Knowledge-Based Systems 180 (2019) 75–90

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Adaptive and large-scale service composition based on deep reinforcement learning☆

Hongbing Wang [a,*], Mingzhu Gu [a], Qi Yu [b], Yong Tao [a], Jiajie Li [a], Huanhuan Fei [a], Jia Yan [a], Wei Zhao [a], Tianjing Hong [a]

[a] School of Computer Science and Engineering and Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing, 211189, China
[b] College of Computing and Information Sciences, Rochester Institute of Tech, USA

ARTICLE INFO

Article history:
Received 4 August 2018
Received in revised form 1 April 2019
Accepted 11 May 2019
Available online 21 May 2019

Keywords:
Service composition
QoS
Deep reinforcement learning
Adaptability

ABSTRACT

In a service-oriented system, simple services are combined to form value-added services to meet users' complex requirements. As a result, service composition has become a common practice in service computing. With the rapid development of web service technology, a massive number of web services with the same functionality but different non-functional attributes (e.g., QoS) are emerging. The increasingly complex user requirements and the large number of services lead to a significant challenge to select the optimal services from numerous candidates to achieve an optimal composition. Meanwhile, web services accessible via computer networks are inherently dynamic and the environment of service composition is also complex and unstable. Thus, service composition solutions need to be adaptable to the dynamic environment. To address these key challenges, we propose a new service composition scheme based on Deep Reinforcement Learning (DRL) for adaptive and large-scale service composition. The proposed approach is more suitable for the partially observable service environment, making it work better for real-world settings. A recurrent neural network is adopted to improve reinforcement learning, which can predict the objective function and enhance the ability to express and generalize. In addition, we employ the heuristic behavior selection strategy, in which the state set is divided into the hidden and fully observable state sets, to perform the targeted behavior selection strategy when facing with different types of states. The experimental results justify the effectiveness and efficiency, scalability, and adaptability of our methods by showing obvious advantages in composition results and efficiency for service composition.

1. Introduction

In the field of service computing, web service is the most promising technology to implement the Service-Oriented Architecture (SOA), which is independent, modular and interoperable [1]. In recent years, along with the development of computer networks especially in the era of cloud computing, quite a few of enterprises publish their products in the form of services for people to use on the Internet, resulting in a dramatic growth in the number of web services. Web service composition, as a way to provide value-added functionalities, does not need to completely recreate a new service. It also provides attractive properties, such as encapsulation, reusability/interoperability, autonomy and loose coupling that effectively improve the efficiency of software development. Given the massive number of services with similar functionalities, Quality of Service (QoS) has become the most important factor to distinguish different services. QoS is an indicator to evaluate the non-functional attributes and the ability of a service to meet the users' requirements. Thus, performing QoS-aware service selection has emerged as a key step in the service composition optimization [2,3].

In practical applications, on the basis of meeting the users' requirements, the quality, adaptability and efficiency are important criteria to evaluate a service composition solution [4]. However, services usually operate in a network environment that is inherently dynamic, which makes these criteria unpredictable. For example, when the transmission time of the network is long and sometimes the network cannot be connected, the response time of the service may be long, even the service is unavailable. Meanwhile, the changes of web services also increase the variability of the composition process [5,6]. Therefore, a good web service composition solution needs to be able to cope with

☆ We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

* Corresponding author.
E-mail addresses: hbw@seu.edu.cn (H. Wang), qi.yu@rit.edu (Q. Yu).

https://doi.org/10.1016/j.knosys.2019.05.020
0950-7051/

these dynamics. In addition, the composition functional workflow becomes more complex along with the complicated users' requirements, leading to the increase of abstract service nodes in the workflow. Moreover, the growth of the homogeneous services (with the same functionality but different QoS) also expands the candidate service space. More specifically, if m is the number of abstract services in a composition workflow and n is the number of candidate services of each abstract service, the number of candidate composition solutions will be n^m. In such a large-scale service composition scenario [7,8], the "curse of dimensionality" issue may arise, creating another significant challenge to service composition.

Facing with the challenge of adaptability in service composition, some existing researches focus on using the reinforcement learning (RL) technology to accommodate the dynamic environment. RL discovers what to do by maximizing the accumulative reward. However, the existing RL methods are falling short for the large-scale scenarios [4]. Some further improvements [9–11], such as multi-agent, hierarchical RL, and Semi Markov Decision Process (SMDP), are insufficient to address the inherent issues of RL. More specifically, the table-based RL algorithms only perform well in small-scale problems because of lack for generalization ability.

Another key limitation of existing solutions is that all states are required to be fully observable conforming to the Markov Decision Process (MDP). This may be opposite to the real-world, which is typically partially observable. Therefore, finding an optimal strategy for a non-Markovian environment is still an opening issue.

In order to address the above challenges, we first propose a deep reinforcement learning model, referred to as Adaptive Deep Q-learning and RNN Composition Network (ADQRCN), for large-scale dynamic service composition. Through deep learning (DL), a function approximation method is used to simulate the Q-value table to address the scalability issue. In addition, the change of QoS of Web services exhibits some temporal regularities, which will be captured by a Long Short Term Memory (LSTM) network, a variant of a Recurrent Neural Network (RNN). We further employ Partially Observable Markov Decision Process (POMDP) to accommodate a partially observable environment, leading to a POMDP based Web Service Composition model (POMDP-WSC). A One State One Network for Web Service Composition (OSON-WSC) framework is developed based on ADQRCN, where OSON is a heuristic strategy to select different behaviors according to the observability of abstract service states. We summarize our major contributions below:

- We introduce a POMDP-WSC model, by considering the partially observable environment in real-world settings. The model depicts the workflow of a composition and takes into account QoS attributes, which allows it to evaluate a composition solution in terms of quality, adaptability, and effectiveness.
- Based on whether the state node is fully observable, we develop the OSON strategy, which consists of two different heuristics, to select actions, aiming to choose the behavior strategy, and effectively improve the accuracy and efficiency of service composition. An OSON-WSC framework integrating ADQRCN with OSON is proposed to achieve large-scale and adaptive service composition in a partially observable environment.
- We conduct a series of experiments to verify the validity of our solutions from the following aspects: effectiveness and efficiency, adaptability, and scalability.

Table 1
Notations.

Notation	Description
QoS	Quality of Service
RL	Reinforcement Learning
MLP	Multi-Layer Perceptron
DL	Deep Learning
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
WSC	Web Services Composition
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
ADQRCN	Adaptive Deep Q-learning and RNN Composition Network
OSON-WSC	One State One Network for Web Service Composition Model

The ADQRCN model was first proposed in a shorten form as a conference paper [12], which is for service composition in a fully observable environment. In this extended paper, the ADQRCN is integrated with the OSON to address the large-scale dynamic service composition problem in a partially observable environment, which is suitable for real-world settings. The remainder of the paper is organized as follows. Section 2 gives an overview of some related works. Section 3 covers the preliminary knowledge, including reinforcement learning, deep learning, and deep reinforcement learning, which forms the important theoretical foundation. Section 4 presents a scenario of service composition, which helps identify the main challenges in real-world service composition. Section 5 details the proposed service composition method using deep reinforcement learning. It also describes a heuristic strategy to further improve the efficiency and adaptability. Section 6 shows the experimental results along with a detailed analysis to verify the validity of the proposed approach. Section 7 presents the concluding remarks and lays out some future directions. Table 1 summarizes the main notations used in the paper.

2. Related work

In this section, some related works will be discussed to review the existing solutions regarding large-scale and adaptive service composition. We focus our discussion on planing solutions, evolutionary algorithms, reinforcement learning (RL) for service composition, the application of deep reinforcement learning (DRL) in artificial intelligence and service composition, and Partially Observable Markov Decision Process (POMDP) in related fields.

The challenge of adaptability has attracted widespread attention nowadays and many solutions have been developed, including integer programming, graph planning, artificial intelligence, and so on. In [13], a service composition model based on AI planning was developed. A preparing approach was used to tackle the changes in process of service composition. The solution effectively avoids restructuring services, resulting in the improvement of efficiency. However, the replacement of individual services lacks an overall consideration of the entire composition result. Ardagna et al. [14] proposed to build a multi-channel adaptive information model to implement service selection and composition, which can flexibly map the abstract services to concrete services. It realizes redundancy mechanisms through multi-channel integrated with integer programming, achieving the optimal composition result. In some other works, the problem of service composition is transformed to a graph planning problem [15]. In order to better reflect the relationship between services, greedy search is applied to repair the internal parts, which targets to replace the failure services. Beauche et al. [16] adopted hierarchical planning to establish an adaptive service composition model.

However, the change of services and the environment should respond to the update of the model, which is not suitable for a large-scale scenario.

Some other existing works use evolutionary algorithms for service composition. Silva et al. used genetic programming (GP) for web service selection and composition [17]. Three different GP methods were proposed to generate the correct solutions, but these methods are not effective in a large-scale environment. An ant colony optimization algorithm was developed for QoS-aware service selection in [18]. This method supported global QoS optimization and dynamic composition. In [19], a Discrete Gbest-guided Artificial Bee Colony (DGABC) algorithm was proposed to simulate the search for the optimal service composition solution through the exploration of bees for food. However, these methods tend to fall into local optimum and suffer from slow convergence.

Besides the above technologies, some efforts have been devoted to incorporate reinforcement learning into service composition. Reinforcement learning (RL) employs the trial and error exploration to discover the optimal strategy [20]. Wang et al. [21] exploited Markov Decision Process (MDP) integrated with reinforcement learning to find the optimal composition result. The MDP model achieved multiple workflows without preliminary knowledge of service quality and environment. When facing with the changeable environment, the learning process can perceive and respond to the changes. Some modifications have been applied to RL to address the efficiency issue. In [22], a multi-objective reinforcement learning method was proposed to achieve the optimal service composition with multiple QoS objectives. The method can address the service composition in the absence of prior knowledge of QoS data and the undefined user preferences. However, this method is not suitable for a large-scale environment. In [9], an adaptive service composition method was developed based on hierarchical reinforcement learning. Another adaptive service composition approach employed a Multi-agent SARSA method that integrates on-policy reinforcement learning and game theory [4]. Multi-agent reinforcement learning has also been used for service composition, which leverages distributed Q-learning and experience sharing to improve the learning efficiency [23,24]. These methods adopted the MDP model, and the Nash equilibrium between the agents was not considered, which makes them easily trap into the local optimum. In [25], a new model was built based on Team Markov Games. In order to solve the problems of agent coordination and equilibrium selection, the coordination equilibrium and fictitious play process was introduced to ensure agents' convergence with a unique equilibrium. However, the communication and coordination between agents still consume significant computation resources. It is also easily trapped into the local optimum.

In order to address the high-dimensional feature space which triggers the low performance of reinforcement learning, deep learning can be used to extract the important characteristics from raw data. Riedmiller et al. [26] employed a multi-layer perceptron to approximate the Q-value to achieve the Neural Fitted Q Iteration (NFQ) algorithm. The work in [27] presented a Deep Auto-Encoder (DAE) model by integrating deep learning with reinforcement learning to solve the problem of visual perception. However, it is limited to a small dimensional space. In recent years, Google's artificial intelligence team, DeepMind, employed Deep Reinforcement Learning (DRL), to achieve substantive breakthrough in large dimensions of raw input data and decision-making tasks. Mnih et al. [27] applied DRL to the Atari 2600 game. It successfully learnt control strategy from the high dimension sensation input and achieved expert level performance. In the work [28], a method was developed to address difficulties of a high dimensional search space, position localization, and action selection. Besides, DRL is also used for service composition. In [29], a Convolutional Neural Network (CNN) was combined with RL to build a deep Q network for large-scale service composition. A double Q-learning with prioritized replay scheme was adopted to improve efficiency. While the experiments demonstrated the effectiveness of this method, it is only suitable for the fully observable MDP environment. In contrast, our method considers the time correlation of QoS value and integrates RNN with RL for large-scale dynamic service composition. In addition, we propose to use the Partially Observable Markov Decision Process to simulate a partially observable environment, which is typical for many real-world applications.

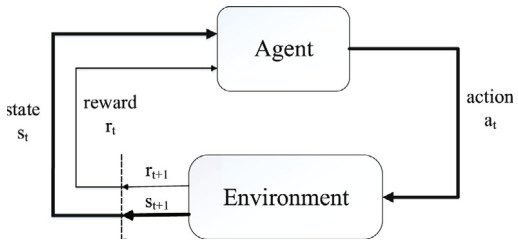


Fig. 1. The framework of reinforcement learning.

Partially Observable Markov Decision Process (POMDP) has been developed as an extension of MDP to solve partially observable problems. In [30], a POMDP framework was introduced where the state of road segment was divided into two types of states: non-collision and collision to formalize the active collision detection problem. Wray et al. [31] proposed to map the PAL problem to a POMDP, which took the observed history information into account. This method considered all aspects of PAL within a unified mathematical framework. Thus, in order to better simulate the process of service composition, some works also adopted POMDP to solve service composition problems. Yu et al. [32] modeled the uncertainty and instability of web service behavior into partially observable variables through POMDP. In [33], a self-maintenance method of service system based on POMDP model was introduced to solve the problem of changes in the network environment and incomplete information of third party services. It can be seen that the POMDP has led to some promising results in modeling service compositions in real-world settings. Therefore, this paper adopts a POMDP in combination with reinforcement learning and deep learning and develops novel strategies to obtain optimal service composition results that are suitable for the large-scale and dynamic service composition scenarios.

3. Preliminaries

Before presenting the proposed approach, we first introduce some preliminaries that lay the foundation for our later discussions. These include Reinforcement Learning, Deep Learning, and Deep Reinforcement Learning.

3.1. Reinforcement learning

As one of the most important branches in machine learning, Reinforcement Learning (RL) maps environmental states to actions and aims to achieve the most cumulative reward value of actions from the environment. RL leverages environmental signals to evaluate actions, rather than through test cases to train an agent to take the correct action. This is different from both supervised and unsupervised learning. In the process of RL, an agent knows little about the external environment. So the main task is to collect significant information to improve the learning.

A standard RL framework is shown in Fig. 1. The agent interacts with the external environment through actions, and the external environment offers the feedback information to inform the agent whether it should continue to perform this action on the current state in the future. In fact, the agent has a state sensor, which can map an environmental state s to its internal perception. The agent will take action a_t according to the current strategy of action selection, and get reward r_t. Then, it will update the strategy according to the feedback information from the environment. Finally, the environment will transit to the next state s_{t+1} under the action a_t. In general, the basic principle of RL is to obtain different signals from the environment. If an agent obtains the reward signal, the tendency of this action will be strengthened and weakened otherwise. Furthermore, the agent should not only consider the short-term reward value, but also take the long-term impact of the agent's behavior into account. Thus, the infinite discount model is adopted in RL, given by:

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{1}$$

where γ is the discount factor, $0 < \gamma \leq 1$, and r_t is the reward from state s_t to the next state s_{t+1}. Obviously, if the objective function has been determined, the optimal behavior strategy can be presented as:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \quad \forall s \in S \tag{2}$$

The basic theory of RL is the Markov Decision Process (MDP), which assumes that the environment conforms to the MDP. In an MDP, the current state and the choice of action will lead to different transition probabilities and rewards. If the state transition probability function P and reward function R are given, the problem can be solved by dynamic programming. However, in a normal situation, P-function and R-function are unknown to an agent, so a common solution is to dynamically adjust the strategy and estimate value of the next state according to the actual situation. The value function can be defined as Eq. (3), which adopts the Bellman iterative strategy:

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha(r(s,a,s') + \gamma V(s')) \tag{3}$$

Q-learning [34] is a widely used method of RL. By approximating the value function of a state–action pair, Q-learning can reduce the difference between neighboring condition estimated Q-value in each step of learning. The iteration of Q-function is defined as

$$Q\ (s,a) \leftarrow (1-\alpha)Q\ (s,a) + \alpha\left[r + \gamma \max_{a'} Q\ \left(s', a'\right)\right] \tag{4}$$

where α is the learning rate, γ is the discount factor, and $Q(s,a)$ is the state–action value under state s executing action a, $max_{a'}Q(s', a')$ presents the optimal reward of state–action value in state s'.

According to the above concept of Q-function, we can achieve Q-learning using table checking to update the value. The behavior strategy depends on whether the Q-value table becomes sophisticated. It also introduces two issues. The first is the "curse of dimensionality", which will happen with the increase of the size of the state space. The second problem is that in a partially observable environment, the position of searching will be more arbitrary. In order to improve the efficiency of Q-learning and accurate prediction of an agent in case of incomplete prior knowledge, this paper proposes to exploit Deep Learning-Recurrent Neural Network (RNN), which will be introduced below.

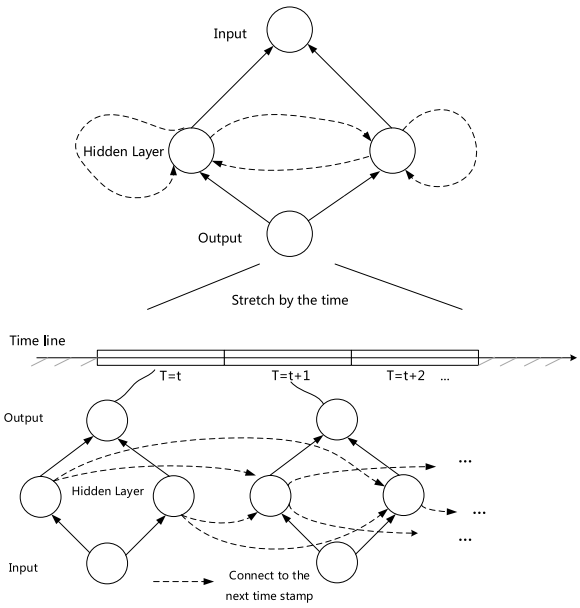


Fig. 2. The principle of RNN.

3.2. Deep learning-RNN

Deep Learning (DL) originates from Artificial Neural Networks (ANN). DL model usually consists of multiple nonlinear prediction units. The output of lower layers serves as the input to the next higher layer. The basic model of deep learning is categorized into Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) [35]. We adopt RNN due to its capability to handle temporal dynamics to address the problem of service composition.

RNN is one type of deep learning network, which consists of one or more feedback loops to strengthen the ability of neural network for time modeling. RNN is suitable for the serialized data and can simulate these data more accurately. The structure of RNN records the activation value of each time, which enhances the time correlation of network through adding the hidden layers. RNN has successfully been applied to a wide range of domains, including linguistic modeling [36], machine translation [37], and pattern recognition [38].

In service composition, each service may change with time. But the changed service may still be related with the original one given a period of time instead of being completely irrelevant. For example, if the accessibility and success ratio of a service are quite high in previous performance, and the response time is also fast, though network condition may cause the fluctuation of service QoS attributes, the trend should still be quite regular and various attributes are following a similar trend. Thus, the former information should be effectively leveraged instead of being completely discarded. We propose to employ RNN to solve the time correlation issue. An RNN contains a circulatory network, allowing information persistence. The Fig. 2 depicts the principle of RNN. By two time-step expansion over the entire network structure, the network can be presented with a loop-free form. From the figure, we can conclude that the depth of the network is reflected not only by the input/output, but also by cross-time steps, each of which can also be considered as a layer.

While being developed in 1996 [39], the wide usage of RNN has been hindered by the difficulty of training serialized parameters because of massive parameters and "vanishing gradien". Hochreiter et al. [40] improved the RNN and proposed the Long

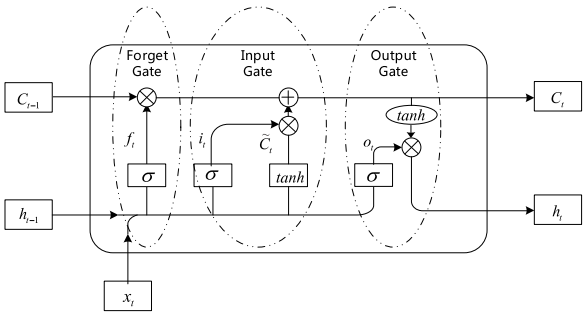


Fig. 3. A simple LSTM block.

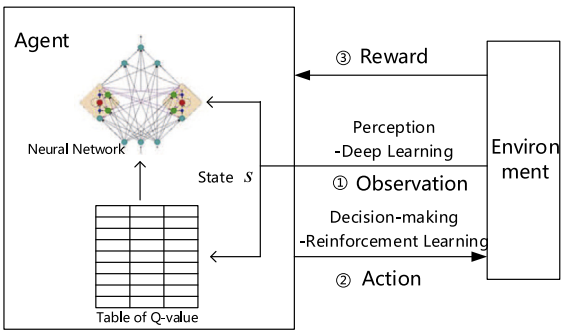


Fig. 4. The principle of deep reinforcement learning.

Short Term Memory (LSTM) network, using the gate and the cell unit to store the dependency information of a longer time. Recently, under the improvement and promotion of Alex Graves, the LSTM network has achieved a huge success and been widely used [41]. Therefore, we incorporate the LSTM network to store historical information and composition results with the relevance in time, in order to solve the large-scale service composition problem.

The core improvement of LSTM is the extension of neural network with memory. Three other layers are added as hidden memory units compared with the original RNN, including the input gate, output gate, and forget gate, respectively. These three gates apply the "inertia" mechanism of information gate, which controls exactly how much original information is preserved and how much new information is added. The following introduction of the LSTM structure forms the core of its information gate, shown in Fig. 3.

The LSTM can be divided into three parts. The forget gate decides what information is discarded from the cell, and the output value (between 0 and 1) delivers to cell state C_{t-1}, where 1 represents full reservation and 0 indicates complete abandonment. The calculation is given by

$$f_t = \sigma(W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f) \tag{5}$$

where W represents the weights, especially W_{hf} represents the weights between the hidden layer h and the forget gate f, W_{xf} represents the weights between the input x and the forget gate f, b represents the bias, b_f represents the bias of the forget gate f, and σ is the sigmoid function.

To determine what kind of information can be put into the cell consists of two parts: one part from the input gate that decides to update which parts and another part is the *tanh* layer to create a new candidate vector, given by

$$i_t = \sigma(W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i) \tag{6}$$

$$g_t = tanh(W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g) \tag{7}$$

Then, it needs to update the old information in the output of the cell. The output gate discards the information and mixes the new information in the former step, given by

$$C_t = f_t * C_{t-1} + i_t * g_t \tag{8}$$

Finally, we need to determine the output calculated by

$$o_t = \sigma(W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o) \tag{9}$$

$$h_t = o_t * tanh(C_t) \tag{10}$$

3.3. Deep reinforcement learning

Deep learning (DL) and the reinforcement learning (RL) are hot research topics in the field of machine learning [42]. The fundamental rationale of DL is to form high-level expression to discover the distributional characteristic expression [43] through the multi-layered network architecture and the nonlinear transformation. The purpose of RL is to maximize the accumulative reward value, which is gained from the environment by an agent, and finally achieve an optimal strategy [44]. Google DeepMind Team combined the perception of Deep Learning and decision-making ability of Reinforcement Learning to form the Deep Reinforcement Learning (DRL). The concept of DRL becomes the new research hot spot of artificial intelligence and is widely applied to diverse domains, including game [27,45], robot control [46,47], vision research [48,49], and beyond. The learning process of DRL is shown in Fig. 4, which is divided into three steps.

1. An agent obtains observation through interacting with the environment (via RL) and delivers the high dimension results to a neural network, in order to learn the abstract representations;
2. The agent evaluates the action based on the reward value, and maps the current state to a corresponding action by the behavior strategy;
3. The environment responds to the action and gets the next observation.

Finally, with the learning of above processes, the optimal strategy can be achieved.

For RL to achieve the optimal result and convergence, it requires two important premises: (1) A lookup table is available to store Q values; (2) The environment should conform to the Markov property. The first premise limits RL to be only suitable for small-scale problems, because of the insufficient expression and generalization ability. In addition, the latter premise implies that the agent clearly knows all of environmental information and the next state can only be decided by the current state and the action. However, the real settings are more complicated and the environment is usually partially observable. Thus, to address these disadvantages, we firstly propose to adopt an RNN to remember the continuous state information in historical timeline, which leads to an Adaptive Deep Q-learning and RNN Composition Network (ADQRCN), and then a One State One Network for Web Service Composition (OSON-WSC) framework, which integrates ADQRCN with One State One Network (OSON), is constructed for large-scale and dynamic service composition in partially observable environment.

4. Problem formulation

We provide the formal problem definition in this section. Before that, we describe a scenario to illustrate some key challenges in service composition.

Table 2
The weather forecast services API.

Service provider	Service invoker
China weather	http://m.weather.com.cn/mweather/101190101.shtml
Google	http://www.google.com/ig/api?weather=Nanjing
YAHOO	http://weather.yahooapis.com/forecastress
Tecent	http://weather.news.qq.com/qresult.html
Sogou	http://123.sogou.com/get123.php?block=wt?ver=v32&city=CN110100
360	http://cdn.weather.hao.360.cn/api_weather_info.php?app=hao360&jsonp=smartloaddata101190101&code=101190101
MSN	http://weather.msn.com/data.aspx?wealocations=wc
Sina	http://weather.news.sina.com.cn/
...	...

4.1. The vacation planning scenario

Consider a planner who wants to arrange his trip schedule after departure and return date/time has been determined. The complete flow chart is shown in Fig. 5, in which we can conclude the difference of user preference from the choice of hotel and transportation. For example, some travelers prefer the comfort of traveling while others may have more strict requirements for travel expenses. Each abstract service may correspond to a large number of candidate services and Table 2 presents some web services for inquiring the weather forecast, which are only a small portion in hundreds of the Internet services. The vacation planning scenario is not particularly complex. However, if we assume that each abstract service corresponds to 300 candidate services, the five abstract services will form 300^5 different service composition solutions, which already forms a huge search space.

From the dynamic perspective of the environment, the network situation has a close relationship with web services, which also determines user's satisfaction to services invoked. For example, if the network condition is poor, the service response time will be lengthened, and will inevitably increase the user waiting time. In the vacation planning scenario, web services themselves and the network environment are all changeable, so the whole scenario is dynamic. In addition to the large scale and dynamics, the environment is also only partially observable. From Fig. 5, the type of transportations to tourist attractions after choosing the hotel is the same as that when the trip is started. The agent facing with the same choices is not able to distinguish different states and make the correct decision. This scenario is typical for the partially observable issue, in which an agent gets the same observation under different environment conditions, but may require different actions.

4.2. Web service composition model

The vacation planning scenario is represented using a flow chart for illustration purpose. In order to solve the problem using a computing system, the flow chart needs to be transformed into a functional model that can be understood by a machine. For this purpose, we develop an abstract model of Web service.

Definition 1 (*Web Service*)**.** A Web Service is a 6-tuple WS = $\langle ID, In, Out, Pr, E, QoS\rangle$, where

- *ID* presents the unique identifier of a WS.
- *In* is the input of a WS.
- *Out* is the output of a WS.
- *Pr* presents the preconditions for the normal operation of a WS.
- *E* is the environmental impact of WS execution.
- *QoS* is the set of attributes of a WS. *QoS* is an n-tuple $\langle attr_1, attr_2, \ldots, attr_n\rangle$, where $attr_i$ presents an attribute of WS (such as throughput, reliability, availability, and response time).

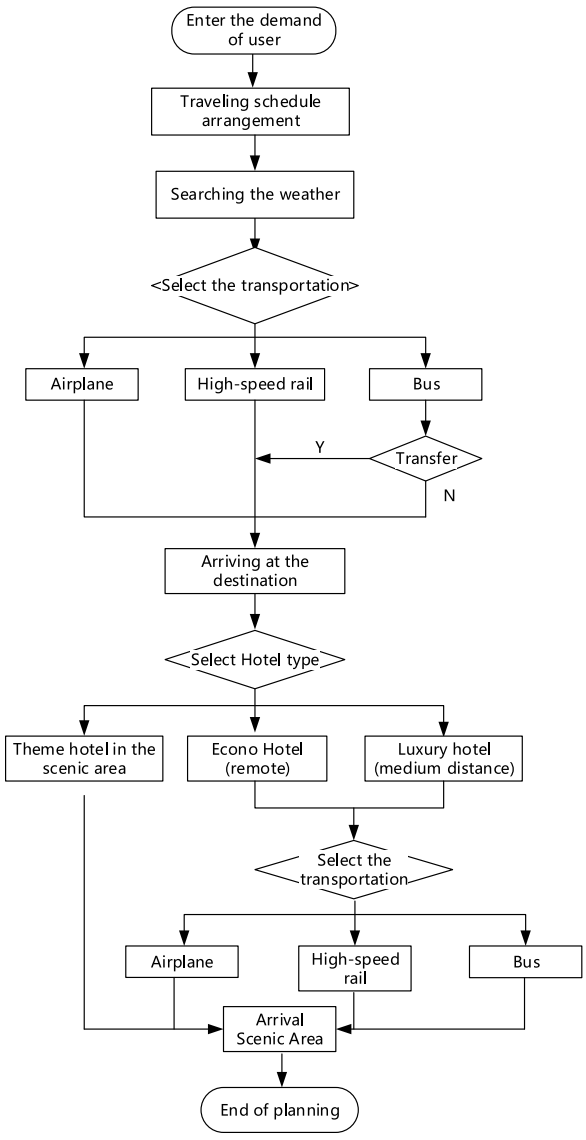


Fig. 5. The flow chart of vacation planning.

For example, a WS of hotel reservation is described as follows.

- *ID*: the hotel reservation service id.
- *In*: location, budget, environment.
- *Out*: the customer order.
- *Pr*: sufficient network resources.
- *E*: the generation of a reservation order.
- *QoS*: service charge: $1, response time: 200 ms, availability: 99%, and price: $120.

Web service composition is a process that integrates several services to a value-added composite service. Based on the description of web services and the uncertainties of the environment, the POMDP is used to describe the WS composition process. POMDP is an extension of MDP, where the agent uses an effective strategy to determine the action sequence when the current environment state is partially observable. For example, the packaging process consists of four steps: open the box, put the gift inside, close the box, and seal the box. Assuming the current box is in the closed state, the agent does not know whether the gift is in the box or not. As a result, it is impossible to decide whether to seal or open the box. Such a problem (referred to as a hidden state problem) can be solved by historical experience, such as whether the gift

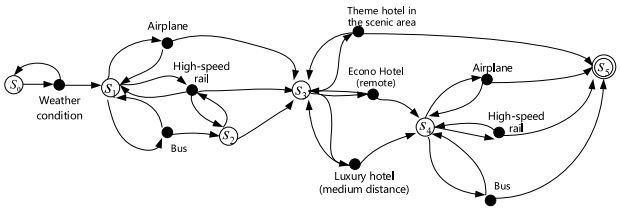


Fig. 6. The POMDP-WSC model for vacation planning.

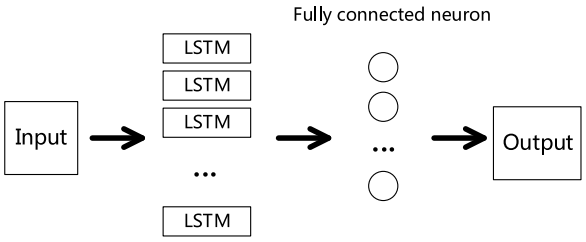


Fig. 7. The structure of ADQRCN.

has been placed in the box to determine the current state. We adopt POMDP to model the service composition problem in the partially observable environment. The service composition model based on the POMDP model is defined as follows.

Definition 2 (*POMDP-based Web Service Composition (POMDP-WSC)*)**.** A POMDP-WSC is an 8-tuple POMDP - WSC $= \langle S, A(s), O, P, R, \Omega, T, B\rangle$, where

- S is a finite set of world states including the initial state and final state.
- $A(s)$ is a set of actions under the state $s \in S$, which also stand for a set of web services under state s.
- O is a set which can be observed.
- P is the function of state transition. $P(s' \mid s, a)$ represents the probability from state s transferring to next state s' through invoking service $a \in A(s)$.
- R is the reward function. When a service $a \in A(s)$ is invoked along with the current state s transferring to the next state s', the environment will feed back immediate reward $r = R(s' \mid s, a)$ simultaneously.
- Ω is the observable information set of agent .
- T is the observation function of agent. It can be used to calculate the possible observation value on the next state s' after taking action a, which can be expressed by $Pr(o \mid s', a)$.
- B is the state space of agent. $b(s)$ can be used to describe the probability in state s.

According to POMDP-WSC, we model vacation planning using a state transition graph as shown in Fig. 6. It consists of two kinds of nodes. The hollow node represents a state node, such as s_0, which indicates the start of the traveling schedule. A node with double circles is terminal state, such as s_5. Another type which is the solid node is an action node, which represents the concrete service node. In each state, numerous services can be invoked, but the figure only depicts one node. The immediate reward r from the environment can be calculated by the services' aggregated QoS values [21].

5. Service composition based on deep reinforcement learning

This section focuses on the optimization method for adaptive service composition based on DRL.

5.1. Deep reinforcement learning based on RNN

Because RL relies on a Q-value table, each execution will update the corresponding table, which inevitably influences the efficiency in large-scale compositions. Thus, we adopt a neural network to estimate the Q-value, which serves as an online inspiration function to promote the learning of RL. To capture the temporal correlation of QoS attributes, an RNN is adopted. We further propose an Adaptive Deep Q-learning and RNN Composition Network (ADQRCN). Fig. 7 shows the basic structure, in which the input layer consists of state and action information, which is then passed through a hidden layer composed of 30 Long Short-Term Memory (LSTM) units and a full connection layer. Finally, the Q-value is generated from the output layer.

We adopt existing methods [27,45] for the training of ADQRCN. Compared with traditional Q-learning algorithm, we take advantage of the target network to relieve unstable phenomenon of a linear network and replay memory units to store samples. Moreover, some other techniques also involved, such as random sampling and dropout approaches, to train the neural network parameters from preventing over-fitting and high computational cost. ADQRCN simulates the Q function given by (11), where a mapping from a state–action pair to the value function is built by a function approximator f.

$$f(s, a; \theta) \approx Q^*(s, a) \tag{11}$$

It uses the Bellman Equation (12) to get the optimal action–value function, which is consistent that if the $Q(s', a')$ in the state s' was known for all the possible actions a', the optimal strategy is to select the action a' to maximize the $r + \gamma max_{a'} Q(s', a')$.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \tag{12}$$

Then, the loss function can be calculated by Eq. (13) and then updates the network parameters by gradient descent by following Eq. (14).

$$L = E[(r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2] \tag{13}$$

$$\frac{\partial L(\theta)}{\partial \theta} = E[(r + \gamma \max_{a'} Q(s', a') - Q(s, a))\frac{\partial Q(s, a; \theta)}{\partial \theta}] \tag{14}$$

Algorithm 1 describes the detailed process of training ADQRCN. In particular, the agent selects actions following an $\varepsilon-greedy$ policy, and the length of stored histories is fixed as N. In experience replay, the agent's experiences are stored and used at each time-step: $g_t = (s_t, a_t, r_t, s_{t+1})$ and then $D = \{g_1, g_2, \ldots, g_t\}$. In the inner loop, the random samples of experiences are applied to update the Q-learning value from D. For improving the stability of the algorithm further, a separate network $\hat{Q}$ is used for getting the objective y_j in the Q-learning value update by cloning the Q in every C steps.

To analyze the complexity of the algorithm, assume that the number of weights in the LSTM is W, the size of minibatch is m. Hence, the time complexity of one time step is $O(W \cdot m)$. By further assuming that the number of states is S, the number of the episodes is K, the overall complexity of the algorithm is $O(K \cdot S \cdot W \cdot m)$. By using the function approximation to simulate the Q-value, the DRL method can reduce the time cost to a certain extent, thus improve the efficiency.

5.2. Heuristic strategies

For the POMDP-WSC model, there may exist uncertainties in the environment, where the agent cannot directly distinguish the

Algorithm 1: ADQRCN Algorithm

Initialize replay memory D and its capacity N
Initialize action–value function Q with weights θ
Initialize target action–value function $\hat{Q}$ with random weights $\theta^- = \theta$
for each *episode* **do**
  for each t **do**
    With probability ε select a random action a_t
    Otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t, observe reward r_t and next state s_{t+1}
    $s_t = s_{t+1}$
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    if episode terminates at step $j+1$ **then**
      set $y_j = r_j$
    else
      $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$
    end if
    Perform a gradient descent step on $(y_j - Q(s_t, a_t; \theta))^2$ with respect to network parameters θ according the equation (14)
    Every C steps reset $\hat{Q} = Q$
  end for
end for

current state. Therefore, the belief state space plays a key role. However, using belief states to generate the accurate solution suffers from the scalability issue, making it not suitable for large-scale service composition. The main difficulty lies in updating all belief states. Furthermore, due to the continuity of the state space, it is difficult to compute the value function accurately. In fact, it may not be necessary to solve the problem based on the overall belief state space in the actual POMDP problem. We classify the states into two types according to the observability to take different behavior strategies.

5.2.1. State space classification

The states can be categorized into hidden and fully observable states. For example, consider three states and two actions. The value function of state–action pairs is given as

$$\begin{aligned} &V(s_0, a_0) = 5, V(s_0, a_1) = 4 \\ &V(s_1, a_0) = 5, V(s_1, a_1) = 4 \\ &V(s_2, a_0) = 0, V(s_2, a_1) = 10 \end{aligned} \tag{15}$$

According to the value function, we choose the strategy respectively:

$$\begin{aligned} &\pi^*(s_0) = a_0 \\ &\pi^*(s_1) = a_0 \\ &\pi^*(s_2) = a_1 \end{aligned}$$

Assume that the belief values of three states are $b = [0.3, 0.3, 0.4]$, then action a_0 will be selected because of the majority probability of 60%. The evaluation demonstrates that the choice of action a_0 is superior to action a_1 in states of s_0 and s_1, but with little gap. However, taking action a_0 may suffer from great loss. So taking into account the partially observable environment, we calculate the expectation of strategy with the belief state. The expectation of action a_0 is $0.3*5+0.3*5+0.4*0 = 3$, and action a_1 is $0.3*4+0.3*4+0.4*10 = 6.4$, from which we can conclude the expectation of action a_1 is obviously higher than action a_0. The example illustrates that the two type of states should adopt different strategies.

In a partially observable environment, such a situation may occur. Even though one action belongs to the optimal strategy, the agent may obtain the negative feedback information, which may cause the reduction of the action execution. This situation contradict our expectation. We give the definition of hidden and fully observable state as follows:

Definition 3 (*Hidden State*)**.** There are at least two optimal actions in one state, but the execution of these actions cannot obtain the stable feedback value. Such state is called hidden state, represented by $H(s)$.

Definition 4 (*Fully Observable State*)**.** If the execution of an action in one state has a stable feedback value, the state is fully observable, which is represented by $O(s)$. If one state exists two optimal actions, and the feedback value is always stable and positive, such a state is also a fully observable state.

In view of state classification, Ohta et al. proposed an entropy solution [50]. The method classified the state through calculating the entropy of each state. But the initial threshold of a state is difficult to be set reasonably. Therefore, the paper presents a classification solution based on the critical stable value of feedback to address how to distinguish hidden states from fully observable states.

First, when the action a is executed on state s, the immediate reward r will be returned as feedback and state will be transferred to next state s'. The feedback deviation is given by

$$\delta(s, a) = r + \gamma \max_{a'} Q(s', a') - Q(s, a) \tag{16}$$

If the feedback is positive, the action should be strengthened corresponding to update δ_+. Conversely, if the δ is negative, the value of δ_- will be updated.

$$\begin{cases} \delta_+(s, a) = \delta_+(s, a) + \delta(s, a) & \text{if } \delta(s, a) \geq 0 \\ \delta_-(s, a) = \delta_-(s, a) + |\delta(s, a)| & \text{if } \delta(s, a) < 0 \end{cases} \tag{17}$$

According to the above feedback deviation, we can calculate the stability of feedback by Eq. (18).

$$d(s, a) = \begin{cases} \dfrac{|\delta_+(s, a) - \delta_-(s, a)|}{\delta_+(s, a) + \delta_-(s, a)} & \text{if} \quad n(s, a) > N \\ 1 & otherwise \end{cases} \tag{18}$$

where $n(s, a)$ is the times that action a has been executed on state s. The stability of state is updated when the execution time is greater than N. It is necessary to experience sufficient exploration before the strategy of agent tends to be stable. Then, we calculate the observable degree $D(s)$ to differentiate the states according to Eq. (19).

$$D(s) = \min_a (d(s, a)) \tag{19}$$

If the state is a hidden state, at least one action feedback will fluctuate significantly and the observable degree will be low. In contrast, the observable degree of fully observable state should be high. So we define a simple filter to distinguish the two types of states as:

$$b(s) = \begin{cases} 1 & \text{if} \quad D(s) > \xi \\ 0 & otherwise \end{cases} \tag{20}$$

where ξ is a constant threshold ranged in (0,1), which represents the observable degree's threshold of a state. If a state is a hidden state, $b(s)$ is set to zero, and for a fully observable state, $b(s)$ is 1. A low observable degree threshold value may increase the classification error rate of hidden states. If some important hidden states cannot be detected, the learning task will consume more time. But if the ξ is set to 1, all states will be regarded as hidden states, which does not satisfy the real environment setting, which may affect the accuracy of the algorithm. If the number of hidden states is large, a higher ξ is needed to find as many hidden states as possible. Otherwise, a relatively small threshold may be required to reduce the error caused by noise data. Due to

the diversity of the real applications, ξ should be set according to the composition scenario and its environment. In the current paper, we set the value of ξ as 0.6 according to our experiment environment and service data.

5.2.2. The strategy for hidden states

The standard Q-learning algorithm follows the Markov Decision Process (MDP). Given a partially observable environment, an agent possibly gets the same observation under different environments. However, these states may need to execute different actions. Thus, choosing the appropriate strategy poses a key challenge in the partially observable environment.

It is conventional to solve the POMDP problem by using historical actions to decide the current action. The introduction of belief states converts the problem to an MDP problem based on the belief state space [51,52]. Belief state is an indication to express probability distribution of a state by making statistics about the historical complete information. However, the belief state method only applies to small-scale problems. When facing with a large-scale scenario, the main difficulty lies in the update of whole belief states with exponential complexity ($|A^{|O|}|$). In addition, the belief state space is continuous, leading to the difficulty of accurately computing the value function. To address these issues, we propose to use an RNN to simulate the value function. The solution attempts to establish the model by simulating the strategy space from historical experience.

The D_t^k indicates the former k steps of observable information until time t, namely the pairs of observable states and actions in POMDP, which can be represented as

$$D_t^k = (o_t, (o_{t-1}, a_{t-1}), (o_{t-2}, a_{t-2}), \ldots, (o_{t-k}, a_{t-k})) \tag{21}$$

Then, the strategy to hidden states is given by

$$\pi : B \times D^k \mapsto Pr(A) \tag{22}$$

where B is set of belief states and $Pr(A)$ is the probability distribution of actions.

In real-wold POMDP problems, agents lack experience to make decisions at the former k steps. The classic Q_{MDP} algorithm, shown as Eq. (23), is adopted to determine the strategy.

$$\pi^* = (b, d^k) = \begin{cases} \arg\max_{a'} \sum b(s)Q(s,a) & t \le k \\ \max_{a'} \Pr(A) & t > k \end{cases} \tag{23}$$

Along with the exploration of an agent and the accumulation of historical data, it can be realized to map belief states and history information at former k steps to one action. We integrate the LSTM network, where the input layer is the information of the belief states and history data (information of state–action pairs), indicated as a vector of N dimension and $x_{|N|}^t = (b(s_1), b(s_2)\ldots b(s_{|S|}), d_t^k)$. The probability distribution of an action is taken as the output layer $y = (pr(a_1), pr(a_2)\ldots pr(a_{|A|}))$, in which the largest probability action is executed.

5.2.3. The strategy for fully observable states

Different from hidden states, the feedback of executing actions is relatively stable for fully observable states, so that actions with positive feedback are strengthened. Hence, we can choose an optimal action through searching the value space directly. Undoubtedly, the table of Q-value stores the most important information, which may also suffer from the problem "curse of dimensionality". For example, there are 1000 fully observable states, each state corresponding to 100 candidate services. Then 100^{1000} pairs of state–action values need to be stored in the table. The scalability issue requires an approximation method to simulate the Q-values.

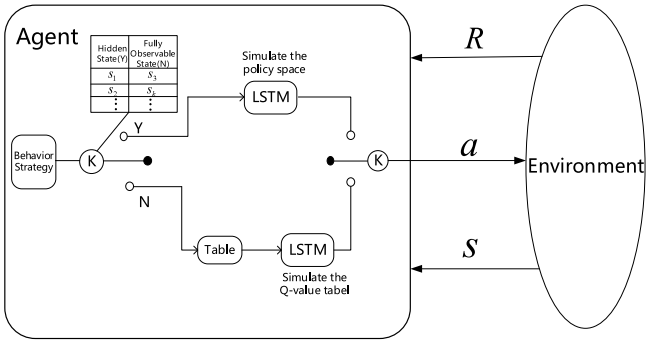


Fig. 8. The strategy model of OSON.

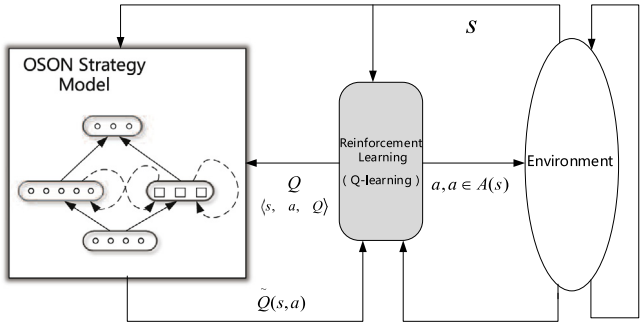


Fig. 9. The framework of OSON-WSC.

In addition, the change trend of each service is usually regular instead of completely stochastic. From the former performance of a service, the accessibility and success rate are relatively high and response time is short. The attributes of service may undergo changes because of the network condition. But the tendency is actually regular and corresponding attributes will follow the same trend. Hence, the change of Q-value is time correlated. Thus, the history information is significant, which should not be discarded. We propose to use an RNN as an approximation function in consideration of the temporal correlation, compensating for the defects of RL in large-scale scenarios.

For fully observable states, the structure and purpose of the neural network are different from hidden states. The neural network is primarily to generalize state–action pairs and their corresponding Q-values. For the structure of the RNN, the input layer is the current information and the output layer is the Q-value.

5.3. The framework of OSON-WSC

The strategy guides an agent to choose one action from numerous candidate services. For RL, the optimal strategy is to execute an action in order to maximize the accumulative reward. But the environment is partially observable in real-world, in which the feedback of hidden states executing the same action will be fluctuate. It is obvious that the strategy for an MDP model is not suitable in this case. Thus, a heuristic strategy model is developed, which corresponds to a One State One Network (OSON).

There are two neural networks, one for each type of states. Net_1 for hidden state information storage takes advantage of the belief state space samples and the performance of adopting the current value function to train the neural network. Meanwhile Net_2 for fully observable state simulates the Q-value function according to Q-values and state–action pairs. Fig. 8 describes the principle of OSON, where an agent needs to first judge the type

of state. If the state is a hidden one, the network consisting of LSTM units is used to simulate the strategy space. In contrast, the fully observable state attempts to generalize the Q-value function by using a neural network to improve efficiency in large-scale scenario. According to the OSON heuristic strategy model, a One State One Network for Web Service Composition (OSON-WSC) framework is constructed to address the large-scale and dynamic service composition problem (see Fig. 9).

The whole OSON-WSC framework is divided into three modules: OSON strategy, RL, and environment. The modules transmit the information mutually. First, the RL module interacts with the environment, which is different from the traditional RL because of adopting the OSON strategy to execute the optimal action. Moreover, the feedback from the environment will update the neural network in the OSON strategy model in return. When training the neural network, it not only generalizes the Q-value of the next state, but also influences other states' Q-values. The algorithm is detailed in Algorithm 2. At first, the agent conducts exploration learning for many times. The feedback deviation $\delta(s, a)$ is recorded according to Eq. (16). Then observable degree can be calculated according to Eqs. (17)–(19). Eq. (20) is used to determine the type of the state. If the state is a hidden one, ADQRCN is used to simulate and train the belief space, where the input is the belief state along with the historic information and the output is the probabilities of actions. If the state is a fully observable one, ADQRCN is used to approximate and train the Q-value, in which the input is the current information about the states and actions and the output is the Q-value.

Compared with the ADQRCN algorithm, the OSON-WSC uses the OSON strategy for optimization. In the beginning, the agent needs to consume some time for the state classification, which can be assumed as $O(t)$, and inside of the loop, just needs to execute one of the *if* and *else*. So the total time complexity can be represented as $O(t) + O(K_2 \cdot (W_h \cdot m \cdot S_h + W_f \cdot m \cdot S_f))$, where the K_2 is the number of episodes, S_h and S_f are the numbers of the hidden states and fully observable states, respectively, and $S_h + S_f = S$. W_h and W_f are the numbers of the weights in LSTM networks. Though OSON-WSC increases the time consumption of the initial explorations, the number of episodes is relatively reduced because it uses different strategies for different states to obtain the higher cumulative reward quickly. So the total time consumption is less than ADQRCN.

Algorithm 2: OSON-WSC Algorithm

```
Initialize two neural network structures and parameters (Net_1
training for the set of hidden states; Net_2 training for the set of fully
observable states)
The Agent conducts the exploration learning and feedback deviation
is recorded.
Calculate the observable degree and determine the type of the state.
repeat
  for Each episode do
    if state b(s) = 0 then
      Use the ADQRCN Algorithm to train neural network Net_1,
      execute a, then reach to the next state s'
    else
      Use the ADQRCN Algorithm to train neural network Net_2,
      execute a, then reach to the next state s'
    end if
    t = t + 1, s = s'
    until state s' is the terminal state
  end for
until the convergence condition is satisfied
```

6. Experiments and analysis

In this section, we present results of our experiments conducted with the two proposed approaches: ADQRCN, and the optimization method combined with the OSON behavior strategy, referred to as OSON-WSC. The purpose of the experiments is to show the validity of our solutions, which will be demonstrated from the following aspects: effectiveness and efficiency, adaptability, and scalability. In addition, we compare them with three competitive methods, QCN [21], the Multi-agent Q-learning (MA-Q) [53], and the Multi-agent SARSA (MA-SARSA) [4], and analyze the results.

It should be noted that the purpose of the methods proposed in the paper is for QoS-aware optimization. Thus, the value of the cumulative reward is the integrated QoS. Since a successful service composition leads to a valid service scheme, we use the optimal cumulative reward of an algorithm to represent the effectiveness, and the convergence speed is used to represent the efficiency. To demonstrate the adaptability of the algorithms, we change the QoS value of services randomly. The verification of the scalability is achieved by changing the number of service nodes and the number of candidate services.

6.1. Experimental settings

The purpose of service composition is to maximize users' satisfaction based on QoS attributes, which can be aggregated into the immediate reward of RL. The experimental data is from the QWS Dataset,[1] in which the data were gathered from the public sources on the Web, such as, UDDI registers and search engines. In the experiments, we mainly consider four types of QoS attributes, including *ResponseTime*, *Throughput*, *Availability* and *Reliability*. We adopted the same approach as in [23] to compute the average accumulative reward r according to the four types of QoS attributes above. We also further expand the QWS dataset in order to test the scalability of the proposed methods. According to the distribution of different attributes of QWS, some services are generated randomly. To be specific, the first step is to calculate the average of a set of candidate services attributes corresponding to each abstract service. Each attribute of services is set between 0.7 and 1. Then we randomly generate POMDP transition graph, where the number of candidate services of each abstract service is set according to different experimental requirements. Candidate services with same functional attributes and different QoS values are taken from the extended dataset. In the experiment results, unless specifically stated, the observable degree threshold value ξ is set to 0.6, the discount factor γ, the learning rate α, and the exploration factor ε of the reinforcement learning are set to 0.9, 0.6, and 0.6, respectively. The N in Eq. (18) is set to 50. Besides, the learning rate of the LSTM is set to 0.01, the number of the memory units is 30, the input layer and the output layer are set based on the actual service composition size, and the number of agents in Multi-agent algorithms is set to 4.

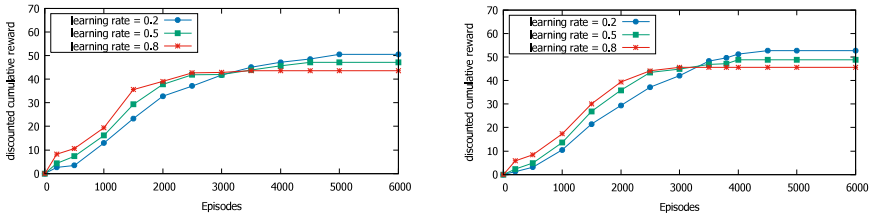
The experiments are conducted on the platform of Windows 7 (64bit), Intel i7-6700K 4.00 GHz CPU and 16 GB RAM, without adopting hardware acceleration means (such as GPU).

6.2. Result analysis

6.2.1. Influence of learning rate

In order to find the impact of the learning rate α on our approaches, we tested different α values, including 0.2, 0.5, and 0.8. We use 100 state nodes (abstract services) and each state node corresponds to 500 candidate services. The results of experiments are shown in Fig. 10. As can be seen, both ADQRCN and OSON-WSC are affected by the learning rate. When α is set to 0.2, 0.5, and 0.8 respectively, the convergence of the ADQRCN is at the 5000th, 4500th, and 3500th episode, respectively. The

[1] http://www.uoguelph.ca/~qmahmoud/qws/.



(a) Different Learning Rates for ADQRCN

(b) Different Learning Rates for OSON-WSC

Fig. 10. The influence of different learning rates.

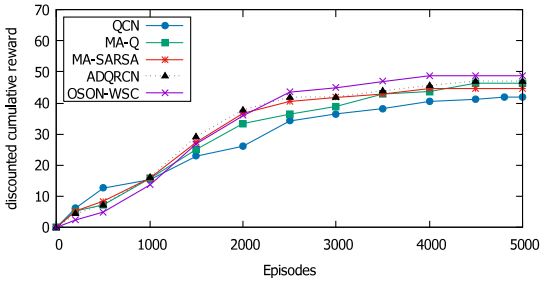


Fig. 11. The validation of effectiveness.

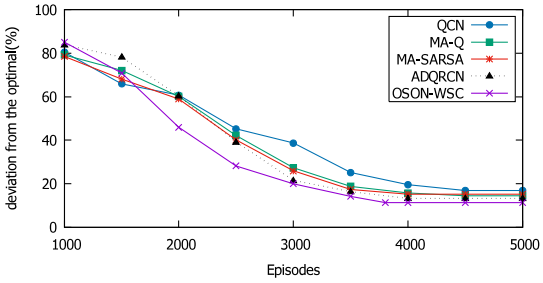


Fig. 12. Statistics Of the 100 group experiments.

optimal discounted cumulative reward values are 50.6, 47.1, and 43.6, respectively. As shown in Fig. 10(b), the convergence of the OSON-WSC is at the 4500th, 4000th, and 3000th episode respectively, and the optimal discounted cumulative reward values are 52.8, 48.9, and 45.7, respectively. These mean that the efficiency and the quality of the service composition are affected by the learning rate α. With the increase of α, the efficiency is improved while the composition quality is degraded. The reason is that if the learning rate is low, the algorithm will spend much time in searching the optimal solution. If the learning rate is high, it will reduce the search time and improve the search efficiency, but will fall into local optimum. Therefore, the tradeoff between efficiency and composition quality should be considered according to the scenario in practical applications. In our experiments below, α is fixed to 0.6, unless otherwise indicated.

6.2.2. Effectiveness and efficiency

In order to show the effectiveness of the proposed approaches, we test with 100 state nodes (abstract services) and each state node corresponding to 500 candidate services. This will result in 500^{100} possible compositions, corresponding to a quite large search space. As shown in Fig. 11, both ADQRCN and OSON-WSC are superior to other methods. They achieve the cumulative reward value 47.1 and 48.9 respectively, which are higher than those from QCN, MA-Q, and MA-SARSA. However, our methods do not perform well in the initial stage. With the accumulation of training samples, the learning efficiency is significantly improved. For the convergence of the algorithms, QCN converges at the 4800th episode, MA-Q, and MA-SARSA converge at around the 4500th and 4000th episode, respectively. ADQRCN and OSON-WSC converge at around the 4500th and 4000th episode, respectively, which demonstrate that the efficiency of the proposed methods are comparable to MA-Q and MA-SARSA. Since QCN, MA-Q, and MA-SARSA are based on the table storage with random exploration, they are outperformed by ADQRCN due to the generalization expression, which makes the latter more suitable for large-scale service composition. Moreover, OSON-WSC adopts a heuristic strategy and achieves a better performance in terms of effectiveness and efficiency than ADQRCN.

To further prove the validity of the proposed approaches, we evaluate the deviation of the cumulative reward from the most optimized composition schema, which is presented as D given by Eq. (24), where the OPR represents optimal convergence value, and the CRR represents the convergence value of the algorithm. The scenario is stochastically generated as 100 groups of service compositions, where the number of abstract services is set to 100 and each abstract service corresponds to 500 candidate services. After calculating the deviation value D of 100 groups of experiments, the average is computed. From Fig. 12, all methods ultimately achieve convergence, obtaining the nearly optimal composition schema. The deviations of QCN, MA-Q, and MA-SARSA after 100 groups of experiments are 16.9%, 14.5%, and 15.2%, respectively, while ADQRCN and OSON-WSC are about 13.1% and 11.3%, respectively. ADQRCN and OSON-WSC outperform other methods in terms of the deviation from the optimal composition schema. Moreover, between ADQRCN and OSON-WSC, the latter achieves better results than the former.

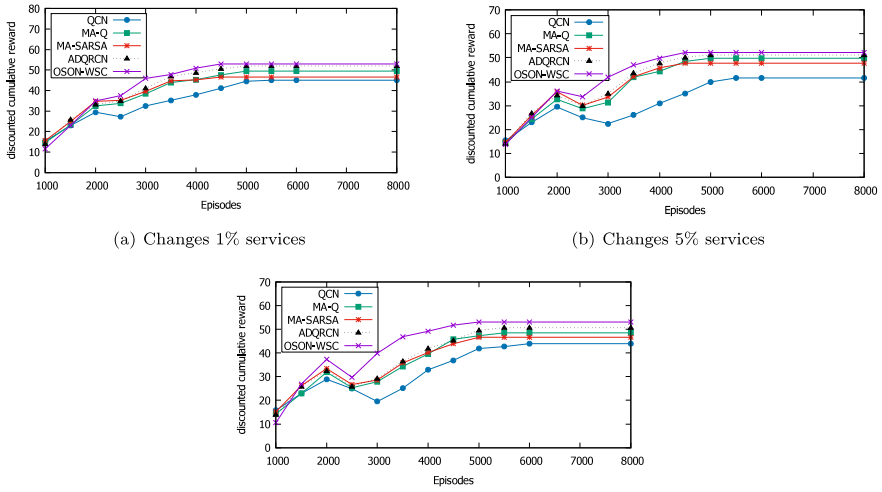
$$D = \frac{OPR - CRR}{OPR} \tag{24}$$

In sum, this experiment verifies the effectiveness and efficiency of ADQRCN and OSON-WSC. It also validates that the heuristic behavior strategy improves the efficiency and effectiveness of service composition.

6.2.3. Adaptability

In a software system, adaptability is defined as the behavior to adjust according to the changes of the operating environment [54]. In order to verify the inherent adaptability of our algorithms, we simulate a dynamic environment by randomly changing the QoS values.

We still set 100 state nodes and 500 candidate services corresponding to each state node. To simulate the dynamic environment, we change 1%, 5% and 10% QoS values of services in a fixed period of time (between the 2000th episode to the 2500th episode). The condition of network, the evolution of services themselves and other factors may all change in a real service



(a) Changes 1% services

(b) Changes 5% services

(c) Changes 10% services

Fig. 13. The validation of adaptability.

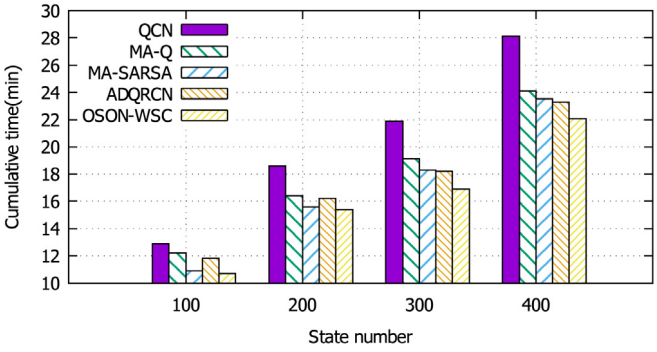


Fig. 14. The influence of different numbers of state nodes.

composition scenario. We assume no prior knowledge of the environment so the service information can only be obtained from the exploration of the agent. Because of the environmental change, it undoubtedly impacts the result of learning and the original optimal strategy. The agent needs to learn part of environment and explores a new solution of service composition.

The three groups of experiments are shown in Fig. 13. The fluctuation of services has certain influence on the learning performance, but these effects are temporary. When changing 1% QoS attributes, we find that it does not bring too much influence to the Q-value. Among these methods to be compared, QCN suffers a lot because services in the most optimal solution change during the fluctuation, while the others have stronger adaptability. With the increasing proportion of changing services (5% to 10% services), the change of environment has gradually affected the performance of the algorithms. From the figure, there is a great fluctuation in the cumulative reward. OSON-WSC achieves slightly better result than ADQRCN by using a shorter time to adjust the behavior strategy. Other methods also show the ability to adjust, but the performances are worse than OSON-WSC. It may be due to the heuristic strategy and prediction characteristic of the neural network.

In sum, the experiments simulate a dynamic scenario, which verifies the adaptability of ADQRCN and OSON-WSC and proves that they provide more reliable service composition schema.

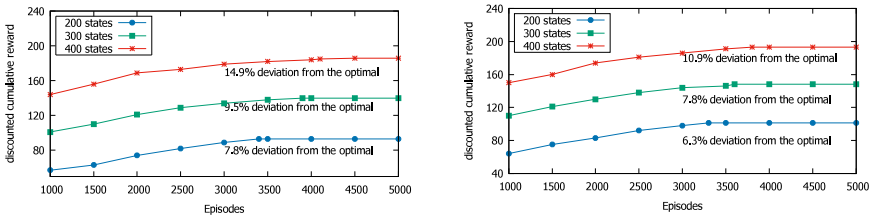
6.2.4. Scalability

To validate the scalability of the proposed algorithms, we extend the scale of service compositions, from two aspects to simulate large-scale scenarios: increasing the number of state nodes and candidate services.

First, the total number of candidate services is fixed at 50000 and the state node number is set to 100, 200, 300 and 400. In addition, candidate services are randomly assigned to each state node. Fig. 14 shows the experimental result. More state nodes will lead to a longer convergence time. This is because with the increase of state node number, the functional structure of the composition becomes more complicated. As shown in Fig. 14, the computation time of QCN dramatically increases from 12.9 min corresponding to 100 state nodes to 28.1 min with 400 state nodes. The other algorithms also exhibit a growing trend. With the increase of the state nodes, ADQRCN and OSON-WSC perform better gradually, and OSON-WSC achieves a faster convergence than ADQRCN.

Next, the number of state node varies from 100 to 400 and candidate services corresponding to each state node is fixed at 500. The deviation degree is also adopted to evaluate the service composition schema obtained from the learning. Fig. 15 clearly shows the good scalability of OSON-WSC faced with the increasing state nodes. The deviation of ADQRCN is 7.8% when the state number is 200, more than OSON-WSC's 6.3%. For the convergence time, OSON-WSC converges at about the 3300th, 3600th and 3800th episode, compared with ADQRCN at about the 3500th, 3900th and 4500th episode.

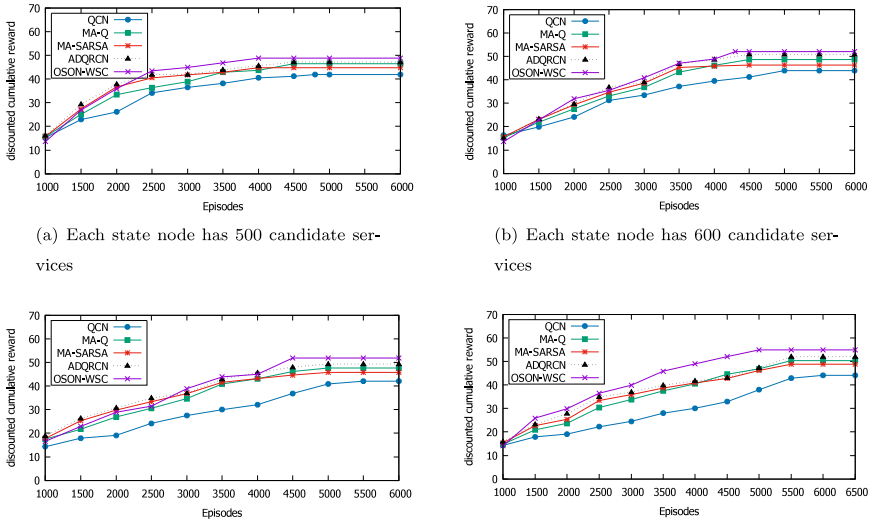
Finally, the number of state nodes is fixed at 100, and candidate services vary from 500 to 800. The number of total candidate services is up to 80000 and the number of composition schemata grows from 500^{100} to 800^{100}, which can be considered to be a fairly large scenario. As shown in Fig. 16, the expansion of candidate services influences the convergence time and discount cumulative reward. From the figure, all the methods use longer time to achieve convergence. QCN uses approximately 4800th, 5000th and 5500th episodes to converge. It exceeds the 6000th episode when the candidate services are up to 800. The convergence time of the other algorithms is better than QCN. Moreover, with the increase of candidate services, the performance of ADQRCN and OSON-WSC are better than that of MA-Q and MA-SARSA. This is mainly because ADQRCN and OSON-WSC adopt the neural network as the generalization value function, which helps them maintain strong ability of generalization and quickly achieve convergence. However, the increase of candidate services does not necessarily improve the optimal composition result. After all, the solution depends on the QoS and the new services are not necessarily superior to the original services.



(a) Influence of Different Numbers of State Nodes to ADQRCN

(b) Influence of Different Numbers of State Nodes to OSON-WSC

Fig. 15. The validation of adaptability.



(a) Each state node has 500 candidate services

(b) Each state node has 600 candidate services

(c) Each state node has 700 candidate services

(d) Each state node has 800 candidate services

Fig. 16. Influence of different numbers of candidate services.

To sum up, the experiments validate the scalability of the proposed methods, when facing with increasing services and more complicate service composition structures. The proposed methods are able to eventually converge to the optimal (or near-optimal) result and show better performance.

6.2.5. Significance test

In statistics, significance test is one of the statistical hypothesis testing methods used to test whether there are differences between the experimental and control groups in the scientific experiments and whether the differences are significant. Therefore, we use this method to evaluate the difference between the proposed methods and the other compared reinforcement learning methods for service composition. Significance test can be divided into two types: parameter test and nonparametric test. Parameter test requires the samples to obey the normal distribution, and the nonparametric test is used when the data does not satisfy the normality and variance assumptions. According to the characteristics of the parameters, we adopt a nonparametric test method, Wilcoxon signedrank test.[2] We have conducted a significance test on ADQRCN and OSON-WSC, respectively, by comparing with the QCN, MA-Q, and MA-SARSA. We first propose the hypotheses, where H_0 is that "the two methods have significant differences in performance (quality of composition results)", and H_1 is that "the two methods have no obvious difference in performance". We conduct 15 experiments based on the three methods, and record the cumulative reward values. The statistics of the cumulative reward obtained by QCN, MA-Q, MA-SARSA, ADQRCN, and OSON-WSC are shown in Tables 3, 4, and 5.

sgn is a symbolic function, which represents the positive and negative of the difference between the two sets of data; *abs* and *Rank* represent the absolute value of the difference and the sorting according to the difference respectively; $sgn \cdot Rank$ is a sorting based on symbols. In Table 3, 13 groups of *sgn* are positive and 2 groups are negative for QCN and ADQRCN, and the same for QCN and OSON-WSC. In Table 4, 12 groups of *sgn* are positive for MA-Q and ADQRCN, and 13 groups of *sgn* are positive for MA-Q and OSON-WSC. In Table 5, 13 groups of *sgn* are positive for MA-SARSA and ADQRCN, which is the same as the MA-SARSA and OSON-WSC. These results verify that the performance of ADQRCN and OSON-WSC are all different from the others obviously. According to the Wilcoxon signedrank test, we can calculate the test statistic: $W = \sum_{i=1}^{N} [sgn_i \cdot Rank_i]$, and the values are 112 and 114 in Table 3, 108 and 112 in Table 4, 104 and 112 in Table 5, respectively. Referring to the Wilcoxon critical value table, when the level of significance was 0.05, the critical value of the two-sided test of the 15 groups was 25, which was obviously less than our statistic results. So the hypothesis H_0 is accepted. It also means that there is a distinct difference between the proposed methods and the other algorithms. The above significance experiments demonstrate that the performance of ADQRCN and OSON-WSC proposed in this paper is obviously better than the

[2] https://en.wikipedia.org/wiki/Wilcoxon_signedrank_test/.

Table 3
Significance test of ADQRCN and OSON-WSC compared with QCN.

i	$QCN\ (x_{1,i})$	$ADQRCN(x_{2,i})$	$OSON-WSC(x_{3,i})$	$x_{2,i}-x_{1,i}$				$x_{3,i}-x_{1,i}$			
				sgn	abs	R_i	$sgn\cdot R_i$	sgn	abs	R_i	$sgn\cdot R_i$
1	41.9	47.1	48.9	1	5.2	10	10	1	7.0	11	11
2	43.1	45.9	49.2	1	2.8	6	6	1	6.1	7	7
3	42.5	46.9	46.9	1	4.4	7	7	1	4.2	4	4
4	44.1	44.9	43.9	1	0.8	2	2	−1	0.2	1	−1
5	45.6	44.3	47.3	−1	1.3	3	−3	1	1.7	3	3
6	43.9	46.5	49.1	1	2.6	4	4	1	5.2	5	5
7	39.9	47.2	48.2	1	7.3	15	15	1	8.3	15	15
8	42.1	46.7	49.7	1	4.6	8	8	1	7.6	12.5	12.5
9	40.5	47.3	46.3	1	6.8	14	14	1	5.8	6	6
10	45.1	44.9	44.5	−1	0.2	1	−1	−1	0.6	2	−2
11	39.7	46.2	47.3	1	6.5	12	12	1	7.6	12.5	12.5
12	40.8	47.5	48.6	1	6.7	13	13	1	7.8	14	14
13	42.6	45.3	49.3	1	2.7	5	5	1	6.7	8.5	8.5
14	43.4	48.1	50.1	1	4.7	9	9	1	6.7	8.5	8.5
15	41.6	47.5	48.5	1	5.9	11	11	1	6.9	10	10

Table 4
Significance test of ADQRCN and OSON-WSC compared with MA-Q.

i	$MA-Q\ (x_{1,i})$	$ADQRCN(x_{2,i})$	$OSON-WSC(x_{3,i})$	$x_{2,i}-x_{1,i}$				$x_{3,i}-x_{1,i}$			
				sgn	abs	R_i	$sgn\cdot R_i$	sgn	abs	R_i	$sgn\cdot R_i$
1	46.4	47.1	48.9	1	0.7	5	5	1	2.5	7	7
2	45.1	45.9	49.2	1	0.8	6	6	1	4.1	14	14
3	45.1	46.9	46.9	1	1.8	13	13	1	1.8	4	4
4	43.4	44.9	43.9	1	1.5	11	11	1	0.5	2	2
5	44.7	44.3	47.3	−1	0.4	3	−3	1	2.6	8	8
6	44.8	46.5	49.1	1	1.7	12	12	1	4.3	15	15
7	45.8	47.2	48.2	1	1.4	10	10	1	2.4	6	6
8	45.7	46.7	49.7	1	1	8	8	1	4	12	12
9	46.6	47.3	46.3	1	0.7	4	4	−1	0.3	1	−1
10	45.1	44.9	44.5	−1	0.2	1.5	−1.5	−1	0.6	3	−3
11	43.2	46.2	47.3	1	3	15	15	1	4.1	13	13
12	45.5	47.5	48.6	1	2	14	14	1	3.1	10	10
13	45.5	45.3	49.3	−1	0.2	1.5	−1.5	1	3.8	11	11
14	47.2	48.1	50.1	1	0.9	7	7	1	2.9	9	9
15	46.3	47.5	48.5	1	1.2	9	9	1	2.2	5	5

Table 5
Significance test of ADQRCN and OSON-WSC compared with MA-SARSA.

i	$MA-SARSA\ (x_{1,i})$	$ADQRCN(x_{2,i})$	$OSON-WSC(x_{3,i})$	$x_{2,i}-x_{1,i}$				$x_{3,i}-x_{1,i}$			
				sgn	abs	R_i	$sgn\cdot R_i$	sgn	abs	R_i	$sgn\cdot R_i$
1	45.8	47.1	48.9	1	1.3	6	6	1	3.1	5	5
2	45.3	45.9	49.2	1	0.6	2	2	1	3.9	9	9
3	44.4	46.9	46.9	1	2.5	9	9	1	2.5	4	4
4	46.1	44.9	43.9	−1	1.2	5	−5	−1	2.2	3	−3
5	42.8	44.3	47.3	1	1.5	8	8	1	4.5	10	10
6	43.3	46.5	49.1	1	3.2	12	12	1	5.8	13	13
7	42.3	47.2	48.2	1	4.9	15	15	1	5.9	14	14
8	43.6	46.7	49.7	1	3.1	11	11	1	6.1	15	15
9	46.1	47.3	46.3	1	1.2	4	4	1	0.2	2	2
10	44.6	44.9	44.5	1	0.3	1	1	−1	0.1	1	−1
11	42.5	46.2	47.3	1	3.7	13	13	1	4.8	11	11
12	44.7	47.5	48.6	1	2.8	10	10	1	3.9	8	8
13	46.1	45.3	49.3	−1	0.8	3	−3	1	3.2	6	6
14	46.7	48.1	50.1	1	1.4	7	7	1	3.4	7	7
15	43.5	47.5	48.5	1	4	14	14	1	5	12	12

compared algorithms, and they have a great advantage in the service composition solutions.

7. Conclusion and future work

This paper presents an adaptive deep reinforcement learning framework for large-scale service composition problem. The framework models the service composition problem using POMDP-WSC and integrates the aggregated QoS into the reward function. The proposed framework allows the integration of the perception ability of DL with the decision making ability of reinforcement learning. In addition, we optimize the behavior decision to further improve the efficiency and accuracy of the composition solution. The main contributions of the paper are summarized as follows:

- To address the limitation of RL, we integrate the perception ability of DL to solve large-scale service composition problems. Furthermore, the ADQRCN method adopts the recurrent neural network to generalize the value function, which not only improves the efficiency of storage and computation but also enhances the prediction accuracy.

- We propose the POMDP-WSC model, which is closer to the real service composition problem and suitable for the large-scale scenario. The POMDP takes into account the behavior strategy in the partially observable states.
- According to the POMDP-WSC model, we propose the OSON-WSC framework based on a heuristic behavior strategy. The states select different behavior strategies respectively according to the type of states, resulting in the improvement of the whole composition performance.

We identify the following directions as our future work to further improve the proposed approaches:

- On the reinforcement learning side, hierarchical reinforcement learning technology can be used to solve the more complex scenarios. On the deep learning side, we plan to consider DQN based on competitive framework [55] and Deep Double Q-network(DDQN) [56].
- The RNN that generalizes the belief state information may still face very complex input. We will try to further optimize the neural network by taking advantage of the Convolutional Neural Network (CNN) to pool the samples.
- Although the application of deep reinforcement learning is promising, the theoretical underpinning can be further strengthened, such as the proof of convergence.
- The scale of QWS does not satisfy our requirements for large-scale service composition. We plan to collect more data from real services to replace the simulation experiments.

Acknowledgments

This work was partially supported by National Key Research and Development Program of China (No. 2018YFB1003800) and NSFC Projects (Nos. 61672152, 61232007, 61532013), Collaborative Innovation Centers of Novel Software Technology and Industrialization and Wireless Communications Technology, China. Qi Yu's work was supported in part by a US NSF IIS award IIS-1814450 and a US ONR award N00014-18-1-2875. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agency.

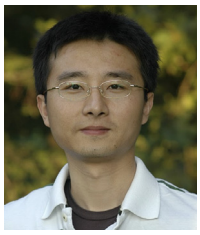
References

[1] I. Trummer, B. Faltings, Optimizing the tradeoff between discovery, composition, and execution cost in service composition, in: Web Services (ICWS), 2011 IEEE International Conference on, IEEE, 2011, pp. 476–483.
[2] G. Canfora, M. Di Penta, R. Esposito, M.L. Villani, A framework for qos-aware binding and re-binding of composite web services, J. Syst. Softw. 81 (10) (2008) 1754–1769.
[3] W. Li, Y. Badr, F. Biennier, Service farming: an ad-hoc and qos-aware web service composition approach, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, 2013, pp. 750–756.
[4] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, A. Bouguettaya, Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition, in: International Conference on Service-Oriented Computing, Springer, 2014, pp. 154–168.
[5] N.B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, V. Issarny, Qos–aware service composition in dynamic service oriented environments, in: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2009, pp. 123–142.
[6] S.M. Sohan, A. Craig, F. Maurer, A case study of web api evolution, in: IEEE World Congress on Services, IEEE, 2015, pp. 245–252.
[7] I. Constantinescu, B. Faltings, W. Binder, Large scale, type-compatible service composition, in: Proceedings of the IEEE International Conference on Web Services(ICWS), IEEE, 2004, pp. 506–513.
[8] S.-C. Oh, D. Lee, S.R. Kumara, Effective web service composition in diverse and large-scale service networks, IEEE Trans. Serv. Comput. 1 (1) (2008) 15–32.
[9] H. Wang, G. Huang, Q. Yu, Automatic hierarchical reinforcement learning for efficient large-scale service composition, in: Web Services (ICWS), 2016 IEEE International Conference on, IEEE, 2016, pp. 57–64.
[10] D. Yu, Y. Lei, Z. Bin, Qos-driven self-healing web service composition based on performance prediction, J. Comput. Sci. Technol. 24 (2) (2009) 250–261, http://dx.doi.org/10.1007/s11390-009-9221-8.
[11] Q. Liu, Y. Sun, S. Zhang, A scalable web service composition based on a strategy reused reinforcement learning approach, in: Web Information Systems and Applications Conference (WISA), 2011 Eighth, IEEE, 2011, pp. 58–62.
[12] H. Wang, M. Gu, Q. Yu, H. Fei, J. Li, Y. Tao, Large-scale and adaptive service composition using deep reinforcement learning, in: International Conference on Service-Oriented Computing, Springer, 2017, pp. 383–391.
[13] Y. Yan, P. Poizat, L. Zhao, Repairing service compositions in a changing world, in: Software Engineering Research, Management and Applications 2010, Springer, 2010, pp. 17–36.
[14] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, IEEE Trans. Software Eng. 33 (6) (2007) 369–384, http://dx.doi.org/10.1109/TSE.2007.1011.
[15] Y. Yan, P. Poizat, L. Zhao, Self-adaptive service composition through graph-plan repair, in: Web Services (ICWS), 2010 IEEE International Conference on, IEEE, 2010, pp. 624–627.
[16] S. Beauche, P. Poizat, Automated service composition with adaptive planning, in: Service-Oriented Computing–ICSOC 2008, Springer, 2008, pp. 530–537.
[17] A.S. da Silva, H. Ma, M. Zhang, Genetic programming for qos-aware web service composition and selection, Soft Comput. 20 (10) (2016) 3851–3867.
[18] O. Hammas, S.B. Yahia, S.B. Ahmed, Adaptive web service composition insuring global qos optimization, in: 2015 International Symposium on Networks, Computers and Communications (ISNCC), IEEE, 2015, pp. 1–6.
[19] Y. Huo, Y. Zhuang, J. Gu, S. Ni, Y. Xue, Discrete gbest-guided artificial bee colony algorithm for cloud service composition, Appl. Intell. 42 (4) (2015) 661–678.
[20] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: A survey, J. Artificial Intelligence Res. (1996) 237–285.
[21] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, A. Bouguettaya, Adaptive service composition based on reinforcement learning, in: International Conference on Service-Oriented Computing, Springer, 2010, pp. 92–107.
[22] A. Moustafa, M. Zhang, Multi-objective service composition using reinforcement learning, in: Service-Oriented Computing, Springer, 2013, pp. 298–312.
[23] H. Wang, X. Wang, X. Zhang, Q. Yu, X. Hu, Effective service composition using multi-agent reinforcement learning, Knowl.-Based Syst. 92 (2016) 151–168.
[24] H. Wang, X. Wang, X. Hu, X. Zhang, M. Gu, A multi-agent reinforcement learning approach to dynamic service composition, Inform. Sci. 363 (2016) 96–119.
[25] H. Wang, X. Chen, Q. Wu, Q. Yu, X. Hu, Z. Zheng, A. Bouguettaya, Integrating reinforcement learning with multi-agent techniques for adaptive service composition, ACM Trans. Auton. Adapt. Syst. 12 (2) (2017) 8:1–8:42, http://dx.doi.org/10.1145/3058592.
[26] S. Lange, M. Riedmiller, Deep auto-encoder neural networks in reinforcement learning, in: Neural Networks (IJCNN), the 2010 International Joint Conference on, IEEE, 2010, pp. 1–8.
[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602.
[28] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[29] A. Moustafa, T. Ito, A deep reinforcement learning approach for large-scale service composition, in: International Conference on Principles and Practice of Multi-Agent Systems, Springer, 2018, pp. 296–311.
[30] Z. Daphney-Stavroula, D. Ugur, M. Urbashi, A pomdp approach for active collision detection via networked sensors, in: 2016 50th Asilomar Conference on Signals, Systems and Computers, 2016, pp. 1697–1701.
[31] W. Kyle Hollins, Z. Shlomo, A pomdp formulation of proactive learning, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016, pp. 3202–3208.
[32] Y. Lei, Z. Jiantao, W. Fengqi, G. Yongqiang, Y. Bo, Web service composition based on reinforcement learning, in: Web Services (ICWS), 2015 IEEE International Conference on, IEEE, 2015, pp. 731–734.
[33] H. Wang, X. Wang, Q. Yu, Optimal self-healing of service-oriented systems with incomplete information, in: Big Data (BigData Congress), 2013 IEEE International Congress on, IEEE, 2013, pp. 227–234.
[34] D.E. Goldberg, J.H. Holland, Genetic algorithms and machine learning, Mach. Learn. 3 (2) (1988) 95–99.
[35] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[36] A. Karpathy, J. Johnson, L. and Fei-Fei, Visualizing and understanding recurrent networks, CoRR abs/1506.02078. arXiv:1506.02078.
[37] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.

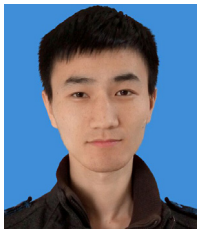
[38] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, T. Darrell, Long-term recurrent convolutional networks for visual recognition and description, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2625–2634.
[39] C. Goller, A. Kuchler, Learning task-dependent distributed representations by backpropagation through structure, in: Neural Networks, 1996., IEEE International Conference on, Vol. 1, IEEE, 1996, pp. 347–352.
[40] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[41] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: Acoustics, Speech and Signal Processing (Icassp), 2013 Ieee International Conference on, IEEE, 2013, pp. 6645–6649.
[42] K. Yu, L. Jia, Y. Chen, W. Xu, Deep learning: yesterday, today, and tomorrow, J. Comput. Res. Dev. 50 (9) (2013) 1799–1804.
[43] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117.
[44] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, Adaptive Computation and Machine Learning, MIT Press, 1998.
[45] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[46] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, CoRR abs/1509.02971. arXiv:1509.02971.
[47] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: Proceedings of the 33rd International Conference on Machine Learning (ICML), 2016.
[48] J. Oh, X. Guo, H. Lee, R.L. Lewis, S. Singh, Action-conditional video prediction using deep networks in atari games, in: Advances in Neural Information Processing Systems, 2015, pp. 2863–2871.
[49] J.C. Caicedo, S. Lazebnik, Active object localization with deep reinforcement learning, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2488–2496.
[50] M. Ohta, Y. Kumada, I. Noda, Using suitable action selection rule in reinforcement learning, in: Systems, Man and Cybernetics, 2003. IEEE International Conference on, Vol. 5, IEEE, 2003, pp. 4358–4363.
[51] A.L. Strehl, M.L. Littman, An empirical evaluation of interval estimation for markov decision processes, in: Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on, IEEE, 2004, pp. 128–135.
[52] D.S. Bernstein, E.A. Hansen, S. Zilberstein, Bounded policy iteration for decentralized pomdps, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 52–57.
[53] H. Wang, Q. Wu, X. Chen, Q. Yu, Z. Zheng, A. Bouguettaya, Adaptive and dynamic service composition via multi-agent reinforcement learning, in: Proceedings of the IEEE International Conference on Web Services (ICWS), IEEE, 2014, pp. 447–454.
[54] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, A.L. Wolf, An architecture-based approach to self-adaptive software, IEEE Intell. Syst. 14 (3) (1999) 54–62.
[55] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, Dueling network architectures for deep reinforcement learning, CoRR abs/1511.06581. arXiv:1511.06581.
[56] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: AAAI, 2016, pp. 2094–2100.

Hongbing Wang is a professor from School of Computer Science and Engineering, Southeast University, China. He received his Ph.D. in computer science from Nanjing University, China. His research interests include Service Computing, Cloud Computing, and Software Engineering. He published more than fifty refereed papers in international conferences and Journals, e.g., Journal of Web Semantics, JSS, TSC, ICSOC, ICWS, SCC, CIKM, ICTAI, WI, etc. He is a member of the IEEE.
E-mail: hbw@seu.edu.cn

Mingzhu Gu is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. Her research focuses on service selection and service composition.



Qi Yu received the PhD degree in computer science from Virginia Polytechnic Institute and State University (Virginia Tech). He is an associate professor in the College of Computing and Information Sciences at the Rochester Institute of Technology. His current research interests lie in the areas of applied machine learning, data mining, and service computing. He has published over 80 papers, many of which appeared in top-tier venues in these fields. He is a member of the IEEE.
E-mail: qi.yu@rit.edu



Yong Tao is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. His research focuses on service selection and service composition.

Jiajie Li is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. His research focuses on service selection and service composition.

Huanhuan Fei is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. His research focuses on service selection and service composition.

Jia Yan is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. His research focuses on service selection and service composition.

Wei Zhao is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. His research focuses on service selection and service composition.

Tianjing Hong is a postgraduate student in the School of Computer Science and Engineering, Southeast University, P.R China. Her research focuses on service selection and composition and service prediction.