Service Selection for Composition with QoS Correlations

Shuiguang Deng, Hongyue Wu, Daning Hu, and J. Leon Zhao, Senior Member, IEEE

Abstract — QoS as an important criterion has attracted more and more attention in the service selection process. Various QoS-aware service selection methods have been proposed in recent years. However, few of them take into account of the QoS correlations between services, causing several performance issues. QoS correlations can be defined as that some QoS attributes of a service are not only dependent on the service itself but are also correlated to other services. Since such correlations will affect QoS values, it is important to study how to select appropriate candidate services while taking into account of QoS correlations when generating composite services with optimal QoS values. To this end, we propose a novel method of service selection, called the Correlation-Aware Service Pruning (CASP) method. It manages QoS correlations by accounting for all services that may be integrated into optimal composite services and prunes services that are not the optimal candidate services. Our experiments show that this method can manage complicated correlations between services and significantly improve the QoS values of the generated composite services.

Index Terms — QoS, QoS correlation, Correlation-aware service pruning method, service composition, service selection.

1 INTRODUCTION

A Web service is a functional module realized by Web-accessible programs, databases, sensors and a variety of other physical devices on the Web, and is identified by Uniform Resource Identifiers [1]. Recently, with the advantages of high interoperability, cross-platform-ability and loose coupling, Web service technologies are being widely used by businesses and the scientific community. However, in many cases, no single Web service can fully satisfy users’ needs. Service composition mechanisms can help to achieve complex requirements by composing a set of services. In the process of service composition, a service plan is given as a workflow according to user request. Service selection is an important step during service composition, which selects a proper service for each task of the plan from a set of candidate services.

Quality of service (QoS), which is usually employed for describing the non-functional characteristics of Web services, has become an important differentiating point of different Web services [2]. With the increase of services on the Web, users are paying more attention to the global QoS of the composite services, as there may be many available services with the same functionality. Hence, the objective of service selection is not only to make the generated composite services satisfy user’s functional requirements, but to optimize certain QoS criteria, known as QoS-aware service composition. QoS-aware service composition is the selection of a service for each task in the service plan so as to optimize the overall QoS criteria of the composite service. Many works have addressed QoS-aware service composition and recommendation problems, such as [3-6] These works focus on this problem and give their solutions from different perspectives. They assume all the QoS criteria of services are predetermined and use greedy-like algorithms to select one service with optimal QoS criteria for each task from a set of services with the same functionality.

However, most studies do not account for the QoS correlations between services, which may have significant impact on the overall QoS values of the composite services. QoS correlations occur when some QoS attributes of a service are not only dependent on the service itself but correlated to some other services. QoS correlations exist everywhere in real life. For example, if two adjacent tasks in a service plan can be achieved by two individual services deployed on the same provider, Alicloud, then the response time of the composite service will be greatly reduced as the parameter transmission between the two services can be finished inside the server, saving the cost of data transmission. Another example is a Microsoft sales promotion that claims that a discount on the execution cost will be given if two or more services are used in the same plan. What’s more, if a user has just successfully invoked a service from Amazon, this implies that other services from Amazon will be more accessible to him. If these relevant QoS correlations are taken into consideration in service selection, the ultimate result will be significantly improved. Nevertheless, this important factor is neglected by most of the existing works.

In this paper, we focus on QoS-aware service composition and take QoS correlations into consideration. To achieve this object, we propose a novel method named Correlation-Aware Service Pruning that reserves the services with QoS correlations and pruning redundancy services step by step (CASP). First, as a component of the CASP approach, a preprocessing algorithm named P4CS

• Shuiguang Deng and Hongyue Wu are with College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mails: dengsg@zju.edu.cn, hongyue_wu@163.com.
• Daning Hu is with Department of Informatics of the University of Zurich E-mail: huaning@gmail.com.
• J. Leon Zhao is with Department of Information Systems, City University of Hong Kong, Kowloon, Hong Kong. E-mail: jleonzhao@gmail.com.
(preprocessing for candidate services) is proposed to remove services that cannot contribute to the optimal composite services from the candidate service sets. Then, two CASP algorithms, namely CASP4AT (CASP for service selection with correlations in adjacent tasks) and CASP4NAT (CASP for service selection with correlations in nonadjacent tasks), are proposed for service selection with correlations in adjacent tasks and nonadjacent tasks respectively. Both of them select services for each task in the service plan step by step. In this process, they account for all services that may compose optimal composite services and prune services that are concluded not the optimal candidate services. It is proved that both algorithms can generate optimal composite services effectively. The experiments show that: (1) the QoS values of the composite services generated by CASP algorithms are significantly improved compared to methods that do not take service correlations into account; (2) the execution times of all three algorithms are in a low order of magnitude.

In this paper, we propose the QoS correlation problem and design a method to solve this problem for the first time. As QoS correlations have significant effect on the QoS of composite services, our method can greatly improve the QoS of the generated composite services by accounting for QoS correlations in service selection processes. Note that, we only consider the service selection methods of service plans with sequence structure. The selection methods of service plans with more complex structure will be focused on in later work.

The rest of the paper is organized as follows. In Section 2, two motivation scenarios are introduced to make the problem clearer. In Section 3, we discuss related works. In Section 4, we define the problem and present the formal models. Then we describe the main operations and algorithms in detail in Section 5. In Section 6, we show the evaluation experiments and analyze the results. Finally, we conclude the paper in Section 7.

2 Motivation Scenarios

In the following, we introduce two specific real-life examples to make the problem clearer and illustrate the necessity and significance of our work.

2.1 Scenario A

In this scenario, we assume that the user wants to get a composite service with the shortest response time. The response time of a service is the expected delay between the moment when a request is sent and the moment when the results are received. The response time of a service is the sum of input parameter transmission time, service execution time, and output parameter transmission time.

Fig. 1 shows a simple example of service selection, where rectangles denote tasks and circles denote services. The service plan contains three tasks; tasks $t_1$ and $t_2$ have two candidate services and task $t_3$ has three candidate services. The details of the seven services are shown in Table 1, where the RT column shows the average response time, SET column shows the average service execution time, IPTT column and OPTT column show the average input and output parameter transmission time respectively, with the relation that RT is the sum of SET, IPTT and OPTT. These data are set according to [7]. Furthermore, these services are from five providers. Although service providers can have data centers scattered over different locations, for convenience, here a provider is considered as a data center whose services are deployed on a same physical location. Therefore, the time spent on parameter transmission inside a provider can be safely omitted.

Fig. 1. Service selection example

<table>
<thead>
<tr>
<th>Table 1: The time cost of the services in the example</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>s_1</td>
</tr>
<tr>
<td>s_2</td>
</tr>
<tr>
<td>s_3</td>
</tr>
<tr>
<td>s_4</td>
</tr>
<tr>
<td>s_5</td>
</tr>
<tr>
<td>s_6</td>
</tr>
<tr>
<td>s_7</td>
</tr>
</tbody>
</table>

The response time of the composed service is the sum of response times of the three selected services. In traditional methods, greedy-like algorithms are employed to select the service with shortest response time for each task. The composite service generated by these methods is $s_3 \cdot s_2 \cdot s_1$, with the response time of 881 ms. However, we find that $s_2$ and $s_3$ are with the same provider, and if both of them are selected, the OPTT of $s_2$ and the IPTT of $s_3$ can be saved, as the parameters can be transmitted inside the server. In that case, the response time of the composite service is 654 ms (the time spent on parameter transmission inside the server can be ignored, as it is in a far smaller order of magnitude). Therefore, the optimal composite service is $s_2 \cdot s_3 \cdot s_1$, with saving of more than 25% in response time. Note that, this result depends on the network condition and the distance between the user and the service server. In some cases with worse conditions, the result may be more evident.

2.2 Scenario B

In this scenario, we assume that a composite service with the lowest price is expected. The execution cost of a service is the fee that a service requester has to pay for invoking the service. The execution cost of a composite service is the sum of the execution costs of all its component services.

We still use the service plan in Fig. 1 as the example. The detailed execution costs of the seven services are shown in Table 2, where the DC column shows the
default execution cost, CS column shows the correlated service and CC column shows the correlated execution cost, which applies to the service if and only if one of its correlated services is invoked earlier in the composite service. For example, the execution cost of \( s_1 \) is 12 if \( s_2 \) is invoked earlier, otherwise, it is 18. This means that if two services from the same provider are invoked in a service plan, a discount on the execution cost will be given to the latter one. “n.a.” denotes that the Web service does not have a correlated QoS value for that attribute.

![Fig. 2. Complex case example](image)

In some cases in the real world, the service plan may be more complex, and each task has many candidate services with numerous correlations existing between them. Fig. 2 shows an example of this case, where arcs denote correlations between services. To achieve the optimal composite service, we should select one service for each task. However, some of the services are correlated to other services, and the combination of two services that have QoS correlations may be better than other combinations. Thus it is difficult to decide which service should be selected to compose the optimal composite service at present.

### 3 RELATED WORK

In the field of service computing, QoS has received increased attention in recent years, and a lot of QoS-driven approaches have been proposed for Web service selection [3], [6], [8], service optimization [9] and optimal service composition [10], [11]. These approaches are proposed from different perspectives to resolve different QoS-aware problems. Many works focus on workflow QoS-aware service compositions that select the most suitable services for tasks in the workflow to optimize the execution plan [5], [12]. Many works describe QoS-aware service composition as a multidimensional, multi-objective, multi-choice knapsack problem (MMMKP) [13], which takes many QoS criteria into consideration and tries to obtain a composite service with optimal synthesitical QoS value. Many approaches focus on reducing the complexity of the composition using standard optimization algorithms [14], [15]. Some researchers propose QoS constraints and exploit simple or complex QoS constraints such as minimum availability and reliability to restrict the composite services [16], [17]. Moreover, several works addressed the problem of generating service composition workflows that satisfy QoS constraints and optimize certain QoS criteria [3-5], [18]. However, most of them do not account for the service and QoS correlations, which may have significant impact on the QoS value of the composite services.

There is a stream of research focusing on preferences-based planning for Web service composition [19], [20], [21]. The relevant researchers considered user preferences as a key component of Web service composition and aimed at automatically generating high-quality plans to optimize user preferences. In this paper, QoS considerations can be viewed as preference-criteria based on which composition of Web services may be conducted. However, there are some differences between these studies and our work: 1) they mainly focus on automatically generating the plans of composite services including tasks and flows, but we aim to select services based on users’ preference with given service plans and candidate services; 2) they do not consider the correlations between the QoS of services, which may have great effect on composite services, but we have taken these correlations into consideration in this paper to improve the QoS of composite services.

Many works have addressed the problem of service correlations, and several kinds of service correlations have been analyzed, such as sequential dependency [22], static dependency [23] and dynamic dependency [23]. QoS correlations play a major role in obtaining composite services with high quality. The authors of [24] presented a novel approach to extract dynamic correlations among services using the concept of vector clocks. Ai and Tang [25] presented a repair genetic algorithm to address the Web service composition optimization problem in the presence of domain constraints and inter-service correlations and conflicts. Winkler et al. [26] studied QoS
correlations and modeled correlations in service composition in composite service-level agreement management. Tao et al. [27] proposed an approach to solve the correlations among services in manufacturing grids by using a particle swarm optimization technique. Barakat et al. [28] presented a correlation-aware composition approach to model quality correlations among services during composite service selection. Wagner et al. [29] proposed an approach that takes into account the time and input aspects which affect the QoS values of a service. However, these approaches address the service selection problem with QoS correlations but do not incorporate QoS correlations directly into the composition process.

Feng et al. [30] have considered QoS-aware service composition in the presence of service-dependent QoS and proposed a method that dynamically refined the composed services in light of QoS correlations. This approach is capable of dealing with QoS-aware service composition with QoS correlations. However, the approach is analogous to an enumeration approach that returns all available composite services that satisfy the topological and QoS constraints. Therefore, the algorithm requires a very large space to store the composite services and spends a considerable amount of time generating these composite services. Moreover, they haven’t considered how to select the optimal composite service from the returned available composite service set.

Our previous work addressed this problem. To address the challenge of the automatic composition of Web services in a large scale, we proposed a novel approach to find top-k solutions with optimal global QoS [31]. In [32] we proposed a novel approach based on planning-graphs to solve the top-k QoS-aware automatic service composition problem, which can not only find the optimal solution but also provide several alternative solutions with the optimal QoS. In [33], we focused on the correlations of response time and proposed an optimal deployment method for service composition to reduce the cost on data transmission. In [34], we propose a parallel optimization method for automatic QoS-aware service composition to improve the efficiency of composition.

4 Problem Definition

In this section, we define the problem. The basic definition of Web service, service plan and service composition are presented. QoS correlations and QoS models are also given to formalize our methods.

4.1 Formalization of Service Composition

First, we give the formal definition of Web service.

Definition 1 (Web Service). A Web service is a 4-tuple ws=(i, f, b, QoS), where:
(1) i is the unique identifier of the service;
(2) f is the functional description of the service, including the input, output, pre-condition and result of the service;
(3) b is the basic information of the service, including its name, location, provider, etc.;
(4) QoS is a set of attributes, including execution cost, response time, reliability, availability, reputation, etc.

In Definition 1, each service is identified by i, f describes the function of a service, which is used to aggregate candidate services for tasks in the service plan, and QoS will be defined in detail by Definition 5.

Definition 2 (Service Plan). A service plan is a triple sp=(T, P, B), where:
(1) T = \{t\}_{i=1}^n is a set of tasks, including two special tasks: a beginning task b and an ending task e;
(2) P is a set of settings in the service plan (e.g., execution probabilities of the branches and loops structures);
(3) B provides the structural information of the service plan, which can be specified by XML-based languages, such as BPEL.

A service plan is an abstract description of a business process. Each service plan begins with task b and ends with task e. Fig. 1 shows a simple example of a service plan that includes three abstract tasks. Each service plan task can be realized by invoking an individual service. There may be multiple services with different QoS and providers that can be adopted to fulfill each task.

Definition 3 (Composite Service). A composite service is a triple cs=(S, B, QoS), where:
(1) S is the set of Web services composed in the composite service;
(2) B provides the structural information of the service plan, which can be specified by XML-based languages, such as BPEL;
(3) QoS = \{q\}_{i=1}^n, expressing the QoS of the composite service.

Composite services are obtained by selecting one individual service for each task in the service plan and composing them according to the plan structure. We regard the composite service that not only satisfies user’s functional requirements but has optimal QoS value as the optimal composite service. In service composition process, the business process proposed by the user is specified as a service plan first; then, service selection is processed to select component services according to the service plan; finally, the selected services are composed according to the plan structure. Service selection is considered the most pivotal step of service composition, as it determines the quality of the composite service.

It is noted that an individual service can also be regarded as a composite service with only one component service. We use the symbol “•” to denote the sequence composition of two services. For example, s•s denotes a composite service composed of s and s in sequence.

Definition 4 (Prefix). Suppose cs, cs, and cs are three composite services, with cs=cs•cs, then cs is called a prefix of cs.

For example, both cs and cs•cs are prefixes of composite service cs•cs•cs. Service selection can be regarded as a process of lengthening the prefix of the optimal composite service one by one and eventually achieving the optimal composite service.

4.2 Service QoS Models

In some conditions, some QoS attributes of a service are not only dependent on the service itself but correlated to
other services. For simplicity in the rest of the paper, we state that a service is correlated to another service instead of saying that a QoS attribute of a service is correlated to another service. QoS attributes of a service may be correlated to more than one service (one-to-many service correlations), and there may be more than one service whose QoS attributes are correlated to one service (many-to-one service correlations). In more complex conditions, a service may be involved in both kinds of service correlations. We have considered all these conditions in this paper.

**Definition 5 (QoS).** QoS is a set of attributes, \( QoS = \{ q \}_{x=1}^{n} \), and each attribute is a 4-tuple, \( q = (d, c, S, s) \), where:

1. \( d \) is the default value of \( q \) of the service;
2. \( c \) is the correlated value of \( q \) of the service;
3. \( S \) is the set of services that the value of \( q \) is correlated to;
4. \( s \) is the set of services whose value of \( q \) is correlated to the service.

Each service may have several QoS attributes, and each QoS attribute is defined by four elements. The default value applies to the service if no services in \( S \) are invoked first, while correlated value applies to the service if and only if one or more services in \( S \) are invoked first. If one service is selected, then all the services in its \( S \) will appear with their correlated value.

**Definition 6 (QoS Correlation).** A QoS correlation is a triple \( cor= (q, s, s) \), implying a relationship, that is, the QoS attribute \( q \) of \( s \) is correlated to \( s \).

For example, suppose two services \( s \) and \( s \) with correlation \( q(s, s, s) \), \( s \) is to be composed with a composite service \( cs \). If \( cs \) contains \( s \), then the value of \( q(cs; s) \) is the aggregation of \( q(cs) \) and \( s_2, \) otherwise, \( q(cs; s) \) is the aggregation of \( q(cs) \) and \( s_1, \) \( s \).

QoS attributes can be divided into user-independent attributes and user-dependent attributes [7]. Values of user-independent QoS attributes, such as execution cost and reputation, are usually determined by services themselves and their providers, thus they are identical for different users. On the other hand, values of user-dependent QoS attributes, such as response time and reliability, may vary widely for different users, as they are influenced by the unpredictable network conditions and the heterogeneous user environments. Various methods have been proposed to forecast the user-dependent QoS attributes for a certain user and environment [35]. In this paper, we assume that all user-dependent QoS attributes are pre-calculated.

It is reasonable to assume that all QoS attributes of a service may have QoS correlations with other services. In the rest of this paper, we will mainly use response time and execution cost as the example to describe our approach, while it is also suitable for other attributes.

If a service is a nested service, its QoS should be the aggregate QoS value of the invoked service and any nested services. For example, the cost of a nested service can be computed by summing the cost of the invoked service and the nested service and the response time of the nested service can be computed by adding the execution times of the two services and the data transmission time. In the process of service selection, every service acts as a whole.

There may exist lock-in in the service selection process. If a lock-in exists between a pair of providers, users will be unable to replace one of them without substantial switching costs. In other words, if a service is involved in a lock-in, it must be composed with the service it is locked with, and if it is composed with other services, a switching cost is required. In this manuscript, we can model lock-in problems by transforming the switching cost to QoS correlations. We can add a correlation between the two services involved in a lock-in, assign the default QoS value of each service with the sum of its inherent value and the switching cost, and assign the correlated QoS value of each service with its inherent value. Once doing so, the switching cost is added to the QoS of both services if they are composed with other services.

5 CASP Approach

In this section, we briefly introduce our approach’s main operations and algorithms. The candidate service sets are preprocessed. Then the CASP approach is implemented. As correlations can be divided into correlations in adjacent tasks and nonadjacent tasks, we propose two kinds of CASP algorithms accordingly.

5.1 Preprocessing of Candidate Service Sets

There may be a large number of candidate services for a task in a service plan, and too many redundant services will increase the complexity of the service selection process. In order to decrease the complexity of service selection, Algorithm-P4CS is proposed to remove the redundant services from candidate service sets.

In the candidate services sets, all services free of correlations will be removed except the service with the optimal default QoS value. For two certain tasks, if there is more than one correlation between the corresponding candidate service sets, only the one with optimal composite QoS value is reserved; others will be removed.

The symbols used in this section are defined in Table 3.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \rightarrow b )</td>
<td>assign variable ( t ) with ( b )</td>
</tr>
<tr>
<td>( b \rightarrow \text{task} )</td>
<td>the subsequent task of ( b ) in the service plan</td>
</tr>
<tr>
<td>( t_{\text{optWS}} )</td>
<td>the service with optimal default QoS value in the candidate service set corresponding to task ( t )</td>
</tr>
<tr>
<td>( t_{\text{firstWS}} )</td>
<td>the first service in the candidate service set corresponding to task ( t )</td>
</tr>
<tr>
<td>( t_{\text{WS}} )</td>
<td>the candidate service set corresponding to task ( t )</td>
</tr>
<tr>
<td>( q(s) )</td>
<td>the value of ( q ) for service ( s )</td>
</tr>
<tr>
<td>( q(s) \geq q(s) )</td>
<td>the value of ( q ) of ( s ) is better than that of ( s. )</td>
</tr>
<tr>
<td>( s_{\text{task}} )</td>
<td>the task that service ( s ) corresponds to</td>
</tr>
<tr>
<td>( s_{\text{S}} )</td>
<td>the set of services that the QoS of ( s ) is correlated to</td>
</tr>
<tr>
<td>( s_{\text{S}} )</td>
<td>the set of services whose QoS is correlated to ( s )</td>
</tr>
<tr>
<td>( s_{\text{S}} )</td>
<td>the composition of services ( s ) and ( s ), by sequence</td>
</tr>
<tr>
<td>( cs )</td>
<td>the composite service obtained by replacing ( s ) with ( cs )</td>
</tr>
</tbody>
</table>

Each task is a step in Algorithm-P4CS. For each task,
the preprocessing on its candidate service set can be
generally divided into two steps. The first step (lines 3 to
10) is to find the service with optimal default value of \( q \) 
and remove all the other services free of correlations. The
symbol "\( \succ \)" is used to represent better than, as for some
QoS criterion (reliability, availability, reputation) a higher
value is better, while for others (execution cost, response
time) a lower value is better. The second step (lines 11 to
18) is to remove services with non-optimal correlations.
As a service may be involved in one-to-many and many-
to-one service correlations, if one correlation involving it
is worse than another, the algorithm will continue to
check if there is any other correlations involving it; if so, it
will be reserved, and otherwise be removed.

The time complexity of Algorithm-P4CS is \( O(mn) \),
where \( m \) denotes the number of tasks and \( n \) denotes
the average number of candidate services. Algorithm-P4CS
has a low time complexity, but it has great impact on the
efficiency of service selection, especially when the number of
candidate services is very large, as it significantly decreases
the number of candidate services.

**Theorem 1.** Let \( sp \) be a service plan, \( q \) is the QoS value
preferred by the user, \( ocws \) is the optimal composite service of
\( sp \) and \( RS \) is the set of removed services after the preprocessing
of Algorithm-P4CS, then \( \forall s \in RS \), \( s \) is not contained in \( ocws \).

**Proof.** We adopt reductio ad absurdum to prove this
theorem. Assume that \( \exists s \in RS \), such that \( s \) is contained in
\( ocws \), then \( s \) must be removed from either the first step or
the second step of Algorithm-P4CS. If \( s \) is removed from the
first step, then \( s \) must be free of correlations and there
must be a service \( s' \) such that \( q(s') \succ q(s) \), so \( q(ocws)
(s\rightarrow s') \succ q(ocws) \) holds, which is contradictory with the 
assumption that \( ocws \) is the optimal composite service of
\( sp \). If \( s \) is removed from the second step, then there must be
another service \( s_1 \), such that \( s_1 \in S \) (or \( s_1 \in S \)), and \( s \)
is also contained in \( ocws \), otherwise \( q(ocws(s\rightarrow
s_task_optWS)) \succ q(ocws) \) holds. Moreover, there must be
\( s \), \( s_1 \), and \( s_2 \), such that \( s_1 \in S_1 \), \( s_2 \in S_2 \), \( s_1 \succ s_2 \), and
\( s_1 \succ s_2 \), with relationship \( q(s_1 \cdot s) \prec q(s_2 \cdot s) \).
Then \( q(ocws(s\rightarrow s_2 \succ s_2 \succ s)) \succ q(ocws) \) holds, which
is contradictory with the assumption. Therefore, \( \forall s \in RS \), \( s \) is
not contained in \( ocws \).

Theorem 1 shows that Algorithm-P4CS does not affect
the optimal composite service of a service plan.

Fig. 3 shows an example of the preprocessing. Suppose
that the preferred QoS attribute is execution cost. For
simplicity, the value of the execution cost of each service
is labeled inside the circle. All correlated services have
two values, the former is the default execution cost and
the latter is the correlated execution cost. Arcs between
services show the correlations among them. As \( q(s_1) \prec q
(s_2) \) and \( s_1 \) is free of correlations, \( s_1 \) is removed first, then \( s_2 \)
and \( s_3 \) are removed for the same reason. As \( q(s_4 \cdot s_5) \prec q(s_6 \cdot s_7) \)
and both \( s_4 \) and \( s_5 \) are free of other correlations, they are
removed at the same time. \( s_4 \) and \( s_5 \) are removed
for the same reason in the next step. Finally, \( s_6 \) is removed
as it is free of correlations and its default QoS value is not
optimal.

---

**Algorithm-P4CS: Preprocessing Algorithm of Candidate Service Sets**

**Input:** service plan \( sp \) with beginning task \( b \) and ending task \( e \), candidate service sets \( css \), correlation set \( cs \) and user preference \( q \)

**Output:** candidate service sets after preprocessing

```plaintext
1 t←(b→task)
2 while t ≠ e
3 t_optWS←t_firstWS
4 for every s∈I_WS
5 if q(s) ≥ q(t_optWS)
6 if t_optWS is free of correlations
7 remove t_optWS from t_WS
8 t_optWS←s
9 else if q(s) ≤ q(t_optWS) and s is free of correlations
10 remove s from t_WS
11 for every s∈I_WS and s_S≠∅
12 if ∃s′∈S, s,s′∈I_task WS, s∈I_WS, such that q(s′ · s) < q(s · s)
13 remove the correlation between s′ and s
14 if s′ is free of correlations and s′ is not t_task_optWS
15 remove s from t_task_WS
16 if s is free of correlations and s is not t_optWS
17 remove s from t_WS
18 t←(t→task)
19 return css and cs
```

---

![Fig. 3. Preprocessing example](image-url)
After preprocessing of candidate service sets, a large number of services are removed, and only the services with optimal default QoS value and the services with excellent correlations are reserved. Therefore, Algorithm-P4CS significantly reduces service searching space and decreases the difficulty of generating the optimal composite services. As the number of tasks in a service plan will not be too large, the number of candidate services after preprocessing is smaller than a determined value, which is related to the task number.

5.2 CASP for Service Selection with Correlations in Adjacent Tasks

In some cases, correlations only exist in services corresponding to adjacent tasks (e.g., example in Scenario A). In these cases, we can reserve all composite services that may be the prefix of the optimal composite service and prune the composite services the first time they are determined not the prefix of the optimal composite service. In other words, service composition can be viewed as a process of generating possible prefixes of the optimal composite service and pruning those that do not meet the requirement, shown as Algorithm-CASP4AT.

Algorithm-CASP4AT: Algorithm for Service Selection with Correlations in Adjacent Tasks

Input: service plan \( sp \) with beginning task \( b \) and ending task \( e \), candidate service sets \( css \), correlation set \( cs \) and user preference \( q \)

Output: composite service with optimal value of \( q \)

1. \( optCWS \leftarrow \text{null} \)
2. \( CorCWSSet \leftarrow \emptyset \)
3. \( t \leftarrow (b \rightarrow \text{task}) \)
4. while \( t \neq e \)
5. \( \text{create new CorCWSSet} \)
6. for every \( s \in \text{Et}_WS \) and \( s \_S \neq \emptyset \)
7. \( \text{add } optCWS \_s \text{ to new CorCWS} \)
8. \( optCWS \leftarrow optCWS \_t \_optWS \)
9. for every \( cws \in CorCWSSet \)
10. \( \text{remark the service correlated to cws as } s \)
11. if \( q(cws \_s) \succ q(optCWS) \)
12. \( optCWS \leftarrow cws \_s \)
13. if \( s \_S \neq \emptyset \)
14. \( \text{add } cws \_s \text{ to new CorCWS} \)
15. \( CorCWSSet \leftarrow \text{new CorCWSSet} \)
16. \( t \leftarrow (t \rightarrow \text{task}) \)
17. return \( optCWS \)

Before the algorithm starts, the current optimal composite service is set as null (line 1). \( CorCWSSet \) (line 2) is the set of composite services that contain one or more services correlated with subsequent services. The algorithm begins with the first task (line 3). For a task, it composes the current optimal composite service with all the services that are correlated with subsequent services and adds them to \( CorCWSSet \) (lines 6 and 7). Then it chooses the service with the optimal default QoS value, composes it with the current optimal composite service and sets it as the new current optimal composite service (line 8). Each composite service in \( CorCWSSet \) is composed with the services that correlated to it; then, the new composed composite service will be compared with the current optimal composite service; the better one of the new composed composite service and the current optimal composite service will become the new current optimal composite service and the worse one will be removed (lines 9-12). If the correlated service is also correlated with other services in the following task, the composed service will be added to \( CorCWSSet \) (lines 13 and 14). The algorithm will go forward like this step by step, finally returning the current optimal composite service as the result (line 17).

The time complexity of Algorithm-CASP4AT is \( O(mn) \), where \( m \) denotes the number of tasks and \( n \) denotes the average number of candidate services. It implies that the execution time of Algorithm-CASP4AT is in a low order of magnitude. It can be proved that Algorithm-CASP4AT can generate the optimal composite service, shown as Theorem 2.

Theorem 2. Algorithm-CASP4AT can generate the optimal composite service of a service plan where correlations exist in adjacent tasks.

Proof. Assume the current optimal composite service before task \( t \) is \( optCWS \). When the process goes forward to task \( t \), the current optimal composite service must be either the composition of \( optCWS \) and the candidate service with optimal default QoS value, or the composition of \( optCWS \) and a service that is correlated with a subsequent service. So all composite services that may generate the optimal composite service of the plan are reserved. If a composite service \( cws \) is removed, there must be \( q(optCWS) \succ q(cws) \) and any service in \( cws \) is not correlated with a subsequent service, so no matter which service \( s \) will be selected in the following, there must be \( q(optCWS \_s) \succ q(cws \_s) \). Therefore, Algorithm-CASP4AT reserves all the composite services that may be the prefix of the optimal composite service and prunes all the composite services that are not the prefix of the optimal composite service, thus it can generate the optimal composite service.

5.3 CASP for Service Selection with Correlations in Nonadjacent Tasks

However, correlations can exist in services that are not adjacent (e.g., example in scenario B). In these cases, the CASP method can also be used to achieve the selection. It reserves all the composite services that may be the prefix of the optimal composite service and prunes the composite services the first time they are determined not as the prefix of the optimal composite service.

Given a composite service \( cws \) in the service selection process, we use \( cws \_corWSSet \) to represent the set of services that are in \( cws \) and correlated with subsequent services. Given two composite services with the same \( corWSSet \), we can conclude that the one with lower composite QoS value cannot be the prefix of the optimal composite service, shown as Theorem 3.

Theorem 3. Let \( sp \) be a service plan with sequence structure, \( q \) is the QoS user-preferred criterion, and \( cws \) and \( cws \_2 \) are two composite services generated in the process of service selection, if \( cws \_corWSSet \succ cws \_2 \_corWSSet \) and \( q(cws) \succ q(cws \_2) \), then \( cws \) is not the prefix of the optimal composite service.
We use reductio ad absurdum to prove this theorem. Assume \( cws \) is the prefix of the optimal composite service, and the optimal composite service is \( cws \cdot cws \). Then \( q(cws) = q(cws) + \sum_{s \in cws} (\exists s \in cws \_corWSSet, such that s \in s \_S \cap cws \_WS) + \sum_{s \in cws \_corWSSet} (\exists s \in cws \_corWSSet, such that s \in s \_S \cap cws \_WS) \), and \( q(cws) \cdot cws = q(cws) + \sum_{s \in cws \_corWSSet} (\exists s \in cws \_corWSSet, such that s \in s \_S \cap cws \_WS) \). As \( cws \_corWSSet = cws \_corWSSet \), and \( q(cws) \cdot cws = q(cws), q(cws) \cdot cws \cdot cws = q(cws) \cdot cws \) holds, which is contradictory with the assumption. Therefore, \( cws \) is not the prefix of the optimal composite service.

Based on Theorem 3, we propose Algorithm-CASP4NAT to achieve service selection with correlations in nonadjacent tasks.

\( CWS \) (line 1) is the set of composite services generated in the process of selection. The algorithm starts with the first task (line 2), \( newCWS \) is set as an empty set in the beginning (line 4). For a task, all the services in \( CWS \) are composed with the service with optimal default value of \( q \), and the generated composite service is added to \( newCWS \) (lines 5 and 6). All services in \( CWS \) are composed with all the services that are correlated with subsequent services (lines 9 and 10). Each composite service in \( CWS \) is composed with the services that correlated to it (lines 13 and 14). In this process, if there is a composite service whose \( corWSSet \) is the same as the newly added composite service in \( CWS \), then the two services will be compared and the one with worse value of \( q \) will be removed (lines 7, 8, 11, 12, 15 and 16). The process will be repeated until it reaches the ending task \( e \). In the last step, all the composite services are with the same \( corWSSet \) (\( \emptyset \)), and only the one with optimal value of \( q \) is reserved; it will be returned as the optimal composite service (line 19).

The time complexity of Algorithm-CASP4NAT is \( O(nmnl) \), where \( m \) denotes the number of tasks, \( n \) denotes the average number of candidate services and \( l \) denotes the average number of composite services in the generated \( CWS \). \( l \) has greater effect on the execution time. It is mainly affected by the number of correlations among services.

**Theorem 4.** Algorithm-CASP4NAT can generate the optimal composite service of a service plan where correlations exist in nonadjacent tasks.

**Proof.** Assume that \( cws \) is a composite service generated before task \( t \). The candidate services of \( t \) can be divided into four categories: 1) the service with optimal default QoS value, 2) services correlated with subsequent services in the candidate service set of the following tasks, 3) services correlated to one or more services in \( cws \), 4) other services. Services in category 4 do not have optimal default QoS value, are not correlated with services in the following tasks, and have no correlation with \( cws \), so \( \forall s \in category 4, cws \cdot s \) can’t be the prefix of the optimal composite service, as for any composite service \( cws' \) of the following tasks, there must be \( q(cws \cdot t \_optWS \cdot cws') \succ q(cws \cdot s \cdot cws') \). So Algorithm-CASP4NAT reserves all the composite services that may be the prefix of the optimal composite service. Moreover, all the removed composite services are not the prefix of the optimal composite service (Theorem 3). Therefore, Algorithm-CASP4NAT can generate the optimal composite service of a service plan.

Theorem 4 illustrates the effectiveness of Algorithm-CASP4NAT. In the following, the example in Section 5.1 is reused to demonstrate the process of Algorithm-CASP4NAT.

Fig. 4 shows an example of Algorithm-CASP4NAT. In Step 1, the services are put into \( CWS \), as they are either with optimal default value or correlated with subsequent services. In Step 2, all the composite services

**Algorithm-CASP4NAT:** Algorithm for Service Selection with Correlations in Nonadjacent Tasks

**Input:** service plan with beginning task \( b \) and ending task \( e \), candidate service sets \( CS \), correlation set \( cs \) and user preference \( q \)

**Output:** composite service with optimal value of \( q \)

1. \( CWS \leftarrow \emptyset \)
2. \( t \leftarrow (b \rightarrow \text{task}) \)
3. **while** \( t \neq e \)
4. \( \text{for every } cws \in CWS \)
5. \( \quad \text{add } cws \cdot t \_optWS \text{ to } newCWS \)
6. \( \quad \text{if } \exists cws \in newCWS \text{ such that } cws \_i \_corWSSet = cws \cdot t \_optWS \_corWSSet \)
7. \( \quad \quad \text{remove the worse one of } cws \text{ and } cws \cdot t \_optWS \text{ from } newCWS \)
8. \( \text{for every } s \in t \_WS \text{ and } s \_S \neq \emptyset \)
9. \( \quad \text{add } cws \cdot s \text{ to } newCWS \)
10. \( \quad \text{if } \exists cws \in newCWS \text{ such that } cws \_i \_corWSSet = cws \cdot s \_corWSSet \)
11. \( \quad \quad \text{then remove the worse one of } cws \_i \text{ and } cws \cdot s \)
12. \( \text{for every } s \in t \_WS \text{ such that } \exists s \in t \_S \)
13. \( \quad \text{add } cws \cdot s \text{ to } newCWS \)
14. \( \quad \text{if } \exists cws \in newCWS \text{ such that } cws \_i \_corWSSet = cws \cdot s \_corWSSet \)
15. \( \quad \quad \text{remove the worse one of } cws \_i \text{ and } cws \cdot s \)
16. \( CWS \leftarrow newCWS \)
17. \( t \leftarrow (t \rightarrow \text{task}) \)
18. **return** \( cws \text{ in } CWS \)
are composed with \( s_i \) and \( s_j \), because \( s_i \) is with the optimal default value and \( s_i \) is correlated with subsequent service. Besides, \( s_i \) is composed with \( s_j \) because there is a correlation between them. After that, \( s_i \) and \( s_j \) are with the same DepWSSet (\( s_i \)), so \( s_i \) is removed, as its QoS is worse. Similarly, \( s_i \) and \( s_j \) are removed. In Step 3, all of the composite services are composed with \( s_i \) and \( s_j \). Besides, \( s_i \) and \( s_j \) are composed with \( s_k \). Thereafter, five composite services are removed just as in Step 2. In Step 4, all of the composite services are composed with the service with \( s_i \) and the service correlated to a service in them. After that, the DepWSSet of all of the new generated composite services are \( \emptyset \). Therefore, \( s_i \) and \( s_j \), the one with optimal QoS, is reserved and feedback to the user and others are removed.

![Fig. 4. Algorithm-CASP4NAT example](image)

In the CASP approaches above, we have assumed complete information about each candidate service for service plans. However, in reality we can only obtain partial-information about candidate services and the QoS value of some services may be missing, due to information gathering nature of included Web services. In that case, we can use assumption-based selection [36] or generation of multiple composite services [37] together with our CASP approaches to select services. Moreover, this problem can also be resolved by adopting QoS prediction methods [7] to predict the missing QoS values of candidate services.

## 6 Experiments

In order to evaluate the effectiveness of the algorithms proposed in this paper, we carry out two sets of experiments. The first set of experiments evaluates the QoS improvement by comparing our approach with the methods that do not account for QoS correlations. The second set of experiments evaluates the efficiency of our methods, where we implement the algorithms in different scale scenarios to examine the scalability of the methods.

We have realized the algorithms using Python language. Experiments are implemented on a computer with Pentium(R) Dual-Core CPU 2.50GHz, 2GB of RAM and Windows 8 operating system.

We choose response time and execution cost to examine the effectiveness and efficiency of Algorithm-CASP4AT and Algorithm-CASP4NAT respectively, while other QoS attributes are analogous. As there is no standard experimental platform and test data sets, we automatically generate the parameters and use them as the experimental data sets. Each service is assigned an integer as the default QoS value, which is generated randomly from 2 to 10. Correlations are randomly generated. When generating correlations, a random value in \((0, 1)\) is generated for each pair of services from different tasks; if the value is smaller than the given percentage, then a correlation is added between the two services. For each correlation, it is certain that the QoS of the service from the latter task depends on the service from the former task, so there is no cycle in the generated correlations. Besides, the ratio between the correlated value and default value is generated randomly from 0.1 to 0.9.

We focus on the following three variables in the experiments:

1. Correlated service percentage. The percentage of services that are involved with correlations among all services.
2. Candidate service number. The number of candidate services of a task in a service plan.
3. Task number. The number of tasks in a service plan.

As analyzed in Section 5, the complexity and efficiency of the three algorithms are mainly related to these three variables. Correlated service percentage reflects the number of correlations among services, which directly affects the results and efficiency of the algorithms. Candidate service number and task number reflect the scales of candidate service sets and tasks respectively, which are the main factors determining the results and efficiency of the algorithms.

### 6.1 Effectiveness Evaluation

To verify the effectiveness of our algorithms, we compare our methods with a greedy algorithm that selects the service with optimal QoS value for each task and does not account for QoS correlations between services in the process of service selection.

#### 6.1.1 Effectiveness Evaluation of Algorithm-CASP4AT

We design three experiments to evaluate the effectiveness of Algorithm-CASP4AT. These experiments vary the values of correlated service percentage, candidate service number and task number, respectively, to examine the QoS improvement of Algorithm-CASP4AT compared to the greedy algorithm. Response time is chosen as the user preference.

**Experiment 1. What is the impact of correlated service percentage on the results of Algorithm-CASP4AT?**

In this experiment, the task number is set to 10; integers randomly generated from 5 to 100 are assigned to each task as the candidate service number; each service is assigned an integer randomly generated from 2 to 10 as the default QoS value. The greedy algorithm is utilized to select the optimal service for each task and eventually...
generate the composite service. Then the correlated service percentage is set to 10; correlations are generated by randomly choosing two services from adjacent tasks; correlated QoS values are randomly generated from 0.1 to 0.9 times of the default QoS value. After that, Algorithm-P4CS and Algorithm-CASP4AT are implemented and the composite QoS value is recorded. Then the correlated service percentage is set to 20, 30, ..., 90, and the experiment is repeated. This process is repeated 1000 times and we adopt the average values.

Results are shown in Fig. 5(a), where the horizontal coordinate represents the value of correlated service percentage and the vertical coordinate represents the average composite QoS value of 1000 experiments. The average QoS value of the composite services generated by the greedy algorithm is about 21. When 10% of the services are correlated, the average composite QoS value decreases to 19.1. When the correlated service percentage increases, the average composite QoS value decreases steadily because there may be more excellent alternative correlations. When the correlated services increase to 90%, the average composite QoS value decreases to 14.6, a more than 30% decrease. We can conclude that the whole QoS value will be significantly improved if correlations are taken into account. Furthermore, the superiority will be more obvious if more correlations are considered.

Experiment 2. What is the impact of candidate service number on the results of Algorithm-CASP4AT?

In this experiment, we set the correlated service percentage to 0.3, and randomly generate integers from 1 to 20 for each task as the candidate service number. Other experimental parameters are set just like Experiment 1. Then we implement Algorithm-CASP4AT and record the composite QoS value. Next, we change all candidate service numbers to randomly generated integers from 21 to 40, from 41 to 60, ..., from 181 to 200, and repeat the experiment. Similarly, each experiment is repeated 1000 times and the average value is adopted.

Fig. 5(b) shows the results. It is obvious that the result is significantly better with QoS correlations taken into consideration. As the candidate services increase, there are more services with better QoS values and more services with better correlations, so the results of both algorithms improve. When the candidate service number exceeds 60, the greedy algorithm results do not decrease any more, as all the candidate services reach the lower limit, while the result of Algorithm-CASP4AT decreases continuously, as more correlations are increased. That is, the superiority of Algorithm-CASP4AT will be more obvious with increased candidate services.

Experiment 3. What is the impact of task number on the results of Algorithm-CASP4AT?

In order to examine the impact, we set the task number to 10, 20, ..., 100, respectively, and randomly generate an integer from 5 to 100 for each task as the candidate service number. Other parameters are the same as Experiment 2. Similarly, each experiment is repeated 1000 times and the average value is recorded.

As shown in Fig. 5(c), Algorithm-CASP4AT performs much better than the greedy algorithm. The superiority will be more obvious with the task number increasing, as more correlations are reserved. Both results are increasing linearly with the task number increasing.

Fig. 5. Effectiveness of Algorithm-CASP4AT

6.1.2 Effectiveness Evaluation of Algorithm-CASP4NAT

In the following three experiments, we verify the effectiveness of Algorithm-CASP4NAT, and execution cost is chosen as the user preference. These three experiments vary the values of correlated service percentage, candidate service number and task number, respectively, to examine the QoS improvement of Algorithm-CASP4NAT. The experimental data sets are similar to that of Experiments 1, 2 and 3, while the correlations are generated among services from two random tasks that may not be adjacent tasks.

Experiment 4. What is the impact of correlated service percentage on the results of Algorithm-CASP4NAT?

We randomly generate the experimental parameters just like Experiment 1, except that correlations are generated among services from two random nonadjacent tasks. Then Algorithm-P4CS and Algorithm-CASP4NAT are implemented.

The results are shown in Fig. 6(a), from which we can see that the greedy algorithm result is about 21. When 10% of the services are correlated, the result decreases to 19.1. With increasing correlated service percentage, it decreases steadily, because there may be more excellent alternative correlations. When the correlated services reach 90%, the average composite QoS value decreases to 14.9, a 30% decrease. We can conclude that the result will be significantly improved if correlations are taken into account. The composite QoS value will steadily improve.
with more correlations considered.

**Experiment 5.** What is the impact of candidate service number on the results of Algorithm-CASP4NAT?

We repeat Experiment 2 and generate correlations among services from two random tasks. Then we implement Algorithm-P4CS and Algorithm-CASP4NAT.

As shown in Fig. 6(b), the results of Algorithm-CASP4NAT are far better than the greedy algorithm. With the candidate services increasing, the greedy algorithm results do not decrease as it reaches the lower limit, while the result of Algorithm-CASP4NAT decreases steadily, as there are more excellent alternative correlations. Therefore, the superiority of Algorithm-CASP4NAT will be more obvious with the candidate service number increasing.

**Experiment 6.** What is the impact of task number on the results of Algorithm-CASP4NAT?

In this experiment, we repeat Experiment 3 to examine the impact of task number. Similarly, we generate correlations among services from two random tasks.

As shown in Fig. 6(c), the result of Algorithm-CASP4NAT is far better than the greedy algorithm. As the task number increases, there are more correlations involved in the optimal composite service; therefore, the gap will be more obvious.

![Fig. 6. Effectiveness of Algorithm-CASP4NAT](image)

These experiments have shown that QoS correlations have great impact on the whole QoS value of composite services. If service correlations are taken into account, the QoS value of the composite services will be significantly improved. With the correlated service percentage, candidate service number and task number increasing, the superiority of our methods will be more obvious.

### 6.2 Efficiency Evaluation

The following experiments are designed to verify the efficiency of the three algorithms. In order to examine the execution time growth trend with the range of the variables, we fit the experimental results and compute the confidence intervals in the following experiments. Similarly, each experiment is repeated 1000 times and the average value is adopt as the result.

#### 6.2.1 Efficiency Evaluation of Algorithm-P4CS

**Experiment 7.** How does the efficiency of Algorithm-P4CS change with the correlated service percentage, candidate service number and task number increasing?

In order to examine the efficiency of Algorithm-P4CS, we randomly generate experimental parameters just like Experiments 4, 5 and 6, respectively, and use Algorithm-P4CS to preprocess the candidate service sets. The execution times are shown in Fig. 7, where the experimental results are shown along with the fitting curves and confidence intervals including upper confidence limit and lower confidence limit curves.

![Fig. 7. Execution time of Algorithm-P4CS](image)

Algorithm-P4CS’s execution time is in a low order of magnitude. As the correlated service percentage, candidate service number or task number increases, the execution time increases at a modest pace. The three groups of experimental results can be fitted by quadratic function, quadratic function and linear function, respectively. Therefore, the results are in accordance with the complexity of Algorithm-P4CS, $O(mn)$.

#### 6.2.2 Efficiency Evaluation of Algorithm-CASP4AT

**Experiment 8.** How does the efficiency of Algorithm-CASP4AT...
CASP4AT change with the correlated service percentage, candidate service number and task number increasing?

We repeat Experiments 1, 2 and 3, record the execution time of Algorithm-CASP4AT and adopt the average value of the 10-times repeated experiments. The results are shown in Fig. 8. The execution time of Algorithm-CASP4AT is in a low order of magnitude. All of the three groups of experimental results can be approximately fitted by linear functions. Therefore, with the correlated service percentage, candidate service number and task number increasing, the execution time of Algorithm-CASP4AT increases linearly, which is in accordance with the complexity of Algorithm-CASP4AT, $O(mn)$.

![Fig. 8. Execution time of Algorithm-CASP4AT](image)

6.2.3 Efficiency Evaluation of Algorithm-CASP4NAT

Experiment 9. How does the efficiency of Algorithm-CASP4NAT change with the correlated service percentage, candidate service number and task number increasing?

Experiments 4, 5 and 6 are repeated again and the average execution time of Algorithm-CASP4NAT is recorded, as shown in Fig. 9. The execution time of Algorithm-CASP4NAT is in a low order of magnitude. However, all of the three groups of experimental results are fitted by exponential functions. Therefore, all the three parameters have big effect on the execution time of Algorithm-CASP4NAT. With the growth of correlated service percentage and candidate service number, the execution time of Algorithm-CASP4NAT increases steadily. The number of tasks in the service plan has a larger effect on the execution time of Algorithm-CASP4NAT. It will spend little time if the task number is small. However, if a service plan has a large number of tasks, it will spend enough time for the Algorithm-CASP4NAT to obtain the result. Fortunately, the task number can’t be very large in the real life.

![Fig. 9. Execution time of Algorithm-CASP4NAT](image)

According to the experiments above, all three algorithms perform with good scalability. As the experimental scale grows, the execution time of Algorithm-P4CS increases steadily. Algorithm-CASP4AT performs linear relations to all experimental variables. As for Algorithm-CASP4NAT, if the task number is not very large, it will cost little time, while if the task number is large, it may cost a large amount of time, but the task numbers are not very large in real life.

7 CONCLUSION

In this paper we focus on QoS-aware service composition and take QoS correlations between services into account. A novel method, CASP, is proposed to achieve the objective. As QoS correlations have significant effect on the QoS of the composite services, our CASP method can greatly improve the QoS of the generated composite services by taking QoS correlations into account in the service selection process. We proved that this method can generate the optimal composite service effectively. Our experiments also show the effectiveness and efficiency of the method.

However, we only consider service plans with sequence structure in this paper. Therefore we will extend our work to service plans with more complex structures in the future. Furthermore, more complex scenarios with requirements on multidimensional QoS attributes will be
taken into account.

ACKNOWLEDGMENT

This research was supported in part by the National Key Technology Research and Development Program of China under Grant No. 2013BAD19B10 and National Natural Science Foundation of China under Grant No. 61170033.

REFERENCES


Shuiguang Deng received the BS and PhD in Computer Science from Zhejiang University in 2002 and 2007, respectively. He is an associate professor in the College of Computer Science at Zhejiang University. At present, he is a visiting scholar in MIT. His research interests include Service Computing, Business Process Management and Data Management. Up to now, he has published more than 30 papers in peer-refereed journals and international conference proceedings as the first author or the corresponding author. And also, he has held a number of patents for his many innovations. He is the recipient of Microsoft Fellowship Award 2005. He is member of ACM.

Hongyue Wu received the B.S. and M.S. degree in computer science and technology in 2010 and 2013, respectively. Now he is a PhD of computer science and technology in Zhejiang University. His research interest focuses on service computing and Petri net theory and application.

Daning Hu is an Assistant Professor of the Department of Informatics at University of Zurich and Head of the Business Intelligence Research Group. He holds his Ph.D. degree in Management Information Systems (minor in Finance) from the University of Arizona, and B.S. degree in Computer Science from Zhejiang University. His research goal is to derive analytical and empirical insights from the analyses of various real-world networks such as organization networks, communication networks, and social networks. Based on such insights, he aims to develop network-based business intelligence techniques and information systems for supporting decision making in various application domains. He has published in MIS Quarterly, Decision Support Systems, ACM Transaction on MIS, Journal of the Association for Information Systems, Journal of the American Society for Information Science and Technology.

J. Leon Zhao is Chair Professor and Head of the Department of Information Systems at City University of Hong Kong. He was Interim Head and Eller Professor in MIS, University of Arizona and taught previously at HKUST and College of William and Mary, respectively. He holds Ph.D. and M.S. degrees from the Haas School of Business, UC Berkeley, M.S. degree from UC Davis, and B.S. degree from Beijing Institute of Agricultural Mechanization. His research is on information technology and management, with a particular focus on workflow technology and applications in knowledge distribution, e-learning, supply chain management, organizational performance management, and services computing. Leon’s research has been supported by NSF, SAP, and other sponsors.