A Reinforcement Learning Method for Constraint-Satisfied Services Composition

Lifang Ren, Wenjian Wang, Hang Xu

Abstract— With increasing adoption and presence of Web services, service composition becomes an effective way to construct software applications. Composite services need to satisfy both the functional and the non-functional requirements. Traditional methods usually assume that the quality of service (QoS) and the behaviors of services are deterministic, and they execute the composite service after all the component services are selected. It is difficult to guarantee the satisfaction of user constraints and the successful execution of the composite service. This paper models the constraint-satisfied service composition (CSSC) problem as a Markov decision process (MDP), namely CSSC-MDP, and designs a Q-learning algorithm to solve the model. CSSC-MDP takes the uncertainty of QoS and service behavior into account, and selects a component service after the execution of previous services. Thus, CSSC-MDP can select the globally optimal service based on the constraints which need the following services to satisfy. In the case of selected service failure, CSSC-MDP can timely provide the optimal alternative service. Simulation experiments show that the proposed method can successfully solve the CSSC problem of different sizes. Comparing with three representative methods, CSSC-MDP has obvious advantages, especially in terms of the success rate of service composition.

Index Terms—Web service composition, constraint-satisfied, uncertainty of service behaviors, undetermined QoS, Markov decision process (MDP), Q-learning algorithm.

1 Introduction

With the rapid development of cloud computing, services are emerging as a powerful vehicle for organizations to deliver their applications over the Internet. As the number of services increases, software applications are no longer built from scratch, but rather through integration of available services distributed in the web, which leads to the service composition. In this way, component services can be integrated into more capable composite services to fulfill more and more complex demands of users. Therefore, it is an inevitable trend to integrate the available services to meet various requirements from users [1], [2].

It is expected that there will be an increasing number of services with the same functionality and different quality of service (QoS), such as response time, availability, reliability, throughput, price, success rate, and so on. Nevertheless, the Internet environment is dynamic and the service evolutions take place erratically, which leads to the uncertain QoS and service behavior. Such as, the increase in network traffic will bring a prolonged response time, and sometimes a service is temporarily unavailable due to the upgrade evolution. All these make service composition a complex task. Along with the fact that users tend to propose different constraints on the QoS of the composite service. For example, users may expect the response time of a composite service to be less than a certain threshold while the execution cost of the composite service must fall within a budget. However, in general there is a tradeoff between cost and response time (or other QoS attributes) for a composite service. The constraints from user further increase the difficulties of service composition. Given the above consideration, it is important and challenging to design a constraint-satisfied service composition (CSSC) method adaptive to the uncertainty QoS and service behavior [3].

So far, many research efforts have been devoted to solving the CSSC problem [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. All these methods regard the QoS values as being determined, and they divide service composition into component services selection stage and composite service execution stage, which means that the composite service are executed after all the component services are selected. However, as the supporting environment, the Internet is highly dynamic, and service evolutions take place casually, thus the QoS values of the same service invoked by the same user at different times can be very different. Due to the uncertainty of QoS and service behavior, it is difficult to guarantee that the optimal component service in the selection stage is still optimal during the execution stage. Even worse, the selected optimal component service may become unavailable during the execution of composite service. Thus, the optimization process has to be performed again, however, the re-optimization cannot guarantee the successful execution of the new optimal composite service.

The goal of CSSC is to find and execute the most appropriate service for each task in the business workflow, so far as to satisfy the functional requirements and QoS constraints as possible as it can be. Taken into account, service behavior can be uncertain. Consequently, this implies variable QoS values and results to difficulties in attempting to satisfy user constraints optimally and robustly. The CSSC process can be considered as an optimization problem of multi-stage decisions within an uncertain decision-making environment.

• *Lifang Ren, Wenjian Wang and Hang Xu are with School of Computer and Information Technology, Shanxi University, Taiyuan 030006, PR China, and Lifang Ren is also with School of Applied Mathematics, Shanxi University of Finance and Economics, Taiyuan 030006, PR China. E-mail: renlf@sxufe.edu.cn; wjwang@sxu.edu.cn; xuh102@126.com.*

Manuscript received M D 2016; revised.

The Markov decision process (MDP), as a model of the reinforcement learning, is a theoretical tool for studying the optimization problem of the multi-stage decision process in stochastic environment [15], so it is especially suitable for solving the CSSC problem. Accordingly, this paper models the CSSC problem as an MDP, namely CSSC-MDP, and designs a Q-learning algorithm to solve the model. By this way, CSSC-MDP integrates the advantages of global optimization approaches and local optimization approaches. The main benefits of CSSC-MDP are as follows:

- CSSC-MDP selects component services during the composite service execution, thus it avoids the failure of whole composite service caused by a component service failure. Therefore, CSSC-MDP is robust.
- CSSC-MDP selects the optimal candidate service for a task based on the constraints which need to be satisfied by the following services. Hence, CSSC-MDP is self-adaptive to the uncertain QoS values.
- The selection strategy of CSSC-MDP aims at maximizing the expected cumulative reward which is on behalf of the satisfaction degree of the user constraints. So, CSSC-MDP is globally optimized.

The remainder of this paper is organized as follows. Section 2 gives the details of our CSSC-MDP approach. In section 3, an illustrative example is presented to explain concretely the process of CSSC-MDP method. Section 4 reports and analyzes our experimental results. Section 5 overviews the related work. Finally, conclusions are given in section 6.

2 THE CSSC-MDP

In this section, the CSSC problem is formally described at first; secondly, the approach of this paper is presented; thirdly, some relevant definitions are formalized; next, the CSSC is modeled as an MDP; then, the decision criterion of CSSC-MDP is presented; finally, the Q-learning algorithm to solve the CSSC-MDP is proposed.

2.1 Problem Description

The aim of CSSC is to find the appropriate component services which can be integrated as an optimal composite service, so as to best meet the user's functional and nonfunctional requirements. In this paper, the functional requirements are described as a workflow. Generally, the workflow can be sequential, conditional, parallel and iterative constructs, or more generically, the combination of these structures. However, in fact, service compositions which involve loops, branches or parallel structures can be converted into sequence structures [16]. Moreover, many studies have been done to compute the QoS of composite service in different structures [12], [13]. Therefore, this paper focuses on the CSSC problem with sequential workflow model. Hence, the process of service composition is to select the component services most suitable for each task in the workflow to form the optimal composite service, meanwhile, all the QoS constraints should be satisfied.

Fig. 1 is the schematic of CSSC. At first, the request for service composition from the user is submitted, it includes the functional requirements and QoS constraints. According to functional requirements, a workflow is built, which is the abstract representation of the composite service. For example, in Fig. 1 the workflow is formed with three sequentially executed abstract tasks (denoted as t_1, t_2 and t_3). In the Internet, there exist many service providers, each of them provides some different services. In Fig. 1, a cloud represents a service provider, different shapes within it represent different services it can provide; and the same shape indicates the same functionality. After service discovery, services with the same functionality are gathered into a candidate service set, such as $A(1)$, $A(2)$, and $A(3)$ in Fig. 1. Thus, each $A(i)$ is the alternative services collection for the abstract task t_i. Then, according to the QoS constraints and the historical execution QoS records of candidate services, one service is selected from each $A(i)$ to form the composite service which can satisfy both the functional and the QoS requirements.

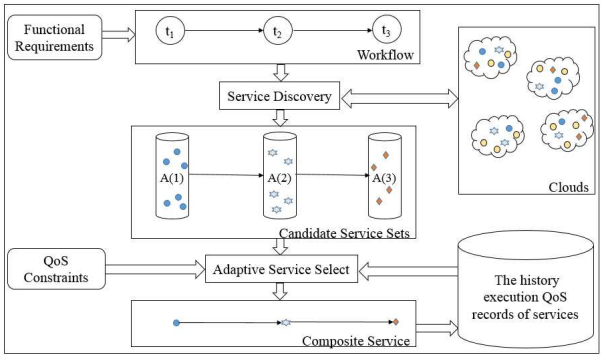

Fig. 1: The schematic of CSSC.

Our work in this paper is to find the appropriate service for each task in the workflow (which meets the user's functional requirements) to form an optimal composite service, which satisfies the user's QoS constraints as well as possible. The main difficulties lie in:

- There are contradictions among QoS attributes. Such as a service with shorter response time may have a higher price, and a lower price may as a result of a lower reliability.
- Because the Internet is dynamic, the QoS values of service are undetermined, which leads to the uncertainty of QoS constraints satisfaction.
- There exists the possibility that a component service fails during the execution of composite service. The failure of component service results in the failure of service composition.

This paper studies how these challenges are dealt with.

2.2 Approach Overview

The approach of CSSC-MDP can be described as Algorithm 1. At first, according to user's functional requirements, the workflow template that represents the abstract service composition are built. Then, the available services for each abstract task in the workflow are discovered and collected. Next, the historical execution QoS records for all candidate services are gathered. There are many works studying the construction of workflow [17] [18] and service discovery [19] [20], so this paper does not discuss the implement

Algorithm 1 The CSSC-MDP

Input:
The function requirements from the user;
The QoS constraints from the user;
Output:
The execution of composite service and QoS records;
Or the negotiation failure information;
1: Build the workflow template;
2: Discover the available services for each abstract task and gather the historical execution QoS records for them.
3: Evaluate the rationality of user constraints.
4: **if** the constraints are not rational **then**
5: Negotiate with the user.
6: **if** No mutually approved constraints **then**
7: Output: Inappropriate QoS constraints, and Exit.
8: **end if**
9: **end if**
10: Build the service composition CSSC-MDP model;
11: Use the Q leaning algorithm to solve the CSSC-MDP model;
12: Send the observed QoS values to the historical records.

of these two steps. And after that, the rationality of user constraints will be evaluated; the evaluation in this paper is based on the 3σ principle; for details, please see Section 3.1. If the constraints are rational, continue to the next step; otherwise, negotiate with the user. If the mutually approved constraints are obtained, proceed to the next step; otherwise, abort. After the mutually approved constraints have been obtained, build the service composition CSSC-MDP model; for details see Section 2.4. Then, the Q-learning algorithm are used to solve the CSSC-MDP; for details, see Section 2.6. Based on the learning result, i.e. matrix Q, the component services are selected and executed until all the tasks in the workflow have been completed; Finally, the observed QoS values are sent to the historical records.

In so doing, some benefits include:

- CSSC-MDP avoids the blind service composition where the rationality of user constraints is not taken into account;
- CSSC-MDP can self-adaptive to the variable QoS and undetermined service behaviors;
- It is effective and efficient to solve CSSC-MDP with the Q-learning.

2.3 Definitions Formalization

In this section, we will give the formal description of some definitions relevant to CSSC-MDP.

Definition 1. (Service) A service is a functionally complete and self-governed resource that can be published, located, and visited through the Web. A service can be formalized as a 4-tuple $(\ ID,\ \ FunC,\ \ QoSE,\ \ QoSR\ )$, where:
ID is the identifier of a service. A service can be uniquely determined by its ID.
$FunC$ represents the functional class of a service, and it is a triple (F, I, O), where F is the functional description of the service, I represents the input items of service and O represents the output items of the service.
$QoSE$ is the expected QoS values offered by the service provider in service description, and it can be formalized as a vector $(v^{(1)}, v^{(2)}, \cdots, v^{(d)})$, where d is the number of QoS attributes which we are concerned with.
$QoSR$ is a container of the historical execution QoS records of the service. It is formalized as a $d \times l$ matrix, where l is the historical execution number of the service. The kth row of matrix $QoSR$ is a vector $(\ v_1^{(k)},\ \ v_2^{(k)},\ \ \ldots, v_l^{(k)}\ )$ that contains l historical recorded values of the kth QoS attribute. The hth column of $QoSR$ is a vector $(\ v_h^{(1)},\ \ v_h^{(2)},\ \ \ldots,\ \ v_h^{(d)}\ )$ that contains all the d QoS attributes' values of the hth execution of the service.
For simplicity, in this paper, we give special focus to two commonly used QoS attributes, the response time and the price, but our approach is equally valid for other QoS attributes.

Definition 2. (Candidate service set) The candidate service set is a collection of alternative services which provide the same functionality but differ in QoS. In other words, services in such a set have the same $FunC$ but differ in QoS. A candidate service set can be formalized as a set $\{ws_1, ws_2, \ldots, ws_m\}$, where m is the number of services with the desired $FunC$, and ws_is (i=1, 2, ..., m) are the IDs of candidate services.

Definition 3. (Workflow) Workflow is an abstract description of the business rules. A sequential execution workflow can be formalized as a sequence $(t_1, t_2, \ldots, t_n)$, where $t_i (i = 1, 2, \ldots, n)$ is the ith abstract task, and n is the total number of tasks.

The power of the services lies in that it can be dynamically integrated to execute a new and more complex task. This process is known as *service composition*.

Definition 4. (Composite service) In order to meet some complex functional requirements, according to certain business logic, services with different functions are integrated into a scalable, loosely coupled, value-added application, namely *composite service*. A composite service can be formalized as a sequence $(ws_1, ws_2, \ldots, ws_n)$, where $ws_i s\ (i = 1, 2, \ldots, n)$ are sequently the IDs of services which compose the composite service and n is the number of services that compose the composite service.

Definition 5. (Constraint) The constraint is usually a QoS requirement about the composite service from the user, such as maximum total price, minimum overall throughput, average response time, etc. A constraint can be expressed in terms of the upper or lower bound of the composite service QoS attribute. Hence, a constraint can be expressed as a triple $(\ att\ \ opr\ \ bnd\ )$, where att is one of the QoS attributes of the user's concern; opr represents relational operators such as $>, <, \geq, \leq$, etc.; and bnd is the bound. For example, $(price < 1000)$ represents the user requires the total cost of the composite service is less than 1000 monetary unit. In general, user may impose more than one constraint to the composite service, thus the constraints form a constraint set.

2.4 CSSC-MDP Model

Suppose the QoS constraints from user are considered to be rational; then, based on the MDP model, the constraint-satisfied service composition model CSSC-MDP can be formalized as a 5-tuple $(\ t,\ \ S,\ \ A,\ \ T,\ \ R\ )$, where:

t is the stage of decision-making; that is, it is the numeric sequence of the task which is being executed. Hence, its possible values are from 1 to the total number of tasks n.

S is the execution state set of the composition service. Consulting literature [21] about the service classifications, we set the four-level-state denoted as $S = \{1, 2, 3, 4\}$. Level 1 represents the user's requirements are excellently satisfied; level 2 means that the composite service can satisfy the user's requirements well; level 3 means that the composite service can basically meet the user's requirements; level 4 indicates the failure of service execution or the violation of user constraints. The number of levels may be determined according to the actual requirements, the details of state division in this paper is given in Section 3.2.

For each concerning attribute, we need to determine the state of composite service. In order to obtain a better performance, we use the worst state of all the QoS attributes as the integrated state, i.e., $s_i = \max\{s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(d)}\}$, where s_i is the state of the service composition after task t_i is finished.

$A = \bigcup_{i=1}^{n} A(t_i)$ is the set of all the candidate services in the model, where $A(t_i) = \{ws_{i1}, ws_{i2}, \ldots, ws_{ik_i}\}$ is the candidate service set for task t_i, and k_i is the total number of services in t_i's candidate set. A is the union of all the $A(t_i)$s; in other words, A contains all the services that may be used in the composite service.

$T(s, ws, s') = Pr(s_{t+1} = s' | s_t = s, a_{t_i} = ws)$ is the probability that the execution of service ws at state s for task t will lead to state s'. This can be calculated according to the historical execution QoS records for service ws.

$R(s', s, ws)$ is the reward function. it is a real-valued function, and $r = R(s', s, ws)$ is the immediate reward received from a state transition from s to s' after ws is executed. We can say that it is used to quantify the benefit of executing service ws, which leads to the state transition. When $r > 0$, it indicates rewards; when $r < 0$, it indicates penalties. After the execution of ws, the greater the difference of $s - s'$, the higher the value of r, and vice versa. The goal of service composition is to select the optimal services to compose the composite service with the highest cumulative reward. The reward function can be defined according to the actual situation, the definition of $R(s', s, ws)$ in this paper can be found in Section 3.3.

In contrast with existing methods, CSSC-MDP considers the state of constraints been satisfied after each service execution. In case of the violation of QoS constraints or the service failure, the stage t remains its value and CSSC-MDP will redo the selection and execution. Otherwise, let $t = t + 1$, and select the service which can lead to an optimal composite service and execute it for the next task. The process of decision-making is described in detail in the following section.

2.5 Decision-Making of CSSC-MDP

The core problem of an MDP is to determine the optimal policy π^* that will maximize the cumulative function of the rewards. The function $\pi(s)$ specifies the action that the agent will choose in state s. For the CSSC-MDP model, the optimal service of a task is selected based on the constraints state and the historical execution QoS records of candidate services.

Fig. 2 shows the decision-making process in CSSC-MDP. At the initial state s_0, according to the historical execution QoS recorded in H_1 for services in $A(1)$, for task t_1 the agent selects and executes the optimal service denoted as ws_1, the constraints state transfers to a new state, denoted as s_1, and the execution QoS data are then appended to the historical records for ws_1 in H_1. Based on the new constraints state s_1, the immediate rewards r_1 can be calculated. According to the historical execution QoS recorded in H_2 for services in $A(2)$, for task t_2 the agent selects and executes the optimal service at state s_2, and so forth, until all the tasks in the business workflow have been done.

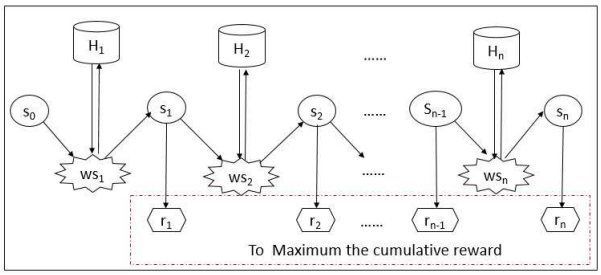

Fig. 2: The decision-making process in CSSC-MDP.

The strategy of CSSC-MDP is a mapping from state space to service space. To determine the optimal strategy that leads to the optimal composite service, we need to define the value function $V_t(s)$ to calculate the expected cumulative rewards of the service execution for task t at state s. The value function $V_t(s)$ is defined by the recursion formula $V_t(s) = \sum_{s'} T(s, ws, s')\big(R(s, s', ws) + \gamma V_{t+1}(s')\big)$, where s' is one of the possible constraint states after the execution of the service ws; $T(s, ws, s')$ is the transition probability from state s to s' caused by service ws; $R(s, s', ws)$ is the immediate reward of the execution of service ws; and γ is the discount factor for future rewards. Consequently, $V_t(s)$ is the cumulative reward resulting from the execution of service ws at task t in state s.

The strategy that leads to the highest value of $V_t(s)$ is the optimal strategy, which can be expressed as $\Pi_t(s) = \arg\max_{ws \in A(t)} V_t(s)$.

Under the guidance of the optimal strategy, the decision-making process of CSSC-MDP maximizes the exceptional cumulative rewards.

CSSC-MDP select a service for a task after the previous task is completed, hence it can update the state according to the observed QoS values of services which have been completed. That is to say, CSSC-MDP can more realistically reflect the QoS constraints which need the following service to satisfy, and can provide a reliable guarantee for the next service selection to meet the user's constraints. This process continues until all the tasks have been accomplished in turn.

In theory, after modeling the CSSC problem as an MDP, the model can select the optimal strategy dynamically, so the service composition satisfies the user with real-time globally optimal results. However, the computational time complexity of the exact solution of an MDP problem has been proved to be completely P [22]. Hence, what follows is a Q-learning algorithm to solve the CSSC-MDP model.

2.6 CSSC-MDP Solving

In practice, the existing exact methods to solve the MDP can only accurately solve small problems [15]. Q-learning is a model-free reinforcement learning method that uses rewards to reinforce actions which lead the model to a better state; hence, it can help to learn the optimal policy in a stochastic environment [16]. Therefore, we use Q-learning to solve the CSSC-MDP model. In Q-learning, the transition probabilities are generally obtained through many simulations. However, because services are mostly commercial, it is unlikely to invoke services to test their performance. Hence, in this paper we learn the transition probabilities of service from their historical execution records. Algorithm 2 is the Q-learning algorithm for CSSC-MDP.

As shown in Algorithm 2, we first initialize Q as a matrix with $3 \times n$ rows and m columns whose elements are all zeros, where n is the total number of sequential tasks, and m is the maximal size of the candidate service sets. For every service, we need to consider its performance at a different state; hence, each row of matrix Q represents the scores of a candidate service at one state. For example, the 5th row of matrix Q represents the scores of the candidate services of task t_2 at state 2. Because state 4 represents service failure or constraint violation, that is, the service composition cannot proceed from state level 4, in the matrix Q, we only consider the performance scores of services in the first 3 states.

Next, an iteration process is performed to learn the matrix Q. At the first iteration, we initialize the task sequence number t to 1, set the initial state s_0 to 1 (we assume that service compositions always start from the best state), and use the original constraints C_0 to initialize the current constraints C. Then for each task from the first to the last, we use an ε-greed strategy to select one service ws from its candidate service set $A(t)$, i.e. when the random number is smaller than ε, the greed strategy is adopted, otherwise, randomly selects a service. On the one hand the ε-greed can prevent the algorithm from premature convergence, on the other hand it gives chances to the newcomer in service selection. For the selected service ws, if it has been invoked, we randomly choose one piece of the QoS record h from the execution historical records of service ws and consider h as the currently executing QoS. We don't choose the best one, the worst one, the average one or the latest one, but choose the random one piece of QoS record is to simulate the various possible behaviors of service. This is because we want to learn the most likely performance of service from all random behaviors of services. Otherwise if ws has no QoS record, we take the expected QoS $ws.QoSE$ given by service provider as its QoS record h. Based on the QoS information of historical record h and the current constraints C, we calculate the constraint-satisfied state s as described in detail in the section 3.2. According to the new state s and the old state s_0, we calculate the immediate rewards r. Details of the method can also be found in the Section 2.4. If the selected service ws executes successfully and the user constraints are not violated, we update the element of Q at the tth row and the wsth column, where we use α $(0 \leq \alpha \leq 1)$ as the learning rate; that is, the new $Q(t, ws)$ is composed of $(1-\alpha)$ original $Q(t, ws)$ and α new rewards. The new rewards include the immediate reward and the expected rewards in the future. The γ in the formula is the discount factor, means that the rewards in the future are not necessarily as important as the immediate rewards, and $\gamma \in [0, 1]$ can be inferred. Then, we update the current constraints based on the QoS values of the selected QoS record h. For example, the response time constraint will be reduced by the response time of h. After all these steps, we update the task sequence number t and use the current state s to update the original state s_0. In the circumstances, either service failure or constraint violation, i.e., at the state $s = 4$, the immediate reward turns into a penalty. And because the composition is interrupted, the expected rewards in the future are not included in the $Q(t, ws)$. Also for the same

Algorithm 2 Q-learning algorithm for CSSC-MDP

Input:
The total number of sequential tasks n;
The maximal size of the candidate service sets m;
The user constraints C_0;
The historical execution QoS records for all services H;
Output:
The results of Q-learning, i.e., matrix Q;

```
Initial Q ← zeros(3 × n, m);
while Q is not convergent do
  Initial variables: t ← 1, s_0 ← 1, C ← C_0;
  while t ≤ n do
    Use the ε-greed policy to choose ws ∈ A(t) based on s_0;
    if ws.QoSR ≠ Φ then
      Randomly choose an execution historical QoS record h of service ws;
    else
      h = ws.QoSE
    end if
    Based on h and the current constraints C, calculate the state s;
    Based on s and s_0, calculate the immediate rewards r;
    if s < 4 then
      Q(3(t−1)+s_0, ws) ← (1−α)Q(3(t−1)+s_0, ws)+ α[r + γ max_{ws'∈A(t+1)} Q(3t + s_0, ws')];
      According to h, update C;
      t ← t + 1; s_0 ← s;
    else
      Q(3(t−1)+s_0, ws) ← (1−α)Q(3(t−1)+s_0, ws)+ αr;
      t ← 1; s_0 ← 1; C ← C_0;
    end if
  end while
end while
return Q;
```

reason, t, s_0, and C are returned to their initial values. The iteration does not terminate until matrix Q is convergent.

Q-learning algorithm learns the ability to satisfy the constraints for every service at every state. Every element of matrix Q is the score of this ability. Based on the matrix Q and the user constraints, service composition can be implemented. The main benefits of Q-learning include:

- The Q-learning algorithm can learn the ability of services to satisfy the constraints from the historical QoS records, without redundant service execution;
- It doesn't need to define transaction function, which can be implied by the Q-learning process automatically;
- The ε-greedy selection strategy in the Q-learning not only makes the algorithm more efficient but also guarantees every service has the opportunity to be executed.

3 An Illustrative Example

This section illustrates the approach of CSSC-MDP through an example. Consider the following CSSC scenario: a user captured a set of multi-angle photos of an object in motion, and he wants to use this set of images to reconstruct the 3D model of the object. Due to limited local resources, the user wants to complete the task by means of Web services provided over the Internet. moreover, he also expects that the total response time is less than 2000 seconds and the total price is less than 300 yuan. To meet the user's requirements, we composite the available Web services. Since the images for this moving object are blurred, we first need the image restoration service to restore the blurred 2D images. Then we utilize the computing power provided by Web services to run the 3D reconstruction algorithm to obtain the 3D model of the object. Finally, we store the 3D model in the cloud space for the user. Meanwhile, we should ensure that the user's QoS constraints are best satisfied. Fig. 3 shows the process of service composition in this scenario.

Obviously, the user's functional requirements can be decomposed into three sequential tasks: image restoration task t_1, the 3D reconstruct task t_2 and the cloud storage task t_3. And the composite service should satisfy the following constraints:

$$(ResponseTime \leq 2000) \text{ and } (Price \leq 300). \quad (1)$$

It is a CSSC problem. We need to discover services for each task at first. In the service computing environment, a service provider can provide services with different functions and services with the same function can be provided by different providers. Suppose, after service discovery, as shown in Fig. 3, there are 4 candidate services for the first task t_1 denoted as $A(1) = \{ws_{11}, ws_{12}, ws_{13}, ws_{14}\}$, 3 candidate services for the second task t_2 denoted as $A(2) = \{ws_{21}, ws_{22}, ws_{23}\}$, and 4 candidate services for the third task t_3 denoted as $A(3) = \{ws_{31}, ws_{32}, ws_{33}, ws_{34}\}$. And let's suppose the prices of each service in $A(1)$ are $(82, 90, 70, 85)$, in $A(2)$ are $(140, 230, 180)$, and in $A(3)$ are $(20, 30, 25, 22)$. The historical response time records for each service in $A(1)$ are stored in the following matrix. Values in the kth row of the matrix represent the response time of service ws_{1k} for different execution histories, where -1 means service failure and NaN represents the absence of record.

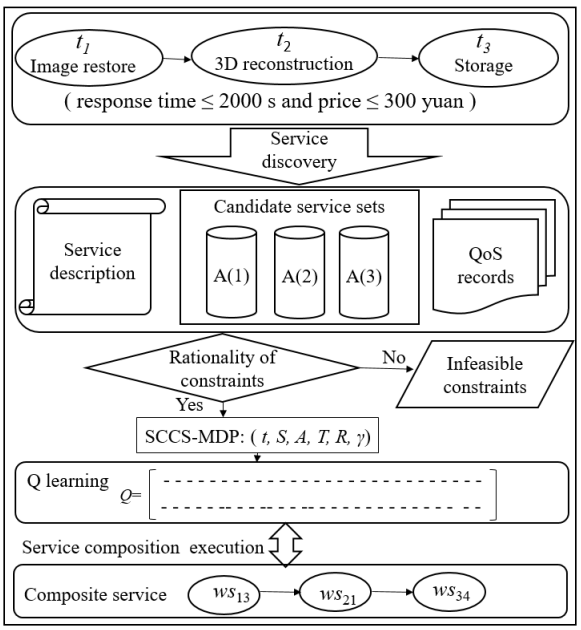

Fig. 3: The process of CSSC-MDP.

$$\begin{pmatrix} 417 & -1 & 429 & 416 & 421 & -1 & 414 & 430 & -1 & 421 \\ 320 & 289 & 307 & 322 & 286 & 295 & 313 & 317 & 321 & 296 \\ 390 & 362 & 371 & 380 & 310 & 373 & 367 & -1 & NaN & NaN \\ 434 & 405 & 421 & -1 & 400 & 403 & 413 & NaN & NaN & NaN \end{pmatrix}$$

The historical response time records for each service in $A(2)$ are stored in the following matrix.

$$\begin{pmatrix} 1417 & 1249 & -1 & 1641 & 1521 & -1 & NaN & NaN & NaN & NaN \\ 1201 & 1277 & 1307 & 1021 & 1086 & 1395 & 1313 & 1417 & 1121 & 996 \\ 1450 & 1242 & 1431 & -1 & 1240 & 1433 & 1627 & -1 & NaN & NaN \end{pmatrix}$$

The historical response time records for each service in $A(3)$ are stored in the following matrix.

$$\begin{pmatrix} 173 & 133 & 121 & 146 & -1 & -1 & 119 & 128 & -1 & NaN \\ 90 & 109 & 97 & 112 & 86 & 95 & 113 & 107 & 111 & 106 \\ 112 & 132 & 101 & 130 & -1 & 122 & 109 & 113 & NaN & NaN \\ 174 & 153 & 124 & -1 & 140 & 133 & 113 & NaN & NaN & NaN \end{pmatrix}$$

After service discovery, our approach can be implemented.

3.1 Constraint Rationality Evaluation

First of all, we will evaluate the rationality of user constraints. Based on the historical records, the means and standard variances of both price and response time for each task can be calculated. In the above example, $\mu^{(c)} = (81.75, 216.67, 24.25)$ and $\sigma^{(c)} = (8.50, 70.95, 4.35)$ are means and standard variances of price for each task. $\mu^{(r)} = (368.10, 1319.30, 120.76)$ and $\sigma^{(r)} = (51.30, 181.60, 21.61)$ are means and standard variances of response time for each task. Hence, the mean of price for the composite service is $\mu_{cs}^{(c)} = \sum_{i=1}^{3} \mu_i^{(c)} = 322.67$, and the standard variances of price for the composite service is $\sigma_{cs}^{(c)} = \sqrt{\sum_{i=1}^{3} (\sigma_i^{(c)})^2} = 71.59$, the mean of response time for the composite service is $\mu_{cs}^{(r)} = \sum_{i=1}^{3} \mu_i^{(r)} = 1808.16$, and the standard variances of response time for the composite service is $\sigma_{cs}^{(r)} =$

$\sqrt{\sum_{i=1}^{3}(\sigma_i^{(r)})^2} = 189.94$. Since $\mu_{cs}^{(c)} - 2\sigma_{cs}^{(c)} < 300 < \mu_{cs}^{(c)} + 2\sigma_{cs}^{(c)}$ and $\mu_{cs}^{(r)} - 2\sigma_{cs}^{(r)} < 2000 < \mu_{cs}^{(r)} + 2\sigma_{cs}^{(r)}$, according to the 3σ principle of normal distribution, the constraints can be satisfied with a probability of 0.9544. Hence, we have reason to believe that the user constraints are rational.

3.2 State Level Division

To set the levels for services, at first, we determine the median QoS attribution value of every service based on its execution historical QoS records (not include the failure records). Such as, for task t_1, the response time medians of candidate service are $v_1^{(r)} = (421, 307, 371, 408)$, and the mean response time of task t_i can be calculated by $\mu_i^{(r)} = \frac{1}{m}\sum_{j=1}^{m} v_i^{(r)}(j)$. Where m is the total number of candidate services for task i. In this way, we can calculate the mean response time for tasks t_1, t_2 and t_3, $\mu_1^{(r)} = 376.75$, $\mu_2^{(r)} = 1380$ and $\mu_3^{(r)} = 121.625$.

Then we can decompose the user's response time constraints for each task by $C_i^{(r)} = C^{(r)}\frac{\mu_i^{(r)}}{\sum_{i=1}^{n}\mu_i^{(r)}}$, where $C^{(r)}$ represents the user's total constraints on the response time and n is the total number of tasks. Therefore, $C_i^{(r)}$ is the excepted value of the response time constraint for task t_i. The excepted value of the price constraint for task t_i, i.e. $C_i^{(c)}$, can be calculated in the same way. Note that the decomposition of constraints is only for service level setting, and during the selection of component service we take the constraints of composite service as a whole.

Here, taking the response time constraint as an example, we formulate the four state levels as formula (2), where $v_{ij}^{(r)}$ is the response time of service ws_{ij}. If $v_{ij}^{(r)}$ is less than or equal to $0.85C_i^{(r)}$, then the state level is 1; if $v_{ij}^{(r)}$ is more than $0.85C_i^{(r)}$ and less than or equal to $C_i^{(r)}$, then the state level is 2; if $v_{ij}^{(r)}$ is more than $C_i^{(r)}$ and less than or equal to the user's total constraints vector $C^{(r)}$, then the state level is 3; otherwise, if $v_{ij}^{(r)}$ is more than $C^{(r)}$ or the service fails, then the state level is 4. This can be denoted as

$$s_i^{(r)} = \begin{cases} 1 & v_{ij}^{(r)} \le 0.85C_i^{(r)} \\ 2 & 0.85C_i^{(r)} < v_{ij}^{(r)} \le C_i^{(k)} \\ 3 & C_i^{(k)} < v_{ij}^{(k)} \le C^{(k)} \\ 4 & v_{ij}^{(k)} > C^{(k)} \text{ or service failure} \end{cases}, \quad (2)$$

where 0.85 is a parameter to tune the levels. In this paper the four state levels for price are set in the same way. In fact, the number of levels and the scope of each level can be set up according to the actual situation.

3.3 Solution

The Q-learning algorithm is used to study the performances of each candidate service from the historical execution records. In this paper, we set the reward function as $r = 10(s - s') + 10$. This means that the reward is determined by the difference between s and s'. That is, the more the new state s' is superior to the old state s, the greater the reward is, and vice versa. If the new state s' is the same as the old state s, then the running service can obtain a basic reward. When the new state s' is much worse, i.e. s' is larger than s by 2 or more levels, then the reward is a negative value; in this case, the reward becomes a penalty.

Executing Algorithm 2 for the above example, the results of Q-learning are shown in the following matrix:

$$Q = \begin{pmatrix} 4.0219 & 1.1104 & 12.3300 & -0.0280 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1.2607 & 3.2561 & 6.7946 & 0 \\ 11.3940 & -7.0000 & 2.3327 & 0 \\ 15.5230 & 3.0000 & 25.3450 & 0 \\ 4.0790 & 12.8200 & 8.2120 & 8.4224 \\ 17.0120 & 19.1020 & 18.6990 & 21.5350 \\ 3.2118 & 14.6410 & 14.8750 & 9.6686 \end{pmatrix}.$$

The matrix Q indicates the performance scores of each service in different sustainable states (each task has three sustainable states, i.e. s=1, 2 or 3) in terms of the services' adaptability to different constraint situations. For example, the 5th row of matrix Q shows the performance scores of services in A(2) at state $s = 2$. From the data in this row, we can infer that service ws_{21} is the best choice for task t_2 at state $s = 2$. It is reasonable to suppose that the service composition always begins from a good start, so the initial state is always $s = 1$; thus, the elements of rows 2 and 3 in matrix Q are always zeros. Furthermore, because task t_2 has 3 candidate services, the elements of the 4th column at rows 4, 5, and 6 are all zeros. During service composition, matrix Q is the basis of service selection under different constraint satisfaction states.

After Q-learning, service composition can be carried out based on the matrix Q. Table 1 shows an execution of the service composition, from Table 1 we can see the selected service for each task, its actual QoS values in the execution, and the new constraints state after the execution. Due to the fact that service compositions always start from a good state, the service with the highest score in the 1st row of the matrix Q, i.e. ws_{13}, is selected for task t_1. Executing service ws_{13}, the state becomes $s = 2$. Hence, for task t_2, the best service is sought in state $s = 2$; that is, the service with the highest score in the 5th row, i.e. ws_{21}, is selected. After the execution of service ws_{21}, the state is still $s = 2$. So, based on the data of the 8th rows in matrix Q, service ws_{34} is selected and executed, and the state becomes $s = 1$. At this point, the service composition for the example is executed successfully. It is important to note that the QoS data should be recorded in the historical execution QoS records after the execution of each service.

TABLE 1: An execution of the service composition.

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	380	70	2
2	ws_{21}	1249	140	2
3	ws_{34}	174	22	1

As can be seen from Table 1, CSSC-MDP always looks for the best service in the current state of the matrix Q to execute. However, because of the dynamic nature of the Internet environment, the performance of service is uncertain. The following sections show the adaptivity of CSSC-MDP to uncertain service behaviors.

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/TSC.2017.2727050, IEEE Transactions on Services Computing

3.4 Adaptivity to Variable QoS

Dynamic environment leads to variable QoS values. Comparing Table 1 and Table 2. In the running of Table 2, service ws_{21} takes a much longer response time in Table 2 than in Table 1, and the state becomes $s = 3$. Therefore, based on the last row of the matrix Q, service ws_{33} are selected and executed.

TABLE 2: Service composition when QoS changes.

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	310	70	2
2	ws_{21}	1521	140	3
3	ws_{33}	101	25	1

From the historical execution records we will find the service ws_{33} usually has a shorter response time and a bit higher price. The results show the self-adaptivity of the CSSC-MDP to the variable QoS.

3.5 Adaptivity to Uncertain Service Behaviors

Because of network fault or service evolution, sometimes service failures may occur. Such as the running of the service composition in Table 3, service ws_{13}, the best candidate service for task t_1, encounters problems. Based on the matrix Q, service ws_{11} is the next-best service in the candidate service set A(1); therefore, service ws_{11} is selected as the alternative to service ws_{13}. After the execution of service ws_{11}, the state becomes $s = 2$. The best candidate service for t_2 in this state, with the maximum value in the 6th row of matrix Q, i.e. ws_{23}, is selected and executed, and the service composition can proceed. It shows when service failure occurrence CSSC-MDP can find a suitable alternative and compliment the service composition.

TABLE 3: Service composition when service fails.

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	-1	70	4
1	ws_{11}	417	82	3
2	ws_{23}	1240	180	3
3	ws_{33}	109	25	2

If the selected alternative service has not fulfilled task t_1 yet, the situation is as shown in Table 4. When service ws_{13}, following ws_{11}, also encounters failure, service ws_{12} is the best choice in the current situation. Therefore, service ws_{12} is executed to fulfill task t_1, and the service composition is able to continue.

TABLE 4: Service composition when continues services fail.

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	-1	20	4
1	ws_{11}	-1	20	4
1	ws_{12}	289	95	3
2	ws_{23}	1240	180	3
3	ws_{33}	130	25	3

In fact, as long as there exist qualified services for each task, service composition can be successfully completed. This shows the robustness of the CSSC-MDP.

3.6 Adaptivity to Different Constraints

Different users have different QoS constraints on the same functional requirement. Suppose, in this example the constraints are changed to

$$(responsetime \leq 1830) \text{ and } (price \leq 330). \quad (3)$$

Compared with the QoS constraints (1), the constraint on price is looser and the constraint on response time is tighter. In this case, the results of Q-learning are shown in the following matrix Q_1.

$$Q_1 = \begin{pmatrix} -4.1436 & -7 & 11.941 & 2.9984 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -6.0003 & -7 & -13.654 & 0 \\ 6.1325 & 21.39 & 5.3209 & 0 \\ 19.652 & 22.403 & 20.629 & 0 \\ 0 & 0 & 0 & 0 \\ 16.943 & 22.996 & 17.753 & 12.674 \\ 9.8305 & 13.53 & 28.287 & 9.5256 \end{pmatrix}.$$

Comparing the 5th row of matrix Q_1 and matrix Q, we can see that service ws_{22} is the worst service for task t_2 in state $s = 2$ in Q, but in Q_1 service ws_{22} is the best service for task t_2 in the same state. This shows that service ws_{22} is more adaptive in circumstance of shorter response time and higher price. This is in accordance with the fact that service ws_{22} always has a lower mean response time and a higher price, which can be seen from the historical execution records. Based on matrix Q_1, the best composite service $(ws_{13}, ws_{22}, ws_{33})$ for the user constraints is performed successfully, as shown in Table 5.

TABLE 5: Service composition with constraint (3).

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	380	70	2
2	ws_{22}	1021	230	3
3	ws_{33}	132	25	2

If the QoS constraints of the composite service are changed to

$$(responsetime \leq 2700) \text{ and } (price \leq 255). \quad (4)$$

Compared with the constraints (1) and (3), the constraint on price is tighter and the constraint on response time is looser. In this case, the results of Q-learning are shown in the following matrix Q_2.

$$Q_2 = \begin{pmatrix} -10.2390 & -6.9993 & 2.7579 & -10.8440 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 19.8790 & -6.9322 & 11.5320 & 0 \\ 14.6440 & 2.9951 & 2.9988 & 0 \\ 0 & 0 & 0 & 0 \\ 20.1350 & 7.8807 & 11.448 & 7.8207 \\ 2.8271 & 2.7529 & 2.2797 & 1.5300 \end{pmatrix}.$$

Focusing on task t_3 in the state $s = 3$, the best service is service ws_{33} in Q; however, in Q_2 service ws_{31} is the best.

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/TSC.2017.2727050, IEEE Transactions on Services Computing

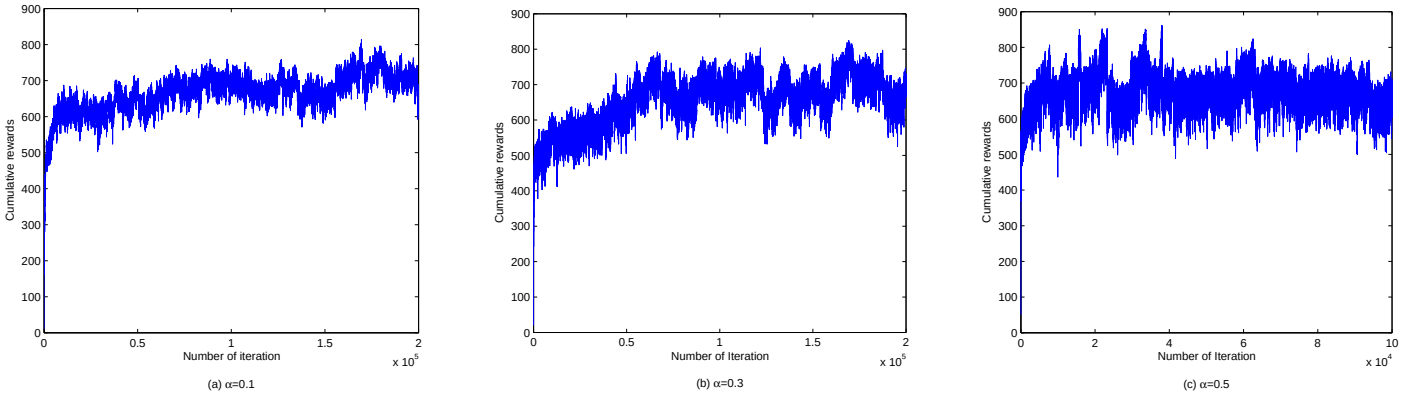

Fig. 4: Efficiency of Q-learning with different learning rates.

This is because service ws_{31} is commonly lower in price and higher in response time than service ws_{33} is, so ws_{31} is more adaptive to this circumstance. Hence, the running of composite service in this case is as shown in Table 6.

TABLE 6: Service composition with constraint (4).

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	373	70	3
2	ws_{21}	1641	140	3
3	ws_{31}	146	20	1

The results in Table 5 and Table 6 show the adaptivity of CSSC-MDP to different user constraints.

4 Experimental Results and Discussion

To test the performance of CSSC-MDP on a larger scale, the following experiments simulates sequential service compositions which contain 20 tasks, and 50 candidate services per task. For each service, 100 historical response time records and a price data was artificially synthesized. The data set was constructed as follows: randomly generate a 1×20 vector $\boldsymbol{tpm}$ whose elements are between $[100, 1000]$; and randomly generate a 1×20 vector $\boldsymbol{tpv}$ whose elements are between $[10, 30]$; take tpm_i as the mean and tpv_i as the variance, randomly generate a 50×1 normal distribution data sp_i; $\boldsymbol{sp} = (sp_1, sp_2, \cdots, sp_{20})$ are 50×20 price data for the 20 tasks' 50 candidate services; let $\boldsymbol{trm} = [0.25 \times \boldsymbol{tpm}] \times 10$ are the mean response time of each task, the function is set to guarantee that for different tasks the longer average response time the higher price; randomly generate a 1×20 vector trv,whose elements are between $[100, 1000]$; take trm_i as the mean and trv_i as the variance, randomly generate 50×1 normal distribution data $\boldsymbol{srm_i}$ as the mean response time of 50 candidate services for task t_i; sort the elements in $\boldsymbol{srm_i}$, let a higher price corresponds to a shorter mean response time, it is in line with the actual situation that for the same task the shorter response time the higher price; randomly generate a 50×20 vector $\boldsymbol{srv}$ whose elements are between $[50, 400]$; take srm_{ij} as the mean and srv_{ij} as the variance, randomly generate a 100×1 normal distribution data $\boldsymbol{srr_{ij}}$, $\boldsymbol{srr}$ are $100 \times 50 \times 20$ historical recorded response time data for the 20 tasks' 50 candidate services 100 records.

The simulations were conducted by using MATLAB R2013a. The experimental platform runs Windows 10 with an Intel Core Quad CPU at a clock speed of 2.67 GHz with 4 GB RAM.

We assume that the service composition system has no knowledge of the QoS performance of all the candidate services. Based on the historical execution QoS records, we let the Q-learning algorithm guide the service composition to reach the optimal policy gradually. In the following experiments, the parameters of Q-learning are determined first; then, the efficiency of Q-learning is studied; finally, comparison experiments with three existing methods are conducted.

4.1 Parameter Tuning

The learning rate and the greedy rate are two important parameters in the Q-learning algorithm. The following experiments show the determination of the two parameters.

4.1.1 Learning Rate

To obtain a faster learning speed and a better learning result, it is necessary to set a proper learning rate. We fix the number of tasks to 5 and the number of candidate services of each task to 10, and vary the learning rate α. Fig. 4 shows the learning efficiency of Q-learning with learning rate $\alpha = 0.1, 0.3, 0.5$. The learning speed here refers to the number of iterations after which the cumulative rewards achieves the maximum. From Fig. 4 we can see, when $\alpha = 0.1$, the cumulative rewards still have an increasing trend after about 100,000 iterations (see Fig. 4 (a)), which indicates the learning speed is low. As demonstrated in Fig. 4 (b) and (c), when $\alpha = 0.3$, the cumulative rewards reach the maximum before 100,000 iterations; when $\alpha = 0.5$, the cumulative rewards reach the maximum before 40,000 iterations; the cumulative rewards reach quickly to a higher value indicates the learning speeds are higher. However, we can also see from Fig. 4 the cumulative rewards fluctuate more and more severely.

It is reasonable that the smaller the learning rate, the slower the learning speed, whereas a higher learning rate results in a severe fluctuation range. Hence, we take a dynamic learning rate, initially set $\alpha = 0.5$, and consider that every iteration has an $\alpha = \alpha - 0.000007$ decay. Fig. 5 shows

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/TSC.2017.2727050, IEEE Transactions on Services Computing

learning efficiency with this dynamic learning rate. From 5, we can see cumulative rewards reach to a higher value after 20,000 iterations, and the fluctuations of cumulative rewards are gradually mild. Hence, we use this dynamic learning rate in the following experiments.

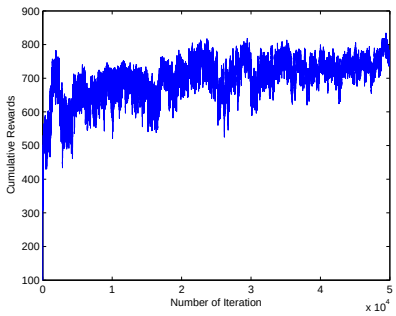

Fig. 5: Q-learning efficiency of a dynamic learning rate.

4.1.2 Greedy Rate

To study the influence of the greedy rate on Q-learning, we vary the parameter ε of ε-greedy algorithm from 0.1 to 0.9. Fig. 6 shows the convergent cumulative rewards and the number of iterations with different ε values.

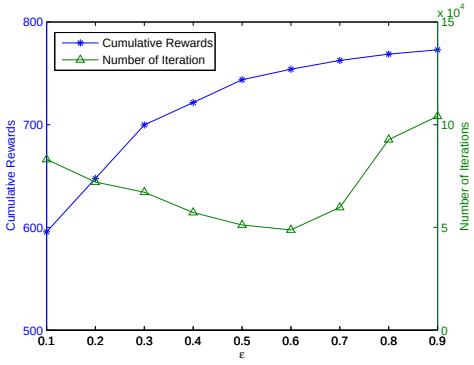

Fig. 6: Q-learning efficiency of different greedy rates.

As demonstrated in Fig. 6, the iteration number in Q-learning is a minimum when the ε-greedy rate is $\varepsilon = 0.6$, and in this case the cumulative reward convergence is near-optimal. Hence, we set the ε-greedy rate of $\varepsilon = 0.6$ in the following experiments.

4.2 Performance Study

In the experiments shown in Fig. 7, we study the performance with respect to the number of tasks and the number of candidate services. The number of tasks varies from 5 to 20, while the number of candidate services of each task varies from 5 to 20. We take moderate constraints for each case in order to ensure that all the 3 different states can be reached in every case. In each case, repeat the experiments 100 times, and record the average learning time.

Fig. 7 illustrates that the time of Q-learning increases obviously with the number of tasks. It is easy to understand, because one more task in the work-flow brings one more

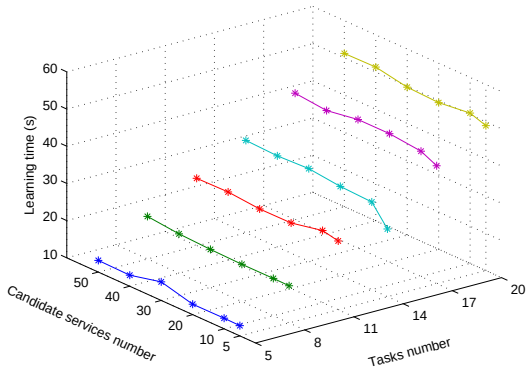

Fig. 7: Performance of Q-learning.

step in each iteration of the Q-learning. Fig. 7 also illustrates that it is not obvious the time of Q-learning increases with the number of candidate service, this is because the time cost of Q-learning depends not only on the number of candidate services but also on the diversity of candidate services.

4.3 Comparison Experiments

For the purpose of performance evaluation, we compare the method proposed in this paper with a classic global optimization method *WS-IP* [5], a local optimization method *Hybrid* proposed in [13] and a MDP-based method [23], for the sake of convenience, we name it *RQASC*.

WS-IP. The purpose of this method is to select the candidate service for each task that satisfies the user's constraint and maximizes a defined utility function. In this method, the CSSC problem is modeled as a 0-1 integer programming (IP) problem and the well-known *lpx-intopt* algorithm is used to find the optimal solution. In [13] and [5], *WS-IP* is regarded as the optimal solution to be compared.

Hybrid. This is a method based on the decomposition of constraints. This method firstly uses a mixed-integer program (MIP) to find the optimal decomposition of the global constraints, and then find the best services that satisfy the local constraints. The purpose of decomposition is to improve the optimization efficiency.

RQASC. Being aware of the undetermined QoS, this method measures the QoS by the mean and variance of random variables, and model the service composition as an MDP. RQASC only selects the optimal composite service, and it care nothing about user's QoS constrains.

To simulate the uncertain environments, 2,326 service failure records were randomly inserted into the response time record data. That is, in our artificially synthesized database, the service failure rate was 0.02326 and we set the constraints as $(Price \leq 7500)$ and $(ResponseTime \leq 20000)$.

Without loss of generality, suppose that the response time and the price are of equal importance; hence, in the WS-IP and hybrid methods, the weight of both response time and price are set to 0.5. In RQASC method, the price and the response time are alternately optimized to obtain the highest success rate. For each method mentioned above, we conduct service composition 100 times. The average success rate of

WS-IP is 54%, the average success rate of Hybrid is 31%, the average success rate of RQASC is 72%, and the average success rate of CSSC-MDP is 100%. From the result we can see the poor adaptivity of WS-IP and Hybrid to dynamic environments. Once one of the selected optimal candidate services fails, the service composition inevitably faces failure. The success rate of RQASC is a little higher. This is because it can guarantee the success of service composition but the QoS constrains are ignored by RQASC. In contrast, CSSC-MDP can perceive the state of service composition and always selects the service best suited to the situation. Moreover, in the case of component service failure, CSSC-MDP can select another qualified candidate service to fulfill the task. Therefore, even if the environment is dynamic, as long as there are qualified candidate services, CSSC-MDP can manage to complement the service composition successfully.

Table 7 shows the response time and the price records recorded from one successful running of the 4 methods. From Table 7, we can see each method has the best performance for response time or price. On the whole, WS-IP is superior in response time, but the price of composition service by WS-IP is the most expensive. While CSSC-MDP performs the best in terms of price, and from the perspective of response time, CSSC-MDP performs the best for 3 tasks and approximates the optimal values for other tasks. This shows that CSSC-MDP can obtain the best tradeoff between the response time and the price.

TABLE 7: Response time and price of a successful execution.

Task	Response Time (s)				Price (Yuan)			
No.	**WS-IP**	**Hybrid**	**RQASC**	**CSSC-MDP**	**WS-IP**	**Hybrid**	**RQASC**	**CSSC-MDP**
1	**498.94**	512.02	545.68	537.40	215.75	178.33	**163.44**	**163.44**
2	**716.22**	735.90	746.12	746.48	271.80	257.33	**231.34**	**231.34**
3	283.61	313.18	**275.60**	315.29	134.59	106.19	145.52	**92.33**
4	**463.90**	476.26	465.49	465.36	178.68	**157.43**	220.64	**157.43**
5	**919.64**	942.00	936.69	927.12	342.36	339.46	**317.70**	**317.70**
6	1518.34	1554.24	**1438.85**	1503.03	579.77	**575.31**	661.16	589.92
7	**566.48**	632.37	585.25	645.41	255.89	223.95	268.44	**202.44**
8	421.91	423.92	423.20	**418.20**	149.72	**137.05**	187.65	137.17
9	**1164.50**	1228.97	1237.91	1171.89	503.66	459.36	**440.37**	493.72
10	389.11	374.05	374.53	**370.64**	113.65	125.53	**107.02**	118.96
11	2004.08	1981.03	**1939.09**	2019.56	772.18	**766.89**	862.04	781.65
12	1735.82	1765.65	1739.19	**1703.49**	643.36	648.06	**626.83**	643.17
13	**468.03**	507.36	510.17	510.25	200.81	175.97	**157.24**	163.95
14	584.97	583.46	**492.46**	564.53	213.48	204.85	267.93	**191.69**
15	**648.18**	653.73	674.88	685.28	239.27	229.67	**207.72**	219.46
16	1636.47	1624.78	**1518.87**	1611.44	626.32	**604.97**	717.74	633.16
17	813.01	839.17	**758.23**	838.11	313.01	301.80	363.65	**277.11**
18	**773.12**	830.81	803.65	815.80	325.43	**276.48**	**276.48**	**276.48**
19	**940.74**	981.64	1043.04	1005.90	418.65	349.34	**314.89**	337.41
20	**1916.04**	1990.62	1982.29	1959.30	757.84	**684.12**	**684.12**	690.02
Total	**18463.11**	18951.16	18491.23	18814.48	7256.22	6802.09	7108.14	**6718.55**

To compare the degree of satisfaction to user constraints, Fig. 8 shows the cumulative rewards of the successful execution of the 4 methods. From Fig. 8, we can see CSSC-MDP has the highest cumulative rewards. This illustrates that CSSC-MDP is best adaptive to the variable QoS values, and satisfy the user constraints as well as possible.

4.4 Experiments on Real-World Dataset

To further evaluate the performance of CSSC-MDP, we do service composition experiments on the dataset 2 of WS-DREAM, a real-world QoS dataset released by [24] and [25]. The dataset includes response time data $rtmatrix$ and through-put data $tpmatrix$, which are collected from 339 users on $5,825$ Web services. We group the services into 50 tasks, hence, each task has about 116 candidate services, and each service has 339 response time and through-put records.

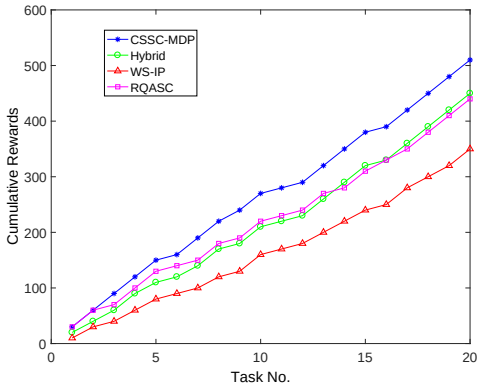

Fig. 8: Cumulative reward comparison for 4 methods.

Setting user constraints as $(ResponseTime \leq 22s)$ and $(ThroughPut \geq 13Kbps)$, and setting the number of tasks $t = 10, 20, 30, 40, 50$, we do the service composition using the 4 methods mentioned above for 1000 times respectively. The results are shown in Table 8 and Fig. 9.

TABLE 8: Comparison of times of successful execution.

Method	t=10	t=20	t=30	t=40	t=50
CSSC-MDP	795	615	514	445	276
RQASC	174	45	37	29	16
WS-IP	249	19	4	1	0
Hybrid	394	17	3	0	0

From the data in Table 8, we can see the success rate of CSSC-MDP on the real world dataset is significantly higher than that of other methods. More seriously, for methods WS-IP and Hybrid, when the number of tasks $t \geq 30$, the service compositions are almost all failed. This is because the uncertainty of service composition increases rapidly as the number of tasks increases. Fortunately, CSSC-MDP is more adaptive to the variable QoS and service behavior, so CSSC-MDP is relatively the most robust method.

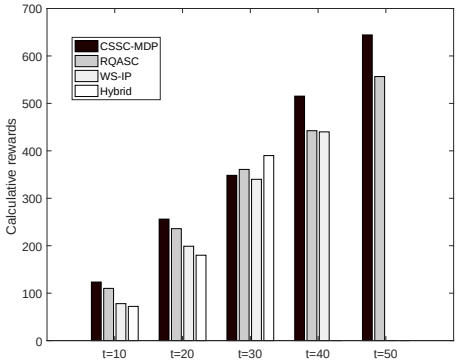

Fig. 9: Comparison of cumulative rewards on a real dataset.

Fig. 9 shows the mean cumulative reward of the successful executions of the 4 service composition methods. From Fig. 9 we can see CSSC-MDP is obviously superior to the others when $t = 10, 20, 40, 50$. And when $t = 30$, CSSC-MDP is very close to the highest cumulative reward. In conclusion, experimental results on the real world dataset verify the superior adaptivity of CSSC-MDP to the uncertain QoS and service behavior.

5 Related Work

In this section, we study the existing methods for constraint-satisfied service composition at first, then we discuss the service composition methods using reinforcement learning.

5.1 Methods for CSSC

The existing methods for solving CSSC problem can roughly be grouped into two groups: global optimization and local optimization.

5.1.1 Global Optimization Methods

The global optimization methods attempt to find the optimal composite service among all possible combinations while meeting user constraints.

In one of the early studies on constraint-satisfied service composition, Hassine et al. [4] formalized the CSSC problem as a constraint optimization problem at first; then an incremental user-intervention-based protocol was used to find the optimal composite service at run time. Yu et al. [5] modeled the CSSC problem in two ways: a multi-dimension multi-choice knapsack problem (MMKP) model and a multi-constrained problem (MCOP) model, and for each model, an efficient heuristic algorithm was proposed. Zhao et al. [11] modeled the CSSC problem using the weighted Tchebycheff distance, avoiding the limitations of linear functions in setting the weight of the QoS attributes, and proposed two evolutionary algorithms to solve optimal problems in different scenarios. Lecue and Mehandjiev [26] balanced semantic fit with QoS metrics, modeled the CSSC problem as a constraint-satisfaction problem, and adapted a hill-climbing algorithm to compute a "good enough" solution that met initial constraints rather than computing the optimal composition. Caporuscio et al. [27] built both design-time and run-time models for the service composition and identified the service composition satisfying the QoS requirements. The quality attributes of the selected composition are monitored, analyzed and, if necessary, plans are generated in terms of modifications.

Other optimization methods, such as, Ardagna and Pernici [6], Kritikos and Plexousakis [7] and He et al. [9] consider the CSSC process as mixed-integer linear programming (MILP) problem, and Garcia et al. [10] model the CSSC problem as constraint shortest path problem. In theory, such exhaustive global optimization methods can get the global optimal constraint-satisfied composite service, if it exists. However, the efficiency of global optimization methods decreases dramatically as the problem grows. Some approximate optimization algorithms, such as the stochastic search method [26], particle swarm optimization [28], evolutionary algorithm [11] and so on, have been studied to improve optimization efficiency in the CSSC problem. In general, most of the global service composition methods assume the QoS values are fixed and the service behaviors are determined, as a result, their adaptivity to the dynamic environments are poor.

5.1.2 Local Optimization Methods

The local optimization method decomposes global constraints for tasks in the workflow and selects the optimal service for each task, so as to meet local constraints independently.

Sun et al. [12] computed the utility of a composite service from the utilities of component services and derived the constraints of component services from the constraints of the composite service. Alrifai et al. [13] used a global optimization method to find the optimal decomposition of constraints, and then used the distributed local selection to find the best services satisfying the local constraints. Raj and Sasipraba [14] used local constraints as the thresholds to filter unqualified services and took service utility as the key value to select the optimal service for a task. Then the highest-ranked service was provided to user.

Because of the undetermined QoS values, it is almost impossible to get an appropriate decomposition of global constraints. The local constraints are either too strict or too loose, and the success rate of service composition is inevitably reduced.

Table 9 shows the pros and cons of these methods.

5.2 Service Compositions by Reinforcement Learning

Reinforcement learning is a typical technology used for planning and optimization in dynamic environments, and many researches has used reinforcement learning to conduct service composition, such as [17] considering the undermined service behaviors, [29] considering the uncertainty of the acture environment, [30] discussing the optimization of different composition structure, [31] finding multiple composition plans and selecting the most appropriate for user, [23] concerning the undetermined QoS, [32] facilitating the service composition, [33] considering the indeterminacy of service behavior, [36] finding the optimal workflow, [34] improving the optimization efficiency for large-scale service composition, [37] presenting an approach of multi-agent service composition, [35] building service pair to handle the change in dynamic environment etc. Table 10 shows the pros and cons of some presentative methods.

In contrast to these works, the proposed CSSC-MDP satisfies the user's functional requirements and QoS constraints furthest and considers the conflicts between the QoS attributes. Moreover, CSSC-MDP selects a component service after the execution of the previous service. Thus, before the selection of a component service, the execution QoS of the previous services are certain values, hence the constraints which need to be satisfied by the following services can be calculated. Based on the results of calculation, the component service mostly satisfied the constraints can be selected. Even if in the case where a selected service failed, the optimal alternative service can be selected immediately to replace it. Therefore, CSSC-MDP is highly adaptive to the dynamic and uncertain environment.

TABLE 9: Methods for CSSC

Method/Model	Advantage	Weakness
incremental user-intervention-based protocol [4]	adaptive to stochastic service evolution	not adaptive to the variable QoS
MMKP and MCOP [5]	optimize the QoS of the composite service	not adaptive to the variable QoS
MILP [6], [7], [9]	get the global optimal solution	the efficiency decreases dramatically as the problem grows
multi-objective optimization [11]	the limitations of linear functions are avoided	not adaptive to the variable QoS
hill-climbing algorithm [26]	high solving efficiency	the solution is locally optimal and not adaptive to the dynamic environment
design-time model and run-time model [27]	services that most likely contribute in QoS violations are get rid off	new services have no chance to take part in
local selection approach[12]	high efficiency	not adaptive to the variable QoS
constraints decomposition [13]	the decomposition of constraints is optimized	QoS values are regarded as determined
local-global combined [14]	use local constraints to filter unqualified services	not adaptive to the variable QoS and uncertain service behaviors

TABLE 10: Service composition methods using reinforcement learning

Model	Advantage	Weakness
MDP+Bayesian learning [17]	can generate robust workflow	can not guarantee the optimality of composite services
MDP+Bayesian learning [29]	improve the quality of Workflow through Bayes learning	without considering the QoS of the composite service
MDP [30]	different structures of service composition are considered	only one QoS attribute can be optimized
MDP+HTN planning [31]	multiple QoS attributes are taken into consideration	take the QoS values as be determined
MDP[23]	measure QoS by mean and variance, reduce the probability of composite service failure	the real QoS value could not be consistent with the theory distribution
improved MDP [32]	facilitate the service composition by improving the optimization equation	the real QoS value could not be consistent with the theory distribution
MDP [33]	integrate multiple workflows and alternative services into service composition	the optimal service composition can only be find in the long run
team Markov Games [34]	applicable to the distributed environment	the optimal service composition can only be find in the long run
MDP [35]	provide flexible service composition	can only get the near optimal composite service

6 Conclusions

In this paper, we presented a reinforcement learning method for solving the CSSC problem in the dynamic environment. This method built a CSSC-MDP model based on an MDP model and used a Q-learning algorithm to solve the model. Extensive experiments show a significant improvement in terms of adaptivity to the uncertain behavior of services in the dynamic environment. In comparison with a global optimization method, a local optimization method and an MDP-based service composition method, CSSC-MDP also showed the superiority in terms of the satisfaction of user's QoS constraints.

The main contributions of this paper are threefold: first, considering the uncertain behavior of service, we selected component services during execution of the service composition; this avoids the service composition failure brought by the failure of a component service. Second, considering that QoS attributes are not always the same for different executions, we select a component service based on the real execution QoS data of the previous services; this will help to calculate the real constraints that need to be satisfied by the following service. Thirdly, the selection strategy of Q-learning is globally optimized, this guarantees the optimality of the composite service.

Acknowledgements

The work described in this paper was partially supported by the National Natural Science Foundation of China(No. 61673249,61273291), Research Project Supported by Shan xi Scholar ship Council of China(No. 2016-004).

Wenjian Wang is the corresponding author.

References

[1] I. Ari and N. Muhtaroglu, "Design and implementation of a cloud computing service for finite element analysis," *Advances in Engineering Software*, vol. 60, pp. 122–135, 2013.

[2] R. Rezaei, T. K. Chiew, S. P. Lee, and Z. S. Aliee, "A semantic interoperability framework for software as a service systems in cloud computing environments," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5751–5770, 2014.

[3] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decades overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[4] A. B. Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal web service composition," in *The Semantic Web-ISWC 2006*. Springer, 2006, pp. 130–143.

[5] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, p. 6, 2007.

[6] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *Software Engineering, IEEE Transactions on*, vol. 33, no. 6, pp. 369–384, 2007.

[7] K. Kritikos and D. Plexousakis, "Mixed-integer programming for qos-based web service matchmaking," *Services Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 122–139, 2009.

[8] D. Ardagna and B. Pernici, "Global and local qos constraints guarantee in web service selection," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[9] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction," *Software Engineering, IEEE Transactions on*, vol. 40, no. 2, pp. 192–215, 2014.

[10] G. A. Garcia Llinas and R. Nagi, "Network and qos-based selection of complementary services," *Services Computing, IEEE Transactions on*, vol. 8, no. 1, pp. 79–91, 2015.

[11] X. Zhao, L. Shen, X. Peng, and W. Zhao, "Toward sla-constrained service composition: An approach based on a fuzzy linguistic preference model and an evolutionary algorithm," *Information Sciences*, vol. 316, pp. 370–396, 2015.

[12] S. X. Sun and J. Zhao, "A decomposition-based approach for service composition with global qos guarantees," *Information Sciences*, vol. 199, pp. 138–153, 2012.

[13] M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient web service composition with end-to-end qos constraints," *ACM Transactions on the Web (TWEB)*, vol. 6, no. 2, p. 7, 2012.

[14] R. J. R. Raj and T. Sasipraba, "Web service selection based on qos constraints," in *Trendz in Information Sciences & Computing (TISC), 2010*. IEEE, 2010, pp. 156–162.

[15] Q. Hu and J. Liu, "An introduction to markov decision processes," *Xidian University, Xian, China*, 2000.

[16] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281–308, 2004.

[17] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using markov decision processes," in *Web Services, 2004. Proceedings. IEEE International Conference on*. IEEE, 2004, pp. 576–582.

[18] A. Slomiski, "On using bpel extensibility to implement ogsi and wsrf grid workflows," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1229–1241, 2006.

[19] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, "Clustering web services to facilitate service discovery," *Knowledge and information systems*, vol. 38, no. 1, pp. 207–229, 2014.

[20] W. Chen, I. Paik, and P. C. Hung, "Constructing a global social service network for better quality of web service discovery," *Services Computing, IEEE Transactions on*, vol. 8, no. 2, pp. 284–298, 2015.

[21] E. Al-Masri and Q. H. Mahmoud, "Qos-based discovery and ranking of web services," pp. 529–534, 2007.

[22] R. Bellman, "A markovian decision process," DTIC Document, Tech. Rep., 1957.

[23] X. Fan, C. Jiang, J. Wang, and S. Pang, "Random-qos-aware reliable web service composition," *Journal of software*, vol. 20, no. 3, pp. 546–556, 2009.

[24] Z. Zheng, Y. Zhang, and M. R. Lyu, "Distributed qos evaluation for real-world web services," in *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 83–90.

[25] Y. Zhang, Z. Zheng, and M. R. Lyu, "Exploring latent features for memory-based qos prediction in cloud computing," in *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*. IEEE, 2011, pp. 1–10.

[26] F. Lecue and N. Mehandjiev, "Satisfying end user constraints in service composition by applying stochastic search methods," *Web Service Composition and New Frameworks in Designing Semantics: Innovations: Innovations*, p. 238, 2012.

[27] M. Caporuscio, R. Mirandola, and C. Trubiani, "Qos-based feedback for service compositions," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2015, pp. 37–42.

[28] X.-Q. Fan, "A decision-making method for personalized composite service," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5804–5810, 2013.

[29] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using markov decision processes," *International Journal of Web Services Research*, vol. 2, no. 1, p. 1, 2005.

[30] A. Gao, D. Yang, S. Tang, and M. Zhang, "Web service composition using markov decision processes," in *Advances in web-age information management*. Springer, 2005, pp. 308–319.

[31] J. Xu, K. Chen, and S. Reiff-Marganiec, "Using markov decision process model with logic scoring of preference model to optimize htn web services composition," *International Journal of Web Services Research (IJWSR)*, vol. 8, no. 2, pp. 53–73, 2011.

[32] X. Fan, X. Fang, and Z. Ding, "Indeterminacy-aware service selection for reliable service composition," *Frontiers of Computer Science in China*, vol. 5, no. 1, pp. 26–36, 2011.

[33] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya, "Adaptive service composition based on reinforcement learning," in *Service-Oriented Computing*. Springer, 2010, pp. 92–107.

[34] H. Wang, Q. Wu, X. Chen, Q. Yu, Z. Zheng, and A. Bouguettaya, "Adaptive and dynamic service composition via multi-agent reinforcement learning," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 447–454.

[35] J. Yang, W. Lin, and W. Dou, "An adaptive service selection method for cross-cloud service composition," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 18, pp. 2435–2454, 2013.

[36] H. Wang, X. Zhou, X. Zhou, W. Liu, and W. Li, "Adaptive and dynamic service composition using q-learning," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1. IEEE, 2010, pp. 145–152.

[37] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, and A. Bouguettaya, "Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition," in *Service-Oriented Computing*. Springer, 2014, pp. 154–168.

Lifang Ren received her M.S. degree from the department of computer science and technology, Taiyuan University of science and technology in 2004. She is currently a PhD candidate with the School of Computer and Information Technology, Shanxi University. Now she is also a lecture with the School of Applied Mathematics, Shanxi University of Finance and Economics. Her main research interests include service computing and trustworthy software.

Wenjian Wang obtained her Ph.D. degree from Institute for Information and System Science, Xi'an Jiaotong University in 2004. Now she is a full-time professor and Ph.D. supervisor at School of Computer and Information Technology and Key Laboratory of Computational Intelligence and Chinese Information Processing, Shanxi University. She has published more than 70 academic papers. Her current research interests include machine learning, data mining, etc.

Hang Xu received her M.S. degree from school of Computer and Information Technology, Shanxi University in 2014. She is currently a PhD candidate with the School of Computer and Information Technology, Shanxi University. Her main research interests include service computing and machine learning.