Multi-Objective Service Composition with QoS Dependencies

Ying Chen, Student Member, IEEE, Jiwei Huang, Member, IEEE, Chuang Lin, Senior Member, IEEE, and Xuemin (Sherman) Shen, Fellow, IEEE

Abstract—Service composition is popular for composing a set of existing services to provide complex services. With the increasing number of services deployed in cloud computing environments, many service providers have started to offer candidate services with similar but different Quality of Service (QoS) levels. Therefore, QoS-aware service composition has drawn extensive attention. Most existing approaches for QoS-aware service composition assume a service’s QoS values are not correlated to those of other services. However, QoS dependency exists in real life, and impacts the overall QoS values of the composite services. In this article, we study QoS dependency-aware service composition considering multiple QoS attributes. Based on the Pareto set model, we focus on searching for a set of Pareto optimal solutions. A candidate pruning algorithm for removing the unpromising candidates is proposed, and a service composition algorithm using Vector Ordinal Optimization techniques is designed. Simulation experiments are conducted to validate the efficiency and effectiveness of our algorithms. We are the first to take advantage of Vector Ordinal Optimization techniques to search for Pareto optimal composition solutions with QoS dependency involved. The capturing of QoS dependency enables us to find truly desirable solutions.

Keywords—service composition; QoS dependency; Pareto optimal solutions; Vector Ordinal Optimization; multiple QoS attributes; multi-objective optimization; candidate pruning

1 INTRODUCTION

Cloud computing is a new paradigm providing shared IT resources through the Internet [1], which provides the respective resources supporting the deployment and execution of web services. A web service [2] is a programmable module with standard interface descriptions providing universal accessibility through standard communication protocols. A comprehensive web service usually requires a set of services to work together to accomplish the desired demands, which is well-known as service composition [3], [4]. Service composition allows developers to compose services into a business process workflow according to predefined requirements. For instance, a travelling service is usually composed of booking flight, booking hotel and renting car services. Nowadays, more and more service providers have started to offer candidate services that are functionally similar but differ in non-functional properties. Consider the payment service for booking flight. Both credit card service and Visa Electron service offer payment services. However, they charge different fees [5]. Non-functional properties are usually represented by Quality-of-Services (QoS). Service composition has become QoS-aware [6], which targets at finding the composition plan with the optimal end-to-end QoS.

As the usual requirements of QoS involve many traditional and widely-used QoS attributes, QoS-aware service composition has to be formulated as a multi-objective optimization problem. To tackle the tradeoff between different QoS attributes, some existing approaches transformed the problem into single-objective optimization by a utility function. Then, the problem can be formulated as a traditional optimization problem, and solved by some well-studied techniques [7]. The optimal solution is defined as the one with best utility value. However, another kind of approach based on Pareto set model has been investigated recently to solve QoS-aware service composition [8], [9]. It searches for a set of Pareto optimal composition solutions instead of a single solution, representing different tradeoffs between different QoS attributes.

Some previous approaches assumed the QoS values offered by a service were independent of other services [10], [11]. However, in many cases, the QoS of a service is correlated to others. For instance, Microsoft sales promotion claims to give a discount on the execution cost if two or more services are selected together in the same workflow [12]. Besides, response time of a composite service would be reduced if two selected services are deployed on the same provider, since the data transmission is much faster. For another example, if a customer just invokes a service from Amazon, then, other services from Amazon will become more accessible.
to him. Thus, QoS dependency need to be considered in service composition.

In this article, we study multi-objective service composition with QoS dependency involved which is largely unexplored by literature. Based on QoS values, we define dominance relationship between different workflows and dominance relationship between different candidate services. Taking advantage of the Pareto set model, we search for the set of Pareto optimal composition solutions. To achieve this, we first propose a candidate pruning algorithm, which keeps all the correlated and promising candidate services. It removes the candidates which can not be part of the optimal composite service. In this way, the search space can be reduced. We theoretically prove that the candidate pruning algorithm will not affect finding the optimal composition solutions. To further improve efficiency, we define the layered Pareto optimal composition solutions and the good enough composition solutions to soften the optimization goals. We propose tree-based Pareto Layers algorithms to lay the solutions, which introduce a natural order for the solutions in search space under multi-objective scenarios. Then, we extend the Vector Ordinal Optimization techniques [13] which are useful for dealing with multi-objective and complex problems. We propose a composition algorithm called Dependency-Aware Service Composition (DASC). The DASC algorithm makes use of QoS dependency information to find sub-optimal composition solutions efficiently. Furthermore, we conduct simulation experiments to verify the effectiveness of our DASC algorithm.

There have been some existing approaches studying dependency-aware service composition [5], [12], [14], [15]. However, multi-objective composition with QoS dependency to search for a set of Pareto optimal solutions is largely unexplored. Our approach fills this gap. Furthermore, to the best of our current knowledge, we are the first to take advantage of Vector Ordinal Optimization techniques to solve dependency-aware service composition, which explores a way to solve service composition problem. Our previous work [16] also studied QoS-aware service composition. The main differences from our previous article are listed as follows.

1) In our previous publication, we did not consider QoS dependency. In this article, we integrate QoS dependency to make our approach more practical and general.

2) In this article, we combine Pareto-based techniques with Vector Ordinal Optimization (VOO) techniques to search for Pareto optimal solutions. While in our previous publication, we did not take advantage of VOO techniques.

3) We propose the dependency-aware service composition algorithm in this article, including a candidate pruning algorithm and tree-based algorithms to obtain Pareto layers. While in our previous publication, only a composition algorithm based on partial selection was proposed.

The remainder of this paper is organised as follows. In Section 2, we introduce QoS dependency and define the Pareto optimal composition solutions. In Section 3, we propose the candidate pruning algorithm prior to service composition, then, we present the DASC algorithm for dependency-aware service composition. In Section 4, we carry out simulation experiments and discuss the simulation results. We analyze related work in Section 5. Section 6 concludes the paper and supplies future work directions.

2 MODEL OF SERVICE COMPOSITION WITH QoS DEPENDENCY

In this section, we first introduce the motivation scenario in Sec. 2.1. Section 2.2 introduces QoS attributes and dependency in service composition. Section 2.3 defines the objective of our service composition problem which is to search for the Pareto optimal solutions. Section 2.4 illustrates the difficulty in solving the multi-objective service composition problem with dependency involved, and introduces how to obtain the desired solutions efficiently. The latter will be discussed in detail in Sec. 3.

2.1 Motivation Scenario

We take advantage of the example in [5] and use it as our motivation scenario. Consider a user planning a holiday service consisting of 3 component services: booking flights, booking hotel, and booking sightseeing. There are two QoS attributes, cost and reliability. In total three companies of A, B and C are involved. The default QoS values for the candidate services are shown in Table 1. Suppose company A gives a discount if the candidates of arlnA and htlA are selected together, and their cost sum would become 580$. Company C also gives a discount if the candidates of htlB and sigC are both selected, due to some business cooperation between company B and C. And the cost sum for htlB and sigC is 500 $.

If the dependency information is not considered, the optimal composition solutions should be (arlnA, htlB, sigA) with QoS values of (650$, 83.03%), (arlA, htlB, sigB) with QoS values of (670$, 85.74%), and (arlA, htlB, sigC) with QoS values of (690$, 87.54%). However, when dependency information is considered, (arlA, htlA, sigA) with QoS values of (630$, 83.03%) would be one optimal solution, since its cost is the lowest. Thus, taking into account QoS dependency is important to find truly desirable solutions.

2.2 QoS Attributes and QoS Dependency

Suppose a composition request with m tasks is represented by a set \( I = \{1, 2, \ldots, m\} \). For each task \( i \), there are \( n_i \) candidate services able to accomplish it, denoted by a set \( C_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\} \). The candidates can be found by service discovery [17], [18]. Let \( S_p = (c_{1j}, c_{2j}, \ldots, c_{mj}) \) denote a concrete workflow which fulfills the request, where each \( c_{ij} \in C_i \) represents the candidate service selected to finish task \( i \). The search
TABLE 1
Default QoS Values

<table>
<thead>
<tr>
<th>service</th>
<th>cost($)</th>
<th>relia</th>
</tr>
</thead>
<tbody>
<tr>
<td>arlnA</td>
<td>150</td>
<td>95%</td>
</tr>
<tr>
<td>sigA</td>
<td>50</td>
<td>95%</td>
</tr>
<tr>
<td>hilB</td>
<td>450</td>
<td>95%</td>
</tr>
</tbody>
</table>

TABLE 2
Notations and Definitions.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Number of tasks.</td>
</tr>
<tr>
<td>n_i</td>
<td>Number of candidates of task i.</td>
</tr>
<tr>
<td>l</td>
<td>Number of QoS attributes.</td>
</tr>
<tr>
<td>l_i</td>
<td>Tasks set.</td>
</tr>
<tr>
<td>C_p</td>
<td>Candidates set of task i.</td>
</tr>
<tr>
<td>Dep_p</td>
<td>Dependency set of the r-th attribute.</td>
</tr>
<tr>
<td>c_{ij}</td>
<td>The p-th attribute of solution sub.</td>
</tr>
<tr>
<td>S_p</td>
<td>Original search space.</td>
</tr>
<tr>
<td>A</td>
<td>QoS attributes vector.</td>
</tr>
<tr>
<td>a_r</td>
<td>The r-th QoS attribute.</td>
</tr>
<tr>
<td>E_r</td>
<td>set of sub-compositions with at least two tasks (including two).</td>
</tr>
<tr>
<td>b_{r}(c_{ij})</td>
<td>Variable denoting whether the r-th attribute of candidate c_{ij} is correlated with others.</td>
</tr>
<tr>
<td>d_{r}(c_{ij})</td>
<td>Default value of the r-th attribute of candidate c_{ij}.</td>
</tr>
<tr>
<td>v_r(S_p)</td>
<td>Value of the r-th attribute of solution S_p.</td>
</tr>
<tr>
<td>S_p^{sub}</td>
<td>The sub-composition with dependency involved.</td>
</tr>
<tr>
<td>v^{sub}</td>
<td>The attribute value of the sub-composition S_p^{sub}.</td>
</tr>
<tr>
<td>V(S_p)</td>
<td>Value vector of solution S_p.</td>
</tr>
<tr>
<td>I_r(c_{ij})</td>
<td>Indicator variable denoting whether the candidate c_{ij} is correlated with other candidates.</td>
</tr>
<tr>
<td>Par(S)</td>
<td>The set of Pareto optimal solutions.</td>
</tr>
</tbody>
</table>

space S of all possible service compositions is the cartesian product defined by S = C_1 x C_2 x ... x C_m. The main notations in Sec. 2 are listed in Table 2. We only consider service composition approach for sequential structure here, and composition approach for other structures will be studied in our future work.

QoS which describes non-functional properties of services is important in service composition. And multiple criteria of QoS are usually considered together. Let a_r denote the r-th QoS attribute and A = (a_1, a_2, ..., a_l) be the vector of the QoS attributes in the paper. We present the formal definitions of QoS attribute, QoS dependency and QoS as below.

**Definition 1:** (QoS attribute) QoS attribute a_r (1 ≤ r ≤ l) represents the r-th non-functional property of services. In this article, we generalize the concept of QoS attribute to include not only the classical property such as response time, throughput, but also other properties like cost [5], [12].

**Definition 2:** (QoS dependency) Dep_p = \{S_p^{sub}, v^{sub}\} represents the set of dependencies of the r-th QoS attribute. Each element in Dep_p is a tuple (S_p^{sub}, v^{sub}), where S_p^{sub} denotes the sub-composition with dependency involved. Let E_r represent the set of sub-compositions having at least two tasks (including two tasks). E_r is the range of S_p^{sub}, v^{sub} defines the attribute value of the sub-composition S_p^{sub}.

**Definition 3:** (QoS) QoS is defined by \{(d_r, Dep_r)\}_{r=1}^{l}. Each element is a tuple (d_r, Dep_r), where d_r stands for the default value of the r-th attribute of the candidate service with no dependency involved. And Dep_r denotes the dependency information which has been defined in Definition 2.

The QoS of a concrete workflow S_p depends on the QoS of each selected candidate service c_{ij}. When there is no QoS dependency among these candidate services, the QoS of a composition solution S_p can be simply calculated by the corresponding aggregate functions. Some typical examples of aggregation functions for attributes in sequential structure are shown in Table 3 [19]. When there exists QoS dependency, it affects the performance of the composite service and cannot be neglected. QoS dependency exists in at least two services (including two services). In this article, we consider the general case where QoS dependency can exist in two services and among more than two services.

Let b_r(c_{ij}) = 1 represent the r-th attribute of candidate c_{ij} has dependency on other services, otherwise b_r(c_{ij}) = 0. d_r(c_{ij}) represents the default value of the r-th attribute of candidate c_{ij}. For instance, d_1(arnlnA) = 150$ in Table 1. The decision variables of the optimization problem are what candidate services to select for each task. S_p and c_{ij} (1 ≤ i ≤ m, 1 ≤ j ≤ n_i) represent the decision variables of the optimization problem. The other symbols are the constants of the problem.

2.3 Optimal Service Composition

Let the QoS values of the composite service S_p be denoted by

\[ V(S_p) = (v_1(S_p), ..., v_r(S_p), ..., v_l(S_p)), \]

where v_r(S_p) represents the value of the r-th attribute of S_p. Different attributes may have different optimization directions. For the attributes to be maximized, we multiply the attribute values with -1. The objective of our optimization problem can be expressed as

\[ \text{minimize } V(S_p) \]

Then, we present the definition of QoS-aware service composition problem in Definition 4.

**Definition 4:** QoS-aware service composition problem is to select the candidate service for each task to form composition solutions, so that the overall QoS values for the composite service are optimized.

Such compositions are called the optimal solutions. This article focuses on selecting the composition solutions based on obtained QoS information. The detailed QoS discovery or matching is not the focus of this current article. To tackle the tradeoff between different objectives, we take advantage of Pareto set model to search for...
TABLE 3
Examples of aggregation functions in sequential workflow.

<table>
<thead>
<tr>
<th>QoS attribute</th>
<th>response time</th>
<th>cost</th>
<th>availability</th>
<th>reliability</th>
<th>throughput</th>
<th>reputation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation function</td>
<td>$\sum_{i=1}^{m} d(c_{ij})$</td>
<td>$\sum_{i=1}^{m} d(c_{ij})$</td>
<td>$\prod_{i=1}^{m} d(c_{ij})$</td>
<td>$\prod_{i=1}^{m} d(c_{ij})$</td>
<td>$\min_{i=1}^{m} d(c_{ij})$</td>
<td>$\frac{1}{m} \sum_{i=1}^{m} d(c_{ij})$</td>
</tr>
</tbody>
</table>

The fine-grained model can guarantee to have the optimal solutions; however, it suffers from large time complexity. For a composite service with $m$ tasks, there can be $\binom{m}{2} + \frac{m}{3} + \ldots + \frac{m}{m} = 2^m - m - 1$ ways to combine them in the worst case. Thus, we focus on obtaining the desired solutions efficiently, which will be discussed in Section 3.

3 APPROACH FOR COMPOSITION WITH QoS DEPENDENCY

3.1 Architecture of the Composition Approach

Our proposed approach consists of two parts, which are preprocessing of candidates and service composition with QoS dependency. Preprocessing of candidates is conducted prior to composition, and it will be introduced in Sec. 3.2. Service composition with QoS dependency is conducted after the preprocessing of candidates finishes, and it will be introduced in Sec. 3.3. More specifically, composition with QoS dependency consists of two parts, i.e., softening goals and Vector Ordinal Optimization based composition.

3.2 Preprocessing of Candidate Service Sets

The composition space grows exponentially with the number of candidates in each task set. A QoS decomposition approach can be used to deal with the problem, and some evolutionary algorithms can also be used [20], [21]. In our article, we reduce the size of each task set by pruning uninteresting candidates. Removing one candidate from the first task set can reduce the search space by $\prod_{i=2}^{m} n_i$. However, the pruning process is much more complex when QoS dependency is introduced. Nevertheless, for those candidates that are free of correlations, we can still prune the unpromising ones among them. Let $I_i(c_{ij}) = 1$ represent that the candidate $c_{ij}$ is correlated with other candidates, otherwise $I_i(c_{ij}) = 0$. $I_i(c_{ij})$ can be calculated by the logical OR of $b_i(c_{ij})$, i.e., $I_i(c_{ij}) = b_1(c_{ij}) \lor b_2(c_{ij}) \lor \ldots \lor b_i(c_{ij}) \lor \ldots \lor b_l(c_{ij})$. Alternatively, we could take the sum and check if it is greater than zero. Recall that $b_i(c_{ij}) = 1$ represents the $r$-th attribute of candidate $c_{ij}$ is correlated to other candidates.

For each task $i \in I$, let $\tilde{C}_i = \{c_{ij} | c_{ij} \in C_i \land I_i(c_{ij}) = 0\}$ be the set of candidates with no dependency on others. Consider two candidates $c_{ij}, c'_{ij} \in \tilde{C}_i$. Since a candidate can be viewed as a special workflow with only one service, the dominance relationship between $c_{ij}$ and $c'_{ij}$ is similar to that of composition solutions. We use the expression $c_{ij} \succ c'_{ij}$ to represent that $c_{ij}$ dominates $c'_{ij}$.
The candidates in $\tilde{C}_i$ that are not dominated by others are promising ones called optimal candidates.

The dominated candidates can be eliminated to reduce search space and improve efficiency. Pairwise comparisons among the candidates in $\tilde{C}_i$ are required to obtain the optimal candidates and eliminate the dominated ones. The enumeration order of candidates can affect the elimination efficiency. Choosing an optimal candidate to be the first enumerated one can improve efficiency. We define the grade of candidate $c_{ij} \in \tilde{C}_i$ as $g(c_{ij}) = \sum_{r=1}^{m} d_r(c_{ij})$. It is easy to see that the candidate with the smallest grade value must be an optimal candidate, i.e., if $c_{ij} \in \tilde{C}_i$ and $g(c_{ij}) = \min_{c_{ij} \in \tilde{C}_i} g(c_{ij})$ hold, then $c_{ij}$ is an optimal candidate in $\tilde{C}_i$.

We let the optimal candidate $c_{ij}$ be the first one to be enumerated in $\tilde{C}_i$. For simplicity, the enumeration process based on the $k$-th candidate is called the $k$-th round, and let $c_k$ represent the $k$-th candidate in $\tilde{C}_i$. The detailed candidate pruning algorithm for each set $\tilde{C}_i$ is shown in Algorithm 1. $n_i = |\tilde{C}_i|$ is the number of initial candidates, and $\tilde{C}_i$-opt represents the set of optimal candidates. The variable $\text{IsDominated}(k)$ denotes whether $c_k$ has been dominated. In the $k$-th round, $c_k$ is pairwise compared with the remaining non-dominated $c_j$ where $k < j \leq n_i$. For $c_j$ that has been dominated already ($\text{IsDominated}(j) = \text{True}$), the pairwise comparisons between $c_k$ and $c_j$ are omitted. Once $c_k$ is dominated by $c_j$, we mark $\text{IsDominated}(j)$ with $\text{True}$ and the process of pairwise comparing $c_k$ with remaining $c_t$ ($j < t \leq n_i$) is omitted. Then the $(k + 1)$-th round begins and the pairwise comparison processes continue until the last round finishes.

Note that instead of exhaustive pairwise comparison of all the candidates, we omit some pairwise comparison operations in Algorithm 1. By this way, we can not only reduce execution time but also guarantee the optimality. Theorem 1 demonstrates that Algorithm 1 can still guarantee to find all the optimal candidates.

**THEOREM 1:** Algorithm 1 can guarantee to obtain all the optimal candidates in $\tilde{C}_i$.

The detailed proof for Theorem 1 has been presented in Appendix A. Algorithm 1 prunes the unpromising candidates and keeps the optimal ones. Let $C_i'$ represent the set of kept candidates, expressed in (6).

$$C_i' = C_i - (\tilde{C}_i - \tilde{C}_i\text{-opt}).$$

$C_i'$ contains all the correlated candidates and the optimal ones of non-correlated candidates. The relationship between sets $C_i$, $\tilde{C}_i$, $\tilde{C}_i\text{-opt}$, and $C_i'$ is illuminated by Fig. 1. Due to space limitation, the analysis for search space reduction is presented in Appendix B.

We will show that service composition based on set $C_i'$ of each task $i$ can obtain the optimal composition solutions, as presented in Theorem 2.

**THEOREM 2:** The optimal composition solutions can be obtained by keeping only set $C_i'$ for each task $i$.

Interested readers can refer to Appendix C for detailed proof for Theorem 2.

### 3.3 Dependency-aware Service Composition

#### 3.3.1 Goal Softening

Let $S' = C_1' \times C_2' \times \ldots \times C_m'$ be the new composition space after candidates pruning. The space size $|S'|$ grows exponentially with the number of tasks, and can be still large after candidates pruning. Thus, we soften the goals by desiring solutions that are approximate to Pareto optimal composition solutions. To achieve this, we discuss how to sort or order the solutions for multi-objective service composition. We will introduce *layers* to sort the solutions. Let $S_1 - S_2$ represent the set containing...
the elements in set $S_1$ but not in set $S_2$. We introduce layers [22] and present the definition of layered pareto optimal solutions for service composition, as shown in Definition 7.

**Definition 7:** (Layered Pareto optimal composition solutions) For a given solution space $S'$, let $L_1 = Par(S')$ represent the set of first layer (Layer 1) Pareto optimal composition solutions. Remind that $Par(S')$ denotes the truly Pareto optimal solutions of set $S'$, according to (5). Then the $e$-th layer (Layer $e$) Pareto optimal composition solutions are defined by (7), which are the Pareto optimal composition solutions after all the previous layers ($L_1, L_2, \ldots, L_{e-1}$) have been removed.

$$L_e = Par(S' \cup \bigcup_{h=1}^{e-1} L_h)$$

(7)

**Insight:** The introduction of layered Pareto optimal composition solutions brings a natural order of the solutions in search space $S'$. To be more specific, let us first consider single-objective situation. For single-objective optimization problem with $S'$ representing the search space, let $S_{p1} \in S_1$ represent the best solution optimizing the single objective. Then, we remove $S_{p1}$ from the search space $S'$, and the new search space becomes $S' - \{S_{p1}\}$. Let $S_{p2} \in S'' - \{S_{p1}\}$ represent the best solution in new search space $S'' - \{S_{p1}\}$. Thus, $S_{p2}$ is the second best solution in original search space $S'$. Then, we can recursively define the third best, fourth best solutions in original search space $S'$.

Similarly, now let us consider the multi-objective optimization situation with search space represented by $S'$. The exactly best solutions (may have more than one) are the Pareto optimal solutions of $S'$. We use $L_1$ to represent the first layer. Then, we remove $L_1$ from $S'$, and we get the new search space $S'' - L_1$. Let $L_2$ represent the set of Pareto optimal solutions of $S'' - L_1$, and we call $L_2$ the second layer. We can define the third layer $L_3$, the fourth layer $L_4$ recursively. In this way, we can bring a natural order for solutions under multi-objective optimization problem. For the solutions in different layers, the ones in front layers are superior. However, for the solutions in the same layer, they cannot dominate each other.

Suppose there are $q$ layers in total. We soften the optimization goals and focus on layered Pareto optimal solutions, which are much easier to obtain. For example, if we randomly pick a composition solution $S_p \in S'$, the probability that $S_p$ belongs to the first three layers is larger than the probability that $S_p$ is Pareto optimal. Moreover, it may be required to obtain more than one good composition solutions. To better characterize this kind of requirement, we define the good enough compositions set as

$$G = \bigcup_{e=1}^{q} L_e$$

(8)

which is the union of the composition solutions in the first $g(1 \leq g \leq q)$ layers. The solutions in $G$ are called the good enough composition solutions, also called the desirable solutions. We define selected compositions set $SC$ as the set of composition solutions chosen based on some certain mechanism (such as random selection, heuristics selection). Providers or users are free to decide $G$ and $k$ based on their requirement.

If $G \subseteq SC$, i.e., all the desirable solutions are covered in the selected set, then, this would be an ideal case. However, this is impractical since it requires to enumerate all the possible composition solutions and check global dependency information for each composition. Instead, we aim to use a coarse-grained model to select set $SC$ to achieve acceptable matching results with high efficiency. Let $P_A$ represent the alignment probability [13] between sets $G$ and $SC$, i.e., the probability that at least $k_1$ compositions in $SC$ are good enough solutions. $P_A$ is defined by

$$P_A = \text{Prob}(G \cap SC \geq k_1)$$

(9)

where $k_1$ represents the alignment level. Both $G$ and $k_1$ are predefined.

In this article, we focus on how to select $SC$ from $S'$ such that we can maintain a satisfactory match between $SC$ and $G$ with high probability, i.e., $P_A \geq \alpha$. $\alpha$ represents the desired alignment probability. Choosing $SC$ with a larger size can help increase the alignment probability; however, this would also increase execution time. Here, we consider a simple mechanism which chooses $SC$ randomly from $S'$, providing guidance on how large enough $|SC|$ should be to achieve satisfactory results. The closed form expression of the alignment probability $P_A$ under random selection mechanism is shown in (10).

$$P_A = \text{Prob}(G \cap SC \geq k_1) = \min \left( \sum_{h=1}^{k} |L_h|/|SC| \right)$$

(10)

The detailed proof for (10) has been presented in Appendix D. Thus, under certain alignment probability requirement $\alpha$, given good enough solutions size $|G|$ and alignment level $k_1$, the size $|SC|$ under random selection mechanism can be calculated. Then, search of optimal composition solutions can be done in set $SC$ instead of $S'$, improving efficiency greatly. Note that the size $|SC|$ under random selection mechanism actually provides an upper bound of selection set size, since none of the knowledge of QoS attributes and service composition is made use of. Thus, we expect higher alignment probability and smaller selected set size when QoS information is taken advantage of, which will be discussed in the following parts.

3.3.2 Background of Vector Ordinal Optimization

We take advantage of Vector Ordinal Optimization (VOO) techniques [22] to solve the problem. Here, we present a short introduction of VOO. VOO techniques are used for multi-objective optimization problem. The
goal is to find a set of good enough solutions that are Pareto optimal or nearly Pareto optimal. Some key concepts or definitions in VOO are listed as below.

(1) Dominance Relationship. It determines the superior relationship of one solution over the other solutions.

(2) Pareto Layers. The current Pareto layer includes the successive Pareto optimal solutions after the previous layers have been removed from consideration. The Pareto layers introduce a natural order for solutions in search space.

(3) Good Enough Set. The true first $g$ layers of solutions. The set is denoted by $G$. When $g = 1$, $G$ is the set of Pareto optimal solutions.

(4) Selected Set. The selected $s$ layers of solutions based on their estimated performance. The set is denoted by $SC$.

(5) Vector Ordered Performance Curve (VOPC). It is a concept to classify the problem type, i.e., the problem is relatively easy to solve, or hard to solve, or just normal.

(6) Error Level. It is used for defining the difference between the estimated performance and the true performance. When the error is small, the truly good solutions are easier to obtain.

Then, the VOO takes the following steps to obtain $k_1$ good enough solutions.

Step 1. Use a crude and computationally fast model to estimate the performance criteria of the solutions.

Step 2. Estimate the VOPC type and error level.

Step 3. The user specifies the size of good enough set $G$, and the required alignment level $k_1$.


Step 5. Select the estimated first $s$ layers as the selected set.

Step 6: The theory of VOO ensures at least $k_1$ truly good enough solutions are obtained with probability no less than 0.95.

3.3.3 Vector Ordinal Optimization based Service Composition

In this section, we propose the DASC algorithm which makes use of QoS information to pick the set $SC$ efficiently, and obtain satisfactory composition solutions.

(1) Crude Model. We use a crude but efficient model to reduce time and space significantly. We pick a set of representative composition solutions from search space $S'$. Each solution $S_p \in S'$ is chosen with equal probability. According to statistical facts, representative composition solutions from the space can be obtained [23]. We are concerned about deciding the number of representative compositions $M \ll |S'|$ so as to cover some good enough solutions from the search space. Let $E_3$ denote the event that at least one of the $M$ compositions is in the top $\xi$-percentile of the space [24]. Thus, it can be obtained that $\text{Prob}(E_3) \approx 1 - (1 - \xi^%)^M$. For example, when $\xi = 0.1, M = 10000$, we have $\text{Prob}(E_3) \approx 1 - (1 - 0.001)^{10000} \approx 1 - 4.5173 \times 10^{-5}$. The probability that the representative compositions set contains good solutions is almost 1 in this case. This is our first procedure of softening optimization goals. In this way, the search space can be reduced by several orders of magnitudes [13], improving efficiency significantly.

Let $S$ denote the representative compositions set. To save execution time, we take advantage of a coarse-grained but computationally easy model to estimate the performance of solutions in $S$. The model ignores dependency information among services, and simply uses the aggregate functions to calculate the QoS values of each composition solution. For brevity, we call this model the coarse-grained model.

For each composition solution $S_p \in S$, let $V_{cc}(S_p)$ represent the estimated QoS of $S_p$ under our coarse-grain model, denoted by

$$V_{cc}(S_p) = (v_{1cc}(S_p), ..., v_{rcc}(S_p), ..., v_{lcc}(S_p)), \quad (11)$$

where $v_{rcc}(S_p)$ denotes the $r$-th attribute value under the coarse-grain model.

(2) Pareto-optimal Composition Layers

Based on the QoS values $V_{cc}(S_p)$ of each solution, we divide the $M$ compositions into Pareto-optimal composition layers. To make full use of dominance relationship, we propose Algorithm 3 which finds Pareto layers based on a tree structure. We build the tree based on the dominance relationship between each pair of composition solutions. For each composition solution $S_p \in S$, two arrays which are $S_p$ and $S_j$ are maintained. $S_p$ contains the solutions that dominate $S_j$, while $S_j$ includes the solutions that are dominated by $S_p$. For brevity, let $S_p$ represent the $i$-th composition solution in $S$. For any two composition solutions $S_p$, $S_j \in S$, if $S_p \succ S_j$, then we add a directed edge from $S_p$ to $S_j$, and add $S_p$ to $S_j$. Pairwise comparison is conducted only once in this process. The detailed procedure about the pairwise comparison is shown in Algorithm 2.

We search for the Pareto-optimal layers based on the constructed tree. At the very beginning, the composition solutions $S_p$ with empty $S_j$ are the first layer Pareto-optimal compositions. Besides, for each current Pareto-optimal composition $S_p$, we find the solutions $S_{p_k}$ in $S_p$. $S_{p_k}$ is dominated by $S_p$. We delete $S_{p_k}$ from $S_p$, since $S_{p_k}$ will be removed from the remaining set. Then this process continues and we search for Pareto-optimal compositions in the next layer. The searching procedure ends when the remaining set is empty.

(3) Selecting Composition Solutions

Based on VOO techniques, $SC = \cup_{i=1}^{s} L_e$ is chosen as the selected set, i.e., the union of the estimated first $s$ layers. Here, $s$ will be determined by the DASC algorithm. For practicality we set the required alignment probability as 0.95, which is a certain high value.

We can make use of $L_e$ to have some knowledge of the problem type, i.e., whether the good enough composition solutions are easy to obtain. Let $F(x)$ represent
Algorithm 2 Tree Building Algorithm

TREEROPE-BUILDING(S)
Input: Set S
Output: \( \hat{S}_p, \tilde{S}_p \) for each \( S_p \in S \)
1: for \( i = 1 \) to \( M \) do
2: \( \hat{S}_p = \emptyset \)
3: \( \tilde{S}_p = \emptyset \)
4: for \( i = 1 \) to \( M \) do
5: for \( j = i + 1 \) to \( M \) do
6: Compare \( S_{p_i} \) and \( S_{p_j} \) based on \( V_{cc}(S_{p_i}) \) and \( V_{cc}(S_{p_j}) \)
7: if \( S_{p_i} \succ S_{p_j} \) then
8: \( \hat{S}_p, \tilde{S}_p \rightarrow \emptyset \)
9: \( \hat{S}_p, \tilde{S}_p \rightarrow \emptyset \)
10: else if \( S_{p_j} \succ S_{p_i} \) then
11: \( \hat{S}_p, \tilde{S}_p \rightarrow \emptyset \)
12: \( \hat{S}_p, \tilde{S}_p \rightarrow \emptyset \)
13: return \( \hat{S}_p, \tilde{S}_p \)

Algorithm 3 Pareto-optimal Composition Layers

Input: Set S
Output: Pareto-optimal Composition Layers \( L_i \)
1: Tree-Building(S)
2: RemainingSet = S
3: \( i = 0 \)
4: while RemainingSet \( \neq \emptyset \) do
5: \( i = i + 1 \)
6: \( L_i = \emptyset \)
7: for \( j = 1 \) to |RemainingSet| do
8: if \( \hat{S}_{p_j} = \emptyset \) then
9: \( L_i, \tilde{S}_{p_j} \rightarrow \emptyset \)
10: for all \( S_{p_k} \in \hat{S}_{p_j} \) do
11: \( \tilde{S}_{p_k}, \tilde{S}_{p_j} \rightarrow \emptyset \)
12: RemainingSet.delete(S_{p_j})
13: \( q = i \)
14: return All layers \( L_i(1 \leq i \leq q) \)

Fig. 2. Typical types of VOPCs.

Error level is another factor influencing \( s \) determination. It defines the deviation between estimated QoS values and true QoS values. The estimated QoS value means the QoS value without considering dependency information. The true QoS value means the QoS value with dependency considered. In this way, we have a rough idea of how large the noise is, i.e., how great the dependency’s influence is. Under a larger error level, we need to select a larger \( s \) so as to cover the good solutions, while \( s \) can be much smaller when the error level is smaller. To estimate the error level under our coarse-grain model, we randomly select \( M' \ll M \) solutions in \( S \) and obtain the true QoS values by the fine-grained model. Then we scale the QoS values by normalization, and calculate the standard deviation of the \( M' \) solutions. The largest one among the \( l \) attribute deviations is regarded as the error level. Let \( \text{diff}_r = \frac{|v_r(S_p) - v_r(S_t)|}{\max \{v_r(S_p)\}} \) represents the normalized difference value for the \( r \)-th attribute. The error can be calculated by

\[
\max \left\{ \frac{\sum_{S_p}(\text{diff}_r)^2}{M} \right\}, 1 \leq r \leq l.
\]  

If the error level \( s \) is small, then it is considered as a small one, and \( 0.5 < \text{error level} \leq 1 \) represents a medium one, while \( 1 < \text{error level} \) denotes a large one [24].

When VOPC and error level have been obtained, the number \( s \) of estimated layers to select can be calculated based on VOO techniques. Providers or users are free to decide \( g, \alpha \) is usually set as 0.95 in VOO techniques. In [22], regression analysis is conducted to derive the coefficient parameters based on 10,000 solutions with 100 layers and two objectives. It is demonstrated that the result is upper bound for the selection set size when more objectives are considered. We borrow such idea and calculate \( s \) as (14)-(17), where \( \gamma, \delta, \eta \) and \( \theta \) are the corresponding coefficients defined in [22] which are obtained through regression analysis. Eq. (14), (15) and (17) are used for mapping our parameters to the parameters in [22] to keep \( g' = g/g, k'/1000 = k_1/M, \) and \( s/g = s'/100 \).

\[
g' = \max \left\{ 1, \frac{100}{q} \cdot g \right\}, \quad k' = \max \left\{ 1, \frac{10000}{M} \cdot k_1 \right\}, \quad s'(k', g') = c^r(k')^\delta(g')^\gamma + \theta, \quad s = \left[ \frac{q}{100} \cdot s' \right].
\]
We choose $SC = \sum_{g=1}^{s} L_g$ as the selected compositions set. The theory of VOO guarantees that $SC$ includes at least $k_1$ good enough composition solutions with probability to be no less than 0.95. Based on the selected set $SC$, we then use the fine-grained model to select the top $k_1$ good enough composition solutions. The main steps of our DASC algorithm are shown in Algorithm 4. The complexity analysis of Algorithm 4 is shown in Appendix E.

**Algorithm 4 Dependency-Aware Service Composition (DASC)**

**Input:** Representative composition set $S$, number of good enough solution layers $g$, alignment level $k_1$  
**Output:** $k_1$ good enough composition solutions

1. Calculate the QoS values of all the compositions in $S$ using coarse-grain model, i.e., ignore the dependency information
2. Estimate the VOPC type, i.e., flat, neutral or steep based on (12)
3. Estimate the normalized error level, i.e., small, medium or large based on (13)
4. Calculate the number $s$ of selected layers based on (14)-(17), and the theory of VOO ensures that the $s$ layers contains at least $k_1$ good enough compositions with probability no less than 0.95
5. Add dependency information to use the fine-grain model introduced in Sec 2.4 to calculate the exact QoS values of the composition solutions in the selected $s$ layers, and choose the top $k_1$ compositions
6. **return** $k_1$ good enough composition solutions

### 4 Evaluation

#### 4.1 Experiment Setup

We conduct simulation experiments to evaluate the effectiveness of our DASC algorithm. We focus on sequential workflows. We adopt the WSDream dataset which records the values of QoS attributes of response time and throughput of 5,825 real-world web services from 73 countries [25]. 5,797 services with both valid response time and throughput are chosen. These services are randomly classified into 4 service categories (tasks). The corresponding aggregate functions of the two attributes are additive and minimal, while the optimization directions of them are minimization and maximization, respectively. Let response time be the first attribute, and throughput be the second attribute. In our algorithm running process, we negate the throughput values to transform to minimization. Nevertheless, in the figures showing the final results, we transform the throughput values to positive numbers to make the figures clear.

We use the original data as the default QoS values of candidates. As there is little standard experimental platform or test data for the QoS dependency data, most dependency-aware service composition approaches used synthetic data for evaluation [5]. Therefore, in our experiment, QoS dependency is randomly generated. The percentage of services with correlations is set to be 5%. The ratio of the correlated throughput and default value is generated randomly from 0 to 1, and the ratio of the correlated response time and default value is generated randomly from 0 to 1.5 [12]. $g$ is set to be 2, and $k$ is set to be 10 [13], [26].

#### 4.2 Experiment Results

Fig. 3 shows the good enough solutions in the composition set, presenting readers an intuitive example of layered Pareto optimal composition solutions. The results are obtained by the fine-grained model and exhaustive search method. There are 29 composition solutions in total. 11 of them are the first layer Pareto optimal compositions and the rest belong to the second layer. The first layer solutions are truly Pareto optimal. Each composition solution in the second layer is dominated by at least one composition solution in the first layer. We use the coarse-grain model to estimate the QoS values of the composition solutions by ignoring the dependency information. Fig. 4 shows the VOPC type of the estimated composition solutions, which is neutral. Thus, the good enough solutions are neither easy nor hard to obtain. Then, we estimate the error level of the composition solutions. We randomly select 100 solutions and calculate the deviation between the estimated QoS values and the true QoS values. We obtain the normalized error as 0.252 according to (13), which corresponds to a small error level. After this, we calculate the number of selected layers $s$ and obtain the composition solutions to select. Fig. 5 shows the selected and matched solutions. A total of 1,028 composition solutions are selected, among which 18 solutions belong to the solutions shown in Fig. 3, i.e., the good enough solutions. 8 out of these 18 solutions are really Pareto optimal, and the rest 10 solutions are near-optimal. All of the 18 solutions are good enough. It can be seen that the number of matched composition solutions is larger than the required alignment level $k$, which validates our DASC algorithm.

Next, we validate the actual alignment probability against required $P_A = 0.95$, where $g = 2$ and $k = 4, 6, \ldots, 20$. For each setting, we carry out 1000 experiments. Table 4 shows the fractions of the 1000 experiments in which there are at least $k$ composition solutions matched. It can be seen that all of the alignment probabilities are greater than 0.95. Furthermore, let $|GSC|$ represent the covering rate of matched solutions versus good enough solutions, i.e., the percentage of good enough solutions that are covered in the selected solutions. We average the results of the 1000 tests. The covering rate for $g = 2, k = 10$ is 84%, and the covering rate for $g = 2, k = 16$ is 90%.

To evaluate the impact of alignment level $k$ on the number of selected solutions, we vary $k$ from 2 to 20. For each $k$, we conduct 1000 experiments, and then
average the results. We compare our DASC algorithm with random selection algorithm, where each composition solution is selected with the same probability. We calculate the number of selected solutions using random selection algorithm with requirements of $P_A = 0.95$ and $P_s = 0.5$. Fig. 6 shows the number of selected solutions with different $k$. It can be seen that the number of selected solutions under our DASC algorithm is even smaller than that of random selection algorithm with $P_A = 0.5$, which shows the efficiency of our DASC algorithm.

To further evaluate the DPSA approach, we adopt another dataset, i.e., the QWS dataset [27], which have a larger number of QoS attributes. We use the data from the QWS dataset as the default QoS values. The other settings are the same as in the WSDDream dataset. To evaluate the impact of workflow size on our DPSA algorithm, we vary the number of tasks $m$ from 4 to 12. For each setting, we conduct 1000 experiments. Table 5 validates the alignment probability against the requirement of 0.95. It can be seen that as the workflow size increases, the alignment probability would tend to drop slightly. Nevertheless, the requirement that the alignment probability should be greater than 0.95 is still guaranteed. Table 6 shows the execution times (in seconds) of our approach under different workflow sizes. We average the results of the 1000 experiments. We can see that as the workflow size rises, the execution time of our approach also increases.

We also compare with two approaches with respect to composition time and optimality. The first one is an exhaustive approach, which can definitely obtain all the Pareto optimal composition solutions. The second one is a heuristic approach extended from [28], which obtains near-optimal Pareto solutions. As there are no concepts of layered Pareto in the two compared approaches, we set $g = 1$ in our DPSA approach. That is to say, the truly Pareto optimal solutions are desirable. We use covering rate to evaluate optimality. Covering rate is defined as the percentage of obtained optimal solutions versus the total optimal solutions. For our DPSA approach, we test three settings of $k = 5, k = 10$ and $k = 20$. The original search space is varied from $10^4$ to $10^7$. The exhaustive search (ES) approach cannot output the optimal solutions within reasonable time if the space size is increased furthermore. For each setting, we conduct 1000 experiments, and average the results.

Fig. 7 shows the covering rates under different approaches. It can be seen that the covering rates of the exhaustive approach are always 1, since this approach can obtain all the Pareto optimal solutions. The covering rate of the DPSA approach rises with the increase of $k$. The covering rates of our DPSA approach are smaller than the heuristic approach if we set $k = 5$. However, if we set $k = 20$, the covering rates of the DPSA approach would be larger than the heuristic one. Fig. 8 shows the execution times of these approaches, respectively. When $k$ increases, the execution times of our approach would increase a little. Nevertheless, the execution times of our DPSA approach are much smaller than the other two approaches. Together with Fig. 8, it is shown that our DPSA approach can cover a large portion of Pareto optimal solutions at acceptable time costs.

The weak points of our approach are two-fold. The first is that it can not guarantee to obtain all the exactly Pareto optimal solutions. The second is that our approach is currently suitable for only sequential workflow. The best points of our approach are also two-fold. Firstly, our approach obtains several good enough Pareto optimal solutions with significantly reduced execution time.
how to handle the tradeoffs between different attributes which will be discussed in the following paragraphs of Sec. 5.1, and how to deal with QoS dependency which will be discussed in Sec. 5.2.

Due to the tradeoffs among different QoS attributes, it is generally hard to get the best QoS values for all the attributes [19]. For example, it is hard to select a composition solution that minimizes both response time and price. Because generally speaking, for intra-provider offerings, services with smaller response time usually cost more. To deal with multiple QoS attributes, some previous approaches transformed the multi-objective service composition problem into single-objective optimization problem. The methods can be classified into 2 categories. The first one is based on Simple Additive Weighting (SAW) [7], which scales the QoS attribute values by comparing with the maximum/minimum/average values, and then aggregates the multiple objectives to a single one by setting their weights. Then, a utility function is defined which is used as an optimization formula for the optimization problem such that the best solution will have the best solution for it. The SAW method is easy to apply. However, it simply sums up the weighted attributes regardless of their different units and ranges, reducing individual attribute value to an overall value; therefore, certain QoS information may be lost [26]. Furthermore, setting weights requires the knowledge of user preferences and priorities, and it is not easy for the user to transform the preferences into exact numeric metrics [9]. The second type of methods is ε-constraint [39], which selects only one attribute as the optimization objective. The other attributes are then expressed by constrains, reducing the problem to single-objective optimization. Then, the problem can be solved by existing approaches. The ε-constraint method is easy; however, there is no specific approach followed for the details of setting added constraints bounds.

Besides, another service composition method based on Pareto set model has been investigated recently. The basic idea is to find the set of Pareto optimal composition solutions instead of a single one [28]. Pareto optimal solutions represent the tradeoffs related to different attributes [19]. Pareto set model is widely used in multi-objective optimization and multi-criteria decision making [40]. It also provides several alternative composition solutions with the optimal QoS. Yu and Bouguettaya [33] presented a Bottom-Up algorithm to compute the Pareto optimal solutions. In our previous work [16], we demonstrated the general applicability of the Pareto set model, and showed that finding Pareto optimal composition solutions could act as preprocessing procedure for other composition methods. However, the services were assumed to be independent.

5.2 QoS Dependency in Service Composition

Different kinds of dependencies exist at the service and QoS levels. For service level, there are flow dependency,
compatibility dependency and statistical cooperate dependency [5], [15], [41]. Flow dependency means one service should be executed before another due to data logical dependency. Compatibility dependency has effect on whether two services can be selected together in a composition (e.g., these services have compatible interfaces) [42], [43]. Statistical cooperate dependency means that two or more services are usually banded in a composition service. For QoS level, time-dependent QoS has been studied in related literature [44], [45], [46]. It means that the QoS values of services evolve with time and are dependent on the execution time. Here in our article, we focus on QoS dependency that the QoS values of a service are not only dependent on itself, but also correlated to other services. To be more specific, when a service is selected together with its correlated services, their QoS values will change. We focus on finding the Pareto optimal solutions with QoS dependency involved.

Some existing approaches proposed evolutionary algorithms to solve QoS-dependency-aware service composition. Florian et al. [20] studied QoS dependency on the time of the execution or the input data. The authors proposed a genetic algorithm to obtain approximations of the Pareto optimal solutions set. We study different things. Our focus is QoS dependency that the QoS values of a service are correlated to other services, and when these services are selected together, their QoS values will change. Also, we propose a candidate pruning algorithm prior to selection. Tao et al. [21] used particle swarm optimization techniques to solve the correlation-aware resource service composition problem in manufacturing grid. They looked for approximations of Pareto optimal solutions. However, they assumed the dependency existed only in two services. Besides, they only conducted global optimization in service composition level, and the search space size was not reduced. Our approach further improves efficiency by a candidate pruning algorithm to reduce the search space before global optimization.

Some approaches used graph theory to study service composition with dependency involved. Feng et al. [14] presented a graph-based algorithm which dynamically refined the composed services with QoS dependency. However, only one QoS attribute was considered, and their approach returned all available composite services satisfying the topological constraints. As a result, it may suffer from efficiency issues. Barakat et al. [5] considered multiple attributes and used Bellman-Ford algorithm to solve the problem. There are three main differences between our approach and theirs. First, our objectives and goals are different. Their approach used SAW techniques to transform multiple objectives to a single one, and then used single-objective optimization to look for one solution. Our approach uses multi-objective optimization techniques, i.e., Pareto-based techniques to look for the set of Pareto optimal solutions. Second, they used graph theory to solve the optimization problem while we make use of Vector Ordinal Optimization techniques to search for solutions. Third, they assumed dependency only existed in two services.

Guo et al. [15] studied business entity dependency meaning that QoS values may change due to the cooperation between different service providers. There are several differences between our approach and their approach. First, our QoS-dependency-aware approach is more general that we include the case that QoS dependency also occurs in a single service provider. Second, they assumed the dependency only existed in two services. However, our approach can deal with dependency among even more services. Third, they used the SAW techniques to transform multi-attributes values to a single value and obtain one solution, while our approach takes advantage of Pareto-based techniques

### TABLE 7
Comparison of different approaches.

<table>
<thead>
<tr>
<th>Approach</th>
<th>QoS Dependency</th>
<th>Multi-attribute Optimization</th>
<th>Optimization Mode</th>
<th>optimality</th>
<th>Used Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuGQoS [32]</td>
<td>not supported</td>
<td>not supported</td>
<td>global</td>
<td>optimal</td>
<td>Breadth First Search</td>
</tr>
<tr>
<td>LOEM [11]</td>
<td>not supported</td>
<td>SAW</td>
<td>local+global</td>
<td>near-optimal</td>
<td>Enumeration</td>
</tr>
<tr>
<td>BUA [33]</td>
<td>not supported</td>
<td>Pareto based</td>
<td>local+global</td>
<td>optimal</td>
<td>Bottom-Up Algorithm</td>
</tr>
<tr>
<td>DIFSA [16]</td>
<td>not supported</td>
<td>Pareto based</td>
<td>local+global</td>
<td>optimal</td>
<td>Distributed Partial Selection</td>
</tr>
<tr>
<td>SL [34]</td>
<td>not supported</td>
<td>SAW</td>
<td>local+global</td>
<td>near-optimal</td>
<td>Mixed Integer Programming</td>
</tr>
<tr>
<td>QCWS [35]</td>
<td>not supported</td>
<td>SAW</td>
<td>global</td>
<td>optimal</td>
<td>Branch-and-Bound</td>
</tr>
<tr>
<td>NSFC [36]</td>
<td>not supported</td>
<td>SAW</td>
<td>local+global</td>
<td>optimal</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>Gao et al. [39]</td>
<td>not supported</td>
<td>SAW</td>
<td>local+global</td>
<td>optimal</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>MOEA [38]</td>
<td>not supported</td>
<td>Pareto based</td>
<td>global</td>
<td>approximated</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>Florian et al. [20]</td>
<td>on time</td>
<td>Pareto based</td>
<td>global</td>
<td>approximated</td>
<td>Particle Swarm Optimization Algorithm</td>
</tr>
<tr>
<td>Tao et al. [21]</td>
<td>QoS correlation</td>
<td>Pareto based</td>
<td>global</td>
<td>approximated</td>
<td>Backward, Breadth-First Graph search</td>
</tr>
<tr>
<td>Feng et al. [14]</td>
<td>QoS correlation</td>
<td>not supported</td>
<td>global</td>
<td>optimal</td>
<td>Bellman-Ford Algorithm</td>
</tr>
<tr>
<td>Barakat et al. [5]</td>
<td>QoS correlation</td>
<td>SAW</td>
<td>local+global</td>
<td>optimal</td>
<td>Exhaustive search</td>
</tr>
<tr>
<td>Guo et al. [15]</td>
<td>QoS correlation</td>
<td>SAW</td>
<td>global</td>
<td>optimal</td>
<td>Greedy Algorithm</td>
</tr>
<tr>
<td>CASP [12]</td>
<td>QoS correlation</td>
<td>not supported</td>
<td>local+global</td>
<td>optimal</td>
<td>Vector Ordinal Optimization</td>
</tr>
<tr>
<td>DASC (ours)</td>
<td>QoS correlation</td>
<td>Pareto based</td>
<td>local+global</td>
<td>near-optimal</td>
<td>Vector Ordinal Optimization</td>
</tr>
</tbody>
</table>
to search for multiple Pareto optimal solutions. Fourth, they did not prune the unpromising candidates prior to selection while we propose a candidate pruning algorithm to reduce search space. Deng et al. [12] proposed two algorithms to solve the service composition problem with dependency involved. One algorithm was for QoS dependency in adjacent services, and the other was for dependency in nonadjacent services. The main differences between our approach and their approach is two-fold. First, they assumed the dependency only existed in two services while our approach is more general to handle dependency among more services. Second, they only considered one QoS attribute, making the problem single-objective optimization problem. However, our approach focuses on multi-objective optimization problem and our result is a set of desirable solutions. To the best of our knowledge, we are the first to take advantage of Vector Ordinal Optimization techniques to deal with multi-objective service composition. Besides, we combine candidate pruning to conduct local optimization which reduces search space size.

6 CONCLUSION

In this article, we have considered multiple QoS attributes and studied the multi-objective service composition problem. We have introduced QoS dependency, which is important in real-life service composition. Based on the Pareto set model, we have defined dominance relationship, and focused on finding the Pareto optimal composition solutions efficiently. We first introduced a candidate pruning algorithm which removes the dominated candidates and keeps only the correlated and potential ones, reducing search space. We then presented theoretical analysis which demonstrates that the optimality can still be guaranteed after candidate pruning. Based on Vector Ordinal Optimization techniques, we proposed the DASC algorithm for service composition in the presence of QoS dependency to obtain good composition solutions with high efficiency. The effectiveness of our algorithms has been further validated by simulation experiments.

For our future work, we would consider multi-objective service composition with incomplete QoS data, i.e., QoS data missing values in some attributes. Exhaustive search method can help to solve this problem; however, the execution time cost would be high. Thus, some efficient approaches should be explored. Besides, domain-specific QoS is also important in service composition problems, and we would conduct research on it in our future work. Another direction of future work is to study other structured workflows apart from sequential workflows. Graph theory may help to solve this problem. Besides, we would spend efforts to searching for QoS dependency data and proposing methods to transform the QoS dependency data to standard input.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 61472199) and the Fundamental Research Funds for the Central Universities (No. 2015RC22).

REFERENCES


This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2607750, IEEE Transactions on Cloud Computing.


[38] B. Batoche, Y. Naudel, I. Jars, T. Latour, and F. Guinand, “A multi-objective evolutionary algorithm to optimize the dynamic composition of semantic web services.”


