficiency of the above-mentioned Chain-DP algorithm by removing the redundant services and ignoring useless search space.

Furthermore, a full validation on WSC-2010's datasets is carried out. According to the experimental results, we can draw the conclusion that the proposed mechanism is effective and efficient in outperforming the state-of-the-art methods.

The organization of the rest of the paper is as follows. Section 2 reviews the related work. Section 3 illustrates the motivation of this research. Section 4 formally defines the composition problem. Section 5 presents the proposed mechanism in detail. Section 6 shows the experimental results.

2 Related Work

QoS-aware service composition has been studied by researchers from different perspectives. Most of the researchers merely concentrated on the optimization of the global QoS [9,10]. There are only a few studies that attempted to optimize the total number of services while meeting the optimal QoS.

In [11], an approximate mechanism was presented to obtain close-to-optimal solutions against time. The authors proposed an on-the-fly algorithm to construct only a path of the auxiliary graph instead of the complete graph. Additionally, a deterministic approach and a probabilistic approach were discussed to find the path with the near-optimal QoS and number of services, which was chosen as the final composition. Although the algorithm had a superior composition time, the greedy strategy adopted was always stuck in local optima.

Yan et al. [12] proposed an algorithm that combined a systematic search algorithm with a planning algorithm called GraphPlan. This method could find and remove redundant services while achieving both functional goals and QoS optimization. Inspired by this, a redundant service removal mechanism was presented by Chen and Yan [13]. This method first modeled a composition problem as an integer programming problem (IP), and then obtained a composition whose global QoS was optimal by solving the IP. The next step was to remove redundant services in the composition while keeping the optimal QoS. This method is not generally applicable to the real-time situation. Thus, the authors in [14] proposed a modified approach to overcome the above drawback. In this approach, the process of redundancy removal was performed in parallel with the process of service composition, which had gone some way toward improving the efficiency. The approximate algorithms could reduce redundant services in compositions more or less, but still failed to work out optimal results.

A recent approach using exact algorithms was proposed by Rodriguez-Mier et al. [6]. A hybrid local-global strategy was presented to optimize the QoS-aware service composition problem. Although the local search strategy could only find a solution with a near-optimal number of services, it was a fast algorithm that saved plenty of time. The global search strategy could improve the solution generated by the local search, but it took a longer time to minimize the total number of services for the optimal QoS. Compared with approximate algorithms, the hybrid strategy can generate solutions with fewer services while guaranteeing the optimal QoS. However, the exhaustive combinatorial search makes it difficult to obtain the solution in a short time, especially in large-scale scenarios.

In summary, the above algorithms all suffer from many disadvantages. The approximate algorithms are fast but cannot generate results with a minimum number of services, while the exact algorithm is optimal but not adequately efficient. Therefore, there is a lack of approaches that have the ability to optimize the global QoS, as well as to minimize the number of services of the composition, effectively and efficiently.

3 Motivating Scenario and Analysis

As shown in Fig. 1, an example is described as a directed graph. The example is actually conducted to solve a practical classification problem whose goal is to predict if a client will subscribe to a term deposit in a bank. The problem of classification is treated as a problem of QoS-aware service composition whose request is R where the input is $\{ont1{:}UserID\}$ and the output is $\{xsd{:}boolean\}$.

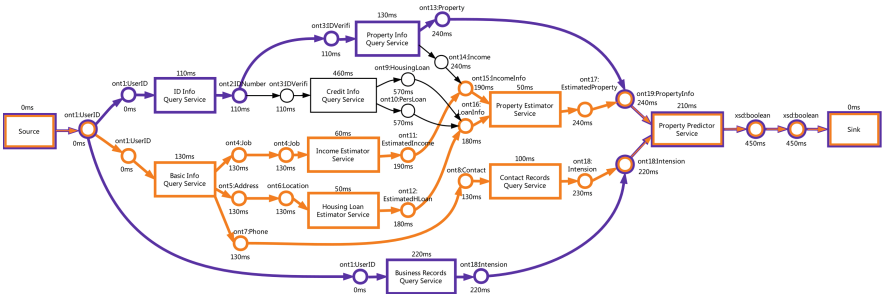


Fig. 1. Example of a *Service Dependency Graph*. (Color figure online)

Every element in the graph has its unique meaning. 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 represent the matching relations between services.

Returning to the original problem of classification in Fig. 1, there are two different solutions of composition fulfilling the request R, which are highlighted in two different colors. The composition highlighted in orange has the optimal global response time (450 ms) and contains eight services in total (including the *Source* and the *Sink*). The composition highlighted in purple owns the same response time but contains only six services. In addition to the compositions highlighted in the graph, there are others with a response time of 450 ms, whereas their numbers of services are unexceptionally more than six. To sum up, the composition highlighted in purple is the optimal solution with a minimum number of services while guaranteeing the lowest response time.

In large-scale scenarios, the directed graph as in Fig. 1 may be exceedingly intricate and complex, which leads to a huge search space. As a result, it is formidable to extract the optimal composition from the graph. An exhaustive combinatorial search can guarantee the optima, but will take an unacceptable length of time to generate the compositions. In the process of service composition, in order to improve the efficiency, many measures can be taken to reduce the useless search space. For instance, in Fig. 1, the optimal composition highlighted in purple has a response time of 450 ms, while the response time of the service *Credit Info Query Service* is 460 ms. Therefore, *Credit Info Query Service* can be removed from the graph because it will not make any contribution to the optimal composition. In a word, we pay attention not only to the quality of the resulting composition but also to the efficiency of the composition algorithm.

4 Preliminaries

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 that are the measures for how well the service s serves the user.*

Relevant services can be combined by connecting matched inputs and outputs.

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 subconcept of* $in_{s'}$*,* out_s *matches* $in_{s'}$ *(i.e.,* $in_{s'}$ *is matched by* out_s*).*

There are two main kinds of structures of the composition, namely, the *sequential structure* and the *parallel structure*. The services organized as sequential structures are invoked in order, while those in parallel structures 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 *QoS-aware web service composition* in this paper is provided.

Definition 3. *QoS-Aware Web Service Composition is defined as follows: for a given composition request* $R = \{In_R, Out_R\}$*, to seek a composition* Ω *with optimization objectives of (1)* $\mathbf{min}\, RT(\Omega)$ *or* $\mathbf{max}\, TR(\Omega)$ *and (2)* $\mathbf{min}\, Len(\Omega)$*.*

5 Framework

5.1 Generation of the Service Dependency Graph

For a given user request $R = \{In_R, Out_R\}$, a service dependency graph [15,16] similar to that in Fig. 1 is constructed to show the input-output

dependencies between 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 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.

5.2 Generation of Subproblems

Description of Optimal Substructure. Once the service dependency graph G is completed for a request R, the composition problem is treated as finding a path with the optimal QoS and the minimum number of services starting from s_o and ending with s_k in the graph. Obviously, a path Λ in G is actually a composition Ω in Definition 2. An abstract path with the optimal *quality* from the service s_o to a service s in the graph is expressed as Λ_s. As shown in Fig. 2, to obtain the path Λ_{s_k}, the set of paths containing Λ_X and Λ_{XI} needs to be determined in advance. Similarly, the path Λ_{XI} depends on Λ_{IX}, while the rest can be done in the same manner.

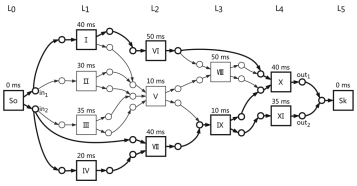


Fig. 2. Example graph.

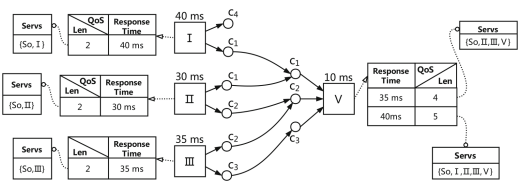


Fig. 3. Subproblem V.

Definition 4. *The set of precursors of a service $s \in L_i$ (the i-th layer) is defined as $Pre(s) = \{s' \mid s' \in L_j(\forall j < i) \wedge In_s \cap Out_{s'} \neq \varnothing\}$. Specifically, $Pre(s_o) = \varnothing$.*

On the basis of the above definition, for a service s, if the paths $\Lambda_{Pre(s)} = \{\Lambda_{s'} \mid s' \in Pre(s)\}$ have already been determined, the decision-making process of the optimal path Λ_s is regarded as a subproblem named s.

Definition 5. *The set of feasible precursor-decisions of a subproblem s is defined as $P_s = \{p_s \mid p_s \subseteq Pre(s) \wedge In_s \subseteq \bigcup_{s' \in p_s} Out_{s'}\}$.*

As a sequence, the composition problem is divided into many subproblems. The solution of a subproblem depends on the optimization results of subproblems in previous layers.

Definition and Generation of Subproblems. We bring up the abstract concept named *quality* of a path solely to explain the idea of the optimal substructure. Assuming that only the optimal QoS is maintained for each path, we will lose the optimal solution. For instance, there are two compositions in Fig. 2:

the composition $\Omega = s_o \rightarrow [II \parallel III] \rightarrow V \rightarrow IX$, whose global response time $RT(\Omega)$ is 55 ms, as well as another composition $\Omega' = s_o \rightarrow IV \rightarrow VII \rightarrow IX$ that has a global response time of $RT(\Omega'_s) = 70$ ms. If the quality of a path is measured merely by the global response time, the optimal path Λ_{IX} is actually Ω, which causes a loss of the optimal path highlighted in the graph. Accordingly, to minimize the number of services simultaneously, we design a dissimilar way to describe a path. A concrete path Λ^l_s starts from the service s_o and ends with a service s. In addition, Λ^l_s has the optimal QoS among those paths whose *length* (the number of services) is l. Let us reconsider the above example in this way. The path $\Lambda^4_{IX} = s_o \rightarrow IV \rightarrow VII \rightarrow IX$ has a response time of 70 ms. There are two different paths with the same length of 5: $\Lambda = s_o \rightarrow [I \parallel III] \rightarrow V \rightarrow IX$ with $RT(\Lambda) = 60$ ms, and $\Lambda' = s_o \rightarrow [II \parallel III] \rightarrow V \rightarrow IX$ with $RT(\Lambda') = 55$ ms. Thus, $\Lambda^5_{IX} = \Lambda' = s_o \rightarrow [II \parallel III] \rightarrow V \rightarrow IX$ owing to $RT(\Lambda') < RT(\Lambda)$. By this means, the path with minimum number of services for the optimal QoS can be kept.

Then, a subproblem s is defined as determining a cluster of concrete paths $\Lambda^L_s = \{\Lambda^l_s \,|\, l \in L\}$ $(L = \{1, \ldots, |G|\})$. Each path Λ^l_s in Λ^L_s is determined by several paths selected from $\Lambda^L_{s'}$, where $s' \in Pre(s)$. For a feasible precursor decision $p_s = \{s_1, \ldots, s_n\}$, let $Cart(p_s) = \Lambda^L_{s_1} \times \ldots \times \Lambda^L_{s_n}$, where $\times$ represents the operation of a Cartesian product. Taking the optimization of the response time as an example, the detailed optimization model of Λ^l_s is

$$RT(\Lambda^l_s) = \min_{p_s \in P_s} \{ \min_{n_{p_s} \in N_{p_s}} \{ \max_{\Lambda \in n_{p_s}} RT(\Lambda) \} \} \; + \; RT(s). \tag{3}$$

where $N_{p_s} = \{N \,|\, N \in Cart(p_s) \wedge |\bigcup_{\Lambda \in N} Servs(\Lambda)| = l - 1\}$ is the set of feasible length-decisions of p_s, and $Servs(\Lambda)$ memorizes all the services in the path Λ.

Table 1. Optimization process for Λ^4_V.

p_s	N_{p_s}	n_{p_s}	$RT(\Lambda^4_V)$
$\{I, III\}$	$\{(\Lambda^2_I, \Lambda^2_{III})\}$	$(\Lambda^2_I, \Lambda^2_{III})$	40 ms
$\{II, III\}$	$\{(\Lambda^2_{II}, \Lambda^2_{III})\}$	$(\Lambda^2_{II}, \Lambda^2_{III})$	**35 ms**
$\{I, II, III\}$	$\varnothing$	$\varnothing$	$\times$

The subproblem V is shown in Fig. 3. Table 1 shows the optimization process of Λ^4_V. According to the table, Λ^4_V is generated by combining Λ^2_{II} and Λ^2_{III}.

The generation process for subproblems is described in Algorithm 1, where *out_serv_map* is a precomputed table that maps each output to those services that own this output.

5.3 Transformation of Subproblems

It is inconvenient for set operations to identify whether a precursor decision is feasible for a given subproblem. Considering the subproblem s, a precursor

Algorithm 1. Subproblem Generation

Input: G, out_serv_map
Output: $subs\_map$

```
1  subs_map ← {}, visited ← {}
2  for index = 0; index < |G|; index++ do
3      for service s ∈ L_index do
4          pres ← {}
5          for concept c ∈ In_s do
6              servs ← out_serv_map[c]
7              for service s' ∈ servs do
8                  if s' ∈ visited and s' ∉ pres then
9                      pres ← pres ∪ {s'}
10         subs_map[s] ← pres
11     for service s ∈ L_index do
12         visited ← visited ∪ {s}
13 return subs_map
```

decision $p_s(p_s \subseteq Pre(s))$ is feasible if $In_s \subseteq \bigcup_{s' \in p_s} Out_{s'}$, which is too complicated to be applied to the following dynamic programming algorithm. Therefore, an **equivalent transformation approach** is proposed to overcome the obstacle.

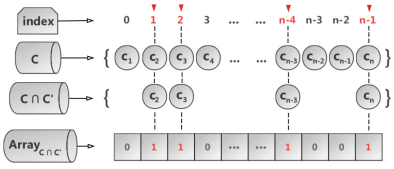


Fig. 4. Schematic diagram.

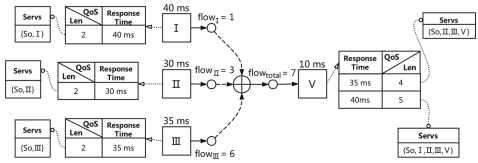


Fig. 5. V after transformation.

Definition 6. *As shown in Fig. 4, given a set of concepts C and another set C', where $C' \cap C \neq \varnothing$, the contribution made by C' to C is defined as*

$$\Delta_C(C') = \sum_{index=0}^{n-1} Array_{C \cap C'}[index] \times 2^{index}. \tag{4}$$

where $\Delta_C(C) = 2^n - 1$. Conversely, if the contribution made by an unknown set X to the set C is $\Delta_C(X)$, the intersection of C and X is calculated as follows:

$$\left.\begin{aligned} \Delta_C(X) &= \sum_{index=0}^{n-1} a_{index} \times 2^{index} \\ \Phi_C(\Delta_C(X)) &= \{C[index] \mid a_{index} = 1\} \end{aligned}\right\}. \tag{5}$$

More examples are shown to further illustrate the above approach. Given two sets of concepts $C = \{c_1, c_2, c_3, c_4\}$ and $C' = \{c_2, c_4, c_5\}$, $\Delta_C(C')$ equals 10 after calculation. Moreover, if an unknown set X makes a contribution $\Delta_C(X) = 5$ to the set C, we first complete the decimal-to-binary conversion of $\Delta_C(X)$, namely, $5 = 1 \times 2^0 + 1 \times 2^2$. Then, $C \cap X = \Phi_C(5) = \{c_1, c_3\}$. There is a phenomenon such that $\Delta_C(C') + \Delta_C(X) = \Delta_C(C)$. Meanwhile, $C \subseteq (C' \cup X)$.

Lemma 2. *Given a set of concepts C, as well as two other sets C' and C'', if $\Delta_C(C') + \Delta_C(C'') = \Delta_C(C)$, then $C \subseteq (C' \cup C'')$.*

Next, a concept called the *flow* of a path is introduced to transform each subproblem into another one, which avoids making use of set operations.

Definition 7. *For a given subproblem s, the flow of a path $\Lambda^l_{s'}$ is defined as $flow_{s'} = \Delta_{In_s}(Out_{s'})$, where $s' \in Pre(s)$. Moreover, the flow of the objective path Λ^l_s is defined as $flow_{total} = \Delta_{In_s}(In_s)$.*

The subproblem V after transformation is shown in Fig. 5. Operator $\oplus$ in the figure is used to calculate the flow of a temporary path after composition. If a temporary path Λ^3_{tmp} ($s_o \rightarrow [I \parallel II]$) is generated by combining Λ^2_I and Λ^2_{II}, the flow of Λ^3_{tmp} is $flow_{tmp} = flow_I \oplus flow_{II} = \Delta_{In_V}(\Phi_{In_V}(flow_I) \cap \Phi_{In_V}(flow_{II}))$. In this subproblem, we expect to obtain a set of paths Λ^L_V where each Λ^l_V owns the flow of 7 after composition.

Hence, P_s changes into $P_s = \{p_s \,|\, p_s \subseteq Pre(s) \wedge \sum_{\oplus_{s' \in p_s}} flow_{s'} = flow_{total}\}$, where $\sum_{\oplus}$ acting on the set $p_s = \{s'_1, \ldots, s'_n\}$ is short for $flow_{s'_1} \oplus \ldots \oplus flow_{s'_n}$.

5.4 Chained Dynamic Programming Algorithm

For a subproblem s, there is no doubt that it is hardly desirable to explore all possible combinations of paths to get the set of feasible precursor-decisions P_s, as well as the set of feasible length-decisions N_{p_s} for each $p_s \in P_s$, especially in large-scale scenarios. However, according to Lemma 2, each subproblem can be further divided into a series of subproblems. Let $Pre(s) = \{s_1, s_2, \ldots, s_n\}$. Considering that the set of paths $\Lambda^L_{s_n}$ is known, if the set of paths Λ^L_s whose flow equals $flow_{total} - flow_{s_n}$ has been already determined by combining the paths selected from $\Lambda^L_{s_i} (1 \leq i \leq n-1)$, paths Λ^L_s with a flow of $flow_{total}$ can be easily determined by combing the above two sets of paths.

Note that for each subproblem s, to differentiate the objective paths with different flows, the set of paths Λ^L_s with a flow of f ($f \leq flow_{total}$) is expressed as $\widetilde{\Lambda}^L_f$ hereafter. Moreover, if a path $\widetilde{\Lambda}^l_f \in \widetilde{\Lambda}^L_f$ is expected to be found in a subproblem of s, then the flow of each $\Lambda^l_{s_i}$ ($1 \leq i \leq n$) should be updated as $flow^f_{s_i} = \Delta_{In_s}(Out_{s_i} \cap \Phi_{In_s}(f))$. For example, as we can see from the subproblem V, $flow^7_{II} = 3$ while $flow^1_{II} = 1$.

On the basis of the above description, a chained dynamic programming algorithm is proposed. Taking the optimization of the response time as an example,

let $F[i][f][l]$ represent the response time of the path $\widetilde{\Lambda}_f^l$ generated by combining the first i clusters of known paths $(\Lambda_{s_1}^L, \Lambda_{s_2}^L, \ldots, \Lambda_{s_i}^L)$. It can be shown that

$$F[i][f][l] = \min_{\substack{l',l''\in L \\ U(l',l'')=l}} \{\mathbf{max}\{F[i-1][f-flow_{s_i}^f][l'], RT(\Lambda_{s_i}^{l''})\}\}. \tag{6}$$

where $U(l',l'') = |Servs(\widetilde{\Lambda}_{f-flow_{s_i}^f}^{l'}) \cup Servs(\Lambda_{s_i}^{l''})|$. Several candidate paths whose flow is f are obtained by combining $\widetilde{\Lambda}_{f-flow_{s_i}^f}^{l'}$ (for each $l' \in L$) and $\Lambda_{s_i}^{l''}$ (for each $l'' \in L$). $\widetilde{\Lambda}_f^l$ is the one with the optimal response time while owning the length of l. Thus, for fixed i and f, paths $\widetilde{\Lambda}_f^L$ can be determined with a time complexity of $O(|L|^2)$. By systematically increasing the values of i (from 1 to n) and f (from 1 to $flow_{total}$), the desired path Λ_s^L will finally be obtained when $i = n$ and $f = flow_{total}$. For each $\Lambda_s^l \in \Lambda_s^L$,

$$RT(\Lambda_s^l) = F[n][flow_{total}][l] + RT(s). \tag{7}$$

Therefore, the subproblem s is solved. The solved subproblems are known conditions of those unsolved; therefore, a chain of decisions is made from s_o to s_k, which is the reason the algorithm is named after **Chain-DP**.

5.5 Global-Local Pruning Strategy

When applying a dp algorithm, it is probable that numbers of useless subproblems are solved, or many idle search spaces are explored. A **global-local pruning strategy** is adopted to further improve the efficiency of the Chain-DP.

As can be seen from Definition 3, the optimal global QoS is the essential prerequisite for seeking the expected composition. We first propose a fast preprocessing approach to compute the optimal global QoS of each path without consideration of the length. In the dependency graph, a path with the optimal QoS from the service s_o to a service s is expressed as $\widehat{\Lambda}_s$. Then, the approach is shown in Algorithm 2, taking the preprocessing of the optimal response time as an example. The optimal response time of each path is stored in a hash table Opt_RT where $Opt\_RT[s] = RT(\widehat{\Lambda}_s)$. Moreover, for each input $in_s \in In_s$, $RT\_in[in_s]$ stores the shortest response time to generate in_s, that is to say, $RT\_in[in_s] = \min_{s'\in Pre(s) \ \wedge \ in_s \in Out_{s'}} RT(\widehat{\Lambda}_{s'})$. Therefore, the decision-making process for the path $\widehat{\Lambda}_s$ can be described as $RT(\widehat{\Lambda}_s) = \max_{in_s \in In_s} \{RT\_in[in_s]\} + RT(s)$.

Global pruning is applied to lessen the number of redundant subproblems, which further reduces the number of useless services in the graph G.

Lemma 3. *For each $s \in G$, if $RT(\widehat{\Lambda}_s) > RT(\widehat{\Lambda}_{s_k})$ or $TR(\widehat{\Lambda}_s) < TR(\widehat{\Lambda}_{s_k})$, the service s will not be involved in the final composition.*

According to Lemma 3, a service s can be removed from G if $Opt\_RT[s] > Opt\_RT[s_k]$. For example, in Fig. 1, the final path highlighted in purple has a

Algorithm 2. Optimal QoS Preprocessing

```
Input: G, subs_map
Output: Opt_RT
Opt_RT ← {s_o : 0}
for index = 1; index < |G|; index++ do
    for service s ∈ L_index do
        tmp_RT ← 0
        for concept c ∈ In_s do
            RT_in[c] ← +∞
        for concept c ∈ In_s do
            for service s̃ ∈ subs_map[s] do
                if c ∈ Out_s̃ and Opt_RT[s̃] < RT_in[c] then
                    RT_in[c] ← Opt_RT[s̃]
            tmp_RT ← max(tmp_RT, RT_in[c])
        Opt_RT[s] ← tmp_RT + RT(s)
return Opt_RT
```

global response time of 450 ms, while the path ending with *Credit Info Query Service* has an optimal response time of 570 ms. Obviously, the *Credit Info Query Service* is not involved in the final path. In addition, eliminating it from the graph will not make a difference in the optimal solution of the composition.

Local pruning aims at reducing the search space of each subproblem.

Lemma 4. *In the decision-making process for each subproblem s, for each $s' \in Pre(s)$ and each $l \in L$, the path $\Lambda_{s'}^{l}$ will make no contribution to the cluster of paths Λ_s^L on the condition that $RT(\Lambda_{s'}^{l}) + RT(s) > RT(\widehat{\Lambda}_s)$ and $l + 1 > |Servs(\widehat{\Lambda}_s)|$, or $TR(\Lambda_{s'}^{l}) < TR(\widehat{\Lambda}_s)$ and $l + 1 > |Servs(\widehat{\Lambda}_s)|$.*

On the basis of Lemma 4, local pruning is applied as follows. For each subproblem s, when determining the cluster of paths Λ_s^L, a precursor path $\Lambda_{s'}^{l}$ can be disregarded, subject to the following constraints: (1) $Rets[s'][l] + RT(s) > Opt\_RT[s]$ and (2) $l + 1 > |Opt\_Paths[s]|$.

6 Experimental Evaluation

We completed three groups of experiments on the datasets of Web Service Challenge (WSC) 2010 to evaluate the performance of the proposed mechanism. The groups of experiments that were sequentially constructed are as follows: (1) validation of the global-local pruning strategy, (2) validation of the Chain-DP from the aspects of results and efficiency, and (3)validation of the applied scenarios.

6.1 Datasets

WSC 2010's datasets range from 572 to 15211 services. Each dataset contains a WSDL file defining the inputs and outputs of the services, a WSLA file storing

the QoS values (response time and throughput) of the services, and an OWL file describing the matching relations between all of the inputs and outputs.

6.2 Validation of the Hybrid Pruning Strategy

We first constructed experiments to validate the global-local pruning strategy.

Table 2. Comparisons of number of services before and after pruning.

WSC-2010's datasets		D-01	D-02	D-03	D-04	D-05
#Graph services		77	135	146	321	225
Validation with resp. time	#Graph Services (opt)	45	100	81	210	210
	Preprocessing time (ms)	1.8	1.6	2.1	4.9	3.1
Validation with throughput	#Graph Services (opt)	11	63	100	174	107
	Preprocessing time (ms)	1.3	1.7	1.6	4.0	4.6

On the one hand, we focused on the reduction of services in the dependency graph after pruning. Moreover, when performing the experiment, we measured the execution time of the preprocessing approach in passing. Table 2 lists the results obtained for each dataset and for each QoS property. Row *#Graph Services* shows the initial number of services in the graph, and *#Graph Services (opt)* the number of services after pruning. As can be seen, the number of services is reduced, on average, by 39% via pruning. Row *Preprocessing Time* shows the execution time of the preprocessing approach. It is evident from the table that the extra time spent by the preprocessing of pruning is no more than 5 ms.

Table 3. Comparison of efficiency of Chain-DP before and after pruning.

WSC-2010's datasets		D-01	D-02	D-03	D-04	D-05
Validation with resp. time	Chain-DP (ms)	256	2125	1027	9259	22463
	Chain-DP-pruning (ms)	57	60	86	209	528
Validation with throughput	Chain-DP (ms)	259	2074	1152	21332	25168
	Chain-DP-pruning (ms)	27	73	179	308	321

On the other hand, for each dataset and for each QoS, we compared the efficiency of Chain-DP before and after pruning. As shown in Table 3, row *Chain-DP* shows the execution time of the Chain-DP algorithm without pruning,

and *Chain-DP-Pruning* shows the execution time of Chain-DP with global-local pruning. The results indicate that Chain-DP with global-local pruning is, on average, over 30 times faster than Chain-DP without pruning.

In conclusion, the above experiments indicate that the proposed pruning strategy can effectively remove the redundant services in the dependency graph, and can also significantly reduce the search space of the composition problem. As a result, the strategy is powerful in improving the efficiency of Chain-DP.

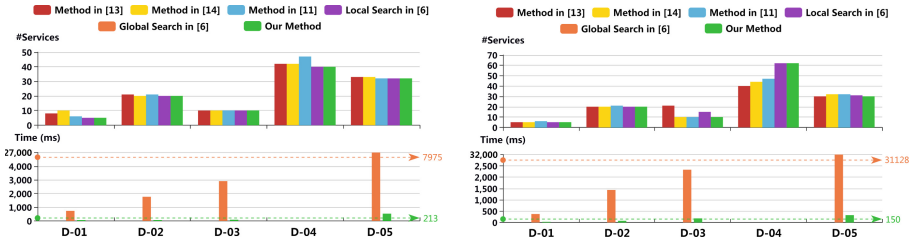
6.3 Validation of the Chain-DP Algorithm

To validate our chained dynamic programming algorithm, we compared our approach with five different approaches in the same experimental environment.

Table 4. Comparisons with other approaches.

Datasets		D-01	D-02	D-03	D-04	D-05		D-01	D-02	D-03	D-04	D-05
	Validation with response time						Validation with throughput					
Method in [13]	RT (ms)	500	1690	760	1470	4070	TR (inv/s)	15000	6000	4000	2000	4000
	#Services	8	21	10	42	33	#Services	5	20	21	40	30
	Time (ms)	27	1491	12	54352	737	Time (ms)	7	158	1380	76125	735
Method in [14]	RT (ms)	500	1690	760	1470	4070	TR (inv/s)	15000	6000	4000	2000	4000
	#Services	10	20	10	42	33	#Services	5	20	10	44	32
	Time (ms)	9	14	8	21	11	Time (ms)	8	14	13	31	15
Method in [11]	RT (ms)	760	2270	1300	2140	5340	TR (inv/s)	10000	6000	4000	1000	4000
	#Services	6	21	10	47	32	#Services	6	21	10	47	32
	Time (ms)	1	2	1	14	3	Time (ms)	1	2	1	14	3
Local search in [6]	RT (ms)	500	1690	760	1470	4070	TR (inv/s)	15000	6000	4000	4000	4000
	#Services	5	20	10	40	32	#Services	5	20	15	62	31
	Time (ms)	711	1098	2877	8475	3212	Time (ms)	385	1325	2235	9256	2715
Global search in [6]	RT (ms)	500	1690	760	-	4070	TR (inv/s)	15000	6000	4000	-	4000
	#Services	5	20	10	-	32	#Services	5	20	10	-	30
	Time (ms)	738	1765	2907	-	26491	Time (ms)	374	1434	2330	-	120375
Our method	RT (ms)	**500**	**1690**	**760**	**1470**	**4070**	TR (inv/s)	**15000**	**6000**	**4000**	**4000**	**4000**
	#Services	**5**	**20**	**10**	**40**	**32**	#Services	**5**	**20**	**10**	**62**	**30**
	Time (ms)	57	60	86	209	528	Time (ms)	27	73	179	308	321

Table 4 shows all of the comparisons. For each dataset and for each QoS property, we mainly paid close attention to the global QoS of the obtained composition (*RT* for the response time and *TR* for throughput), the number of services included in the obtained composition (*#Services*), and the execution time to extract a composition from a graph (*Time*). A composition is better if (1) its global QoS is better, or (2) it owns the same global QoS but fewer services. It can be seen that our approach can always generate the same or better compositions. As an exact algorithm, the global search proposed in [6] could not find a solution for *D-04* owing to a combinatorial explosion, while our approach succeeded in finding one with better throughput (4000 inv/s) than other methods, except for the local search [6].



(a) Validation with Response Time. (b) Validation with Throughput.

Fig. 6. Further comparison with other approaches. (Color figure online)

Considering that almost all of the methods in Table 4 could find compositions with the same global QoS, we further compared our method with those approximate algorithms in terms of the number of services, and also compared our method with the exact algorithm in terms of the execution time. Figure 6 shows that compared with the approximate algorithms, our algorithm always found compositions with the same or fewer services while guaranteeing the optimal global QoS. In addition, compared with the global search, it took far less time to generate solutions. The orange line represents the average execution time of the global search, while the green line shows the average time of Chain-DP with hybrid pruning. Our algorithm is, on average, over 35 times faster than the global search without regard to *D-04*, which is a significant improvement. In summary, our algorithm achieves an ideal trade-off between quality and efficiency.

6.4 Validation of the Applicable Scenarios

Lastly, we tested and validated the applicable scenarios that the proposed mechanism can be generalized to use.

To evaluate the effects of hybrid pruning in different scenarios, we define an indicator as

$$Ratio\_PR = \frac{Time\_WO}{Time\_WI}. \tag{8}$$

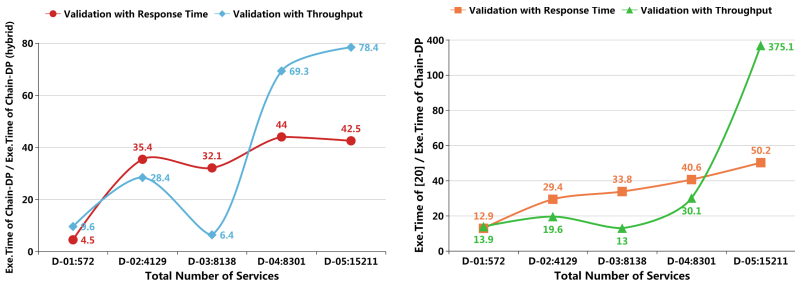
where $Time\_WO$ represents the execution time of Chain-DP without pruning, and $Time\_WI$ the execution time of Chain-DP with hybrid pruning. As can be seen from Fig. 7(a), the larger the size of a dataset, the better the hybrid pruning strategy performs.

Furthermore, in order to compare Chain-DPs efficiency with that of the global search in different scenarios, we similarly define another indicator:

$$Ratio\_CP = \frac{Time\_GS}{Time\_DP}. \tag{9}$$

where $Time\_GS$ represents the execution time of the global search, and $Time\_DP$ the execution time of Chain-DP with hybrid pruning. Seeing that the global search could not find a solution for *D-04*, we used the execution time

of the local search instead. As shown in Fig. 7(b), with an increase in the size of the datasets, the advantage of our Chain-DP becomes progressively obvious. For *D-05* (Validation with Throughput), Chain-DP is even two orders of magnitude faster than the global search while guaranteeing the same optimal solution.



(a) *Ratio_PR* for WSC datasets (b) *Ratio_CP* for WSC datasets

Fig. 7. Validation of applied scenarios.

The above experiments indicate that both the hybrid pruning strategy and the Chain-DP algorithm can be easily generalized and applied to a variety of scenarios, especially large-scale scenarios.

7 Conclusion

In this paper, we proposed an effective and efficient mechanism to automatically generate compositions by minimizing the number of services while simultaneously satisfying the optimal global QoS. The mechanism combines a global-local pruning strategy and a chained dynamic programming algorithm to extract the optimal composition from the service dependency graph with high efficiency. A large number of experiments on two different groups of datasets show that our mechanism performs better than the state-of-the-art methods, as it not only obtains compositions with fewer services for the optimal QoS than the approximate algorithms, but also executes much faster than an exact algorithm while obtaining nearly the same results. It is proven to achieve a very good trade-off between quality and efficiency in various scenarios, especially in large-scale scenarios.

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. Ma, H., Jiang, W., Hu, S., Huang, Z., Liu, Z.: Two-phase graph search algorithm for QoS-aware automatic service composition. In: 2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–4. IEEE (2010)
2. Strunk, A.: QoS-aware service composition: a survey. In: 2010 IEEE 8th European Conference on Web Services (ECOWS), pp. 67–74. IEEE (2010)
3. 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)
4. 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)
5. 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)
6. 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)
7. Fan, S., Yang, Y.: Efficient web service composition via knapsack-variant algorithm. arXiv preprint arXiv:1801.09102 (2018)
8. 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)
9. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient QoS-aware service composition. In: Proceedings of the 18th International Conference on World Wide Web, pp. 881–890. ACM (2009)
10. 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)
11. 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)
12. Yan, Y., Chen, M., Yang, Y.: Anytime QoS optimization over the PlanGraph for web service composition. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 1968–1975. ACM (2012)
13. 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)
14. 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)
15. Rodriguez-Mier, P., Mucientes, M., Vidal, J.C., Lama, M.: An optimal and complete algorithm for automatic web service composition. Int. J. Web Serv. Res. (IJWSR) **9**(2), 1–20 (2012)
16. Rodriguez-Mier, P., Pedrinaci, C., Lama, M., Mucientes, M.: An integrated semantic web service discovery and composition framework. IEEE Trans. Serv. Comput. **9**(4), 537–550 (2016)