A Real-Time Subtask-Assistance Strategy for Adaptive Services Composition

Li QUAN†(a), Member, Zhi-liang WANG†, and Xin LIU†, Nonmembers

SUMMARY  Reinforcement learning has been used to adaptive service composition. However, traditional algorithms are not suitable for large-scale service composition. Based on Q-Learning algorithm, a multi-task oriented algorithm named multi-Q learning is proposed to realize subtask-assistance strategy for large-scale and adaptive service composition. Different from previous studies that focus on one task, we take the relationship between multiple service composition tasks into account. We decompose complex service composition task into multiple subtasks according to the graph theory. Different tasks with the same subtasks can assist each other to improve their learning speed. The results of experiments show that our algorithm could obtain faster learning speed obviously than traditional Q-learning algorithm. Compared with multi-agent Q-learning, our algorithm also has faster convergence speed. Moreover, for all involved service composition tasks that have the same subtasks between each other, our algorithm can improve their speed of learning optimal policy simultaneously in real-time.

key words: service composition, reinforcement learning, Q-learning, subtask-assistance

1. Introduction

With the development of Internet of Things (IOT) [1], more and more smart devices came into our lives, such as smart phone, smart sensors and so on. At present, most of these devices are private which can be used only by their owners. But, it’s a word with information sharing in future. How can we exploit these smart devices to live a more intelligent life in the world? Service-oriented Architecture (SOC) [2] represents a new generation of distributed computing architecture. Data resources, software resources and hardware resources can be utilized in the form of Web Services. SOC has become the most important architecture for distributed applications and provide a variety of cloud services. Web service are the main components of modular applications, it has an open, standardized interface (UDDI, Universal Description Discovery and Integration) [3]. Web service is the main technology to implement service-oriented architecture. Web services [4] are conducive to building cloud based flexible, loosely coupled application. These technologies provide us with a good way to exploit smart devices. Devices are connected to the cloud platform, one device could correspond to one or more web services, and we can use them by invoking the corresponding services. We call this pattern as smart devices as a service (SDaaS). Generally speaking, a single web service can’t satisfy the needs of complex applications, we can achieve it by combining a number of specific web services, and web service composition (WSC) [5] has become more and more important.

Many studies have focused on the optimization of web service composition, and most of the web service composition methods are based on static environment [6]–[10]. The usual way of web service composition is to construct an abstract composition service for meeting the user’s functional requirements firstly, and then select specific web services from candidate services based on the user’s nonfunctional requirements. Finally, an optimized web service composition could be created. In this way, the composed service runs in a static workflow mode, it does not adapt itself to changes in the environment automatically. Once some of the services which are concluded in the composed service change (QoS, deleted or invalid), it needs to be adjusted to adapt to the new changes artificially. Most of the web services are changing constantly in the network. Therefore, an adaptive approach for web service composition is needed.

Researchers have applied reinforcement learning to the web service composition problem, this method enable service composition to adapt to environment autonomously. Modeling the web service composition problem as a Markov decision process (MDP) [11]–[13], a variety of alternative services and work flows can be included in a single web service composition process, the optimization of web service composition can be conducted by using reinforcement learning at runtime. Q-Learning [14] is a classical algorithm for reinforcement learning, it is suitable for adaptive web service composition. Papers [15], [16] are to address the adaptive problem of service composition when facing high dynamic web services. But it will take a long time to find optimal policy when the number of services continues to increase. Multi-agent is applied to service composition to solve this problem. The paper [17] integrates on-policy reinforcement learning with multi-agent to achieve adaptive service composition for large number of services. By interacting with more participant agents, paper [18] presents a service composition method, which outperforms other learning methods. The Paper [19] focuses on one task that has a simple hierarchical structure. The task can be decomposed into hierarchical subtasks and most of the subtasks are primitive actions. The MAXQ-Q learning algorithm is proposed to learn optimal policy for a single task. The paper [20]
applies multi-agent to hierarchical task in order to speed up the acquisition of cooperative multi-agent tasks. Both of the two papers are interested in tasks with hierarchical structure. In current paper, we proposed multi-Q learning algorithm based on task decomposition. However, the advantage of this algorithm lies in solving the problem of multiple service composition tasks and these tasks are decomposed into several independent subtasks with no hierarchy. We focus on the same subtasks belonging to different service composition tasks, which contributes to each other in order to find optimal policy faster.

Actually, because of the popularity of the Internet of things and cloud computing, there will be more and more composition tasks on the server side at the same time. All of service composition tasks will experience the process of learning optimal policy. The existing research mostly focus on one task to make a better performance and don’t take other service composition tasks into account. This paper presents a real-time subtask-assistance strategy to improve learning speed of all involved service composition tasks by exchanging experiences between them. First, a service composition task is decomposed into several subtasks, and the same subtasks can be included in different service composition tasks. Secondly, based on Q-Learning algorithm, a multi-task oriented algorithm: multi-Q learning is proposed to make exchanging experiences between different service composition tasks possible. Finally, a subtask-assistance strategy for adaptive service composition is realized by exchanging learning experience between the same subtasks from different service composition tasks.

This paper is organized according to the following structure: In Sect. 2, we introduce the model of adaptive service composition and Reinforcement learning (RL). And then we model service composition by using RL. In Sect. 3, based on graph theory, we decompose complex service composition task into multiple subtasks, and propose multi-Q learning algorithm to realize real-time subtask-assistance strategy. In Sect. 4, compare with other algorithms, the advantages of our algorithms are presented. In Sect. 5, we summarize our paper.

2. Adaptive Service Composition

2.1 Definition

Adaptive service composition could be described: Based on satisfying the function of web service composition, select the best service dynamically from some similar services to make the global QoS of composite service optimized. In fact, the process of web service composition is similar to that of the workflow management system [6]. The workflow management system can achieve specific function according to the predefined workflow model, and some tasks and transformations are included [21]. We select and combine existing services to form a specific workflow after creating an abstract description in the service composition problem. Workflow is considered as a task sequence to achieve specific goals, each atomic task represents a web service that performs the corresponding functions. As shown in Fig. 1, it represents a complete processing framework of service invocation. The input of the framework is the user’s service request, and then the request is converted to the target of service composition. Adaptive Service Composition framework builds mathematical model for service composition task by using MDP and interacts with dynamic environment in real time to update web services constantly. After the composite service is determined, the framework could invoke it to realize users’ request. The framework can adapt to environmental changes effectively which helps to adaptive service composition.

2.2 Reinforcement Learning

Reinforcement learning could solve such a problem: how to learn to choose the best action for an agent that can perceive the environment. Agent will receive reward or punishment after deciding to make an action in the environment. We define this as reward value. Agent’s task is to learn from the delayed reward, in order to choose the following actions to make the cumulative reward maximization. This paper model this problem based on MDP. Agent can sense the set of different states in the environment, S. For each state, there is a set of actions corresponding A. When agent is in a state $s_t$, it selects the appropriate action $a_t$ to execute and then get a reward value $r_t = r(s_t, a_t)$, a new state $s_{t+1} = \delta(s_t, a_t)$ is generated where $\delta$ and $r$ are part of the environment. Agent has no prior knowledge about environment, function $\delta(s_t, a_t)$ and $r(s_t, a_t)$ are only related to the current state and action. The task of agent is to learn a policy $\pi: S \rightarrow A$ which could choose an action $a_t$ to act based on current state $s_t$ and make agent get maximum cumulative reward. Let $V^\pi(s_t)$ represent cumulative reward that agent acquires by following a policy $\pi$ and $s_t$ is the initial state:

$$V^\pi(s_t) = r + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$  (1)

The reward sequence $r_{t+i}$ is reward value that agent acquires by acting $a_{t+i}$ when it is in state $s_{t+i}$, $a_{t+i}$ can be obtained according to the policy $\pi$. $\gamma (0 \leq \gamma < 1)$ is a constant which determines the discount value of a delayed reward and an immediate reward. In other words, the reward value that obtained by agent which acts in $i$th step will discount on $\gamma^i$. It means that we take immediate reward into consideration.
When \( \gamma = 0 \). Otherwise, the immediate reward and the reward of the action to be performed in the future are equal important when \( \gamma \) close to 1.

The task of reinforcement learning is to find a policy \( \pi: S \rightarrow A \), which maximizes \( V^\pi(s) \) for any \( s \in S \). The policy \( \pi \) is called optimal policy, it’s denoted as \( \pi^* \). Expressed as a formula:

\[
V^\pi(s) = \max_{\pi} V^\pi(s)
\]

or

\[
\pi^*(s) = \arg\max_{a}[r(s, a) + \gamma V^\pi(\delta(s, a))]
\]

Q-Learning [22] is an important algorithm for reinforcement learning, the evaluation function \( Q(s,a) \) is defined as: Its value is the maximum discount cumulative reward when agent starts with state \( s \) and uses \( a \) as the first action. Expressed in formula:

\[
Q(s,a) = r(s,a) + \gamma V^\pi(\delta(s,a))
\]

According to the formula Eq. (4), \( Q(s,a) \) is the maximum value when agent is in state \( s \) and select the optimal action \( a \). It can be concluded that:

\[
\pi^*(s) = \arg\max_{a} Q(s,a)
\]

According to the formula Eq. (5), in order to obtain the optimal policy \( \pi^* \), agent only need to select the action that maximizes \( Q(s,a) \) when agent is in state \( s \). Defines the formula for iteratively updating the \( Q(s,a) \) value:

\[
Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]
\]

\( r \) represents immediate reward when agent is in state \( s \) and select the action \( a \), \( 0 \leq \alpha \leq 1 \) is learning rate, it plays the role of regulatory factors. Generally speaking, the larger the value of \( \alpha \), the faster the \( Q \) function converges, and the algorithm will find the optimal policy.

2.3 Reinforcement Learning for Service Composition

Some studies have used MDP to model web service composition. The process of searching for the optimal policy can be regarded as the optimal choice of multi states and multi paths for agent. We can integrate multiple alternative workflows and web services into a single service composition by MDP. The alternative web service set \( A \) is equivalent to the actions collection in MDP.

As shown in Fig. 2, there are several workflows to choose for a finite user’s request. The user’s request will be decomposed into a variety of atomic services, and the model can complete the request by combine several specific services. \( S_0 \) is the initial state, agent can reach the final state through different service combinations. Each edge represents a progress of invoking the service with specific function to arrive at an intermediate state for agent. \( A_i \) represents a collection of candidate services when the agent is in state \( S_i \). If the agent selects a service \( (a \in A_i) \) and executes, it will arrive at another state.

In the process of searching for optimal service composition, agent can obtain reward value after invoking a selected service. We use the QoS attributes of the web service to define the reward value. Each web service may have multiple QoS attributes, such as response time, cost, availability, reliability and etc. Users may need to composite web services with less response time and higher stability in adaptive web service composition. In order to give an overall evaluation of a web service, we map user’s preferences to a utility value based on Utility Theory. The higher the utility value is, the more it can satisfy the user’s needs.

Assuming a web service with \( I \) QoS attributes, \( q_i (i \in I) \) represents the value of the \( ith \) QoS attribute and is mapped to a value between 0 and 1 using the utility function, denoted as \( U(q_i) \). \( q_i^{max} \) and \( q_i^{min} \) respectively represent maximum and minimum value of the \( ith \) QoS attribute among the web services with the same function.

\[
U(q_i) = \begin{cases} 
\frac{q_i - q_i^{min}}{q_i^{max} - q_i^{min}}, & \text{if larger } q_i \text{ is needed} \\
\frac{q_i^{max} - q_i}{q_i^{max} - q_i^{min}}, & \text{if smaller } q_i \text{ is needed}
\end{cases}
\]

In the Eq. (7), \( q_{max} - q_{min} \neq 0 \). If \( q_{max} - q_{min} = 0 \), then \( U(q_i) = 1 \). In order to obtain the overall utility function of all the QoS attributes for specific service, all the QoS attribute values are mapped into a real number by using the weighted sum of each QoS attribute utility function. A simple weighted method is applied to calculate the utility value of Web Service, the importance of each QoS attribute is represented by normalized weights and the sum of the normalized weights for all QoS attributes is equal to 1: \( \sum_{i=1}^{I} w_i = 1 \). Let \( R(s) \) denotes the return value of service \( s \).

\[
R(s) = \sum_{i=1}^{I} w_i U(q_i)
\]
3. Subtask-Assistance Strategy

3.1 Task Decomposition

Complex service composition task can be decomposed into multiple subtasks that are independent service composition tasks. In Fig. 2, if we delete a node \( s_i (0 < i < t) \) and all the edges connected to it and cannot find a path to connect \( s_0 \) with \( s_t \) then we call the node \( s_i \) articulation point. In Fig. 3, \( s_1, s_6, s_9, s_{10} \) and \( s_{12} \) are all articulation point which split graph in Fig. 2 into three subgraphs. As shown in Fig. 3, we can regard each subgraph as a subtask.

We define \( \pi(s_i, s_j) \) as the service selection policy from \( s_i \) to \( s_j \), \( s_0 \) is the initial state and \( s_t \) is the terminal state, and \( \pi^*(s_i, s_j) \) is the most optimal policy. Both \( s_{11} \) and \( s_{12} \) are articulation point. Therefore, we can conclude that:

\[
\pi(s_0, s_t) = \begin{cases} 
\pi(s_0, s_{11}) \\
\pi(s_{11}, s_{12}) \\
\pi(s_{12}, s_t)
\end{cases}
\] (9)

According to the definition of articulation point, there is no path between \( s_0 \) and \( s_t \) if we remove articulation point \( s_{11} \) and \( s_{12} \). Formula (9) is a piecewise function, besides \( \pi(s_0, s_{11}), \pi(s_{11}, s_{12}) \) and \( \pi(s_{12}, s_t) \) are independent each other. Therefore the optimal policy from \( s_0 \) to \( s_t \) is expressed as a formula:

\[
\pi^*(s_0, s_t) = \begin{cases} 
\pi^*(s_0, s_{11}) \\
\pi^*(s_{11}, s_{12}) \\
\pi^*(s_{12}, s_t)
\end{cases}
\] (10)

3.2 Multi-Q Learning for Subtask-Assistance

As shown in Fig. 2, a service composition model which contains several states from \( s_0 \) to \( s_t \). In order to find optimal composited service, we need to choose optimal path from \( s_0 \) to \( s_t \) which maximize QoS of composited service. So the problem is to find global optimal policy \( \pi^*(s_0, s_t) \). The task could be decomposed into three subtasks shown in Fig. 3. According to the Eq. (10), we can get:

\[
\pi^*(s_0, s_i) = \begin{cases} 
\pi^*(s_0, s_{11}) \\
\pi^*(s_{11}, s_{12}) \\
\pi^*(s_{12}, s_i)
\end{cases}
\]

(11)

In other words, the global optimal policy is the sum of the three subtasks. Based on the above theory, if a service composition workflow based on MDP, which contains \( n \) subtasks, then it contains \( n-1 \) articulation points, and \( s_{li} (1 \leq i < n) \) are articulation points. The optimal policy \( \pi^* \), which the agent will learn, can be defined as follows:

\[
\pi^*(s_0, s_i) = \begin{cases} 
\pi^*(s_0, s_{11}) \\
\pi^*(s_{11}, s_{12}) \\
\vdots \\
\pi^*(s_{ln-1}, s_{ln})
\end{cases}
\]

(12)

In the Eq. (12) \( n \geq 2 \). We can conclude that global optimal policy could be found only to find optimal policy for each subtask.

Subtasks could be regard as sub functions that are a part of workflow. They can be invoked in a specific order to achieve a final goal. Therefore, the loop between subtasks is not considered in current paper. We construct workflow that contains no loop for service composition in our paper. Therefore, according to the definition of articulation point, we can see that subtasks are independent of each other and the choice of the optimal policy for each subtask can be regarded as a Q-Learning process. Let \( S_k \) \( (s_k \in S_k) \) as a set of states for the \( k \)th subtask and \( A_k \) \( (a_k \in A_k) \) is a set of actions corresponding, so we can get the Q learning expression corresponding to the \( k \)th subtask.

\[
Q_k(s_k, a_k) = r_k(s_k, a_k) + \gamma \max_{a' \in A_k} Q_k(s_k', a_k')
\]

(13)

The global Q function expression of service composition task from the state \( s_0 \) to \( s_t \) is defined as Eq. (14):

\[
Q'(s, a) = Q_k(s_k, a_k) + \sum_{i=k+1}^{n} \max_{a_i} Q_i(s_{i-1}, a_i)
\]

(14)
So, the problem of convergence of a global Q function is decomposed into convergence problem of multi-Q functions, and the global optimal policy is defined as Eq. (15):

\[
\begin{align*}
\pi^*(s) &= \arg\max_a Q^*(s, a) \\
&= \arg\max_{a_k} Q_k(s_k, a_k) \\
&= \arg\max_{a_k} \max_{1 \leq k \leq n} Q_k(s_k, a_k)
\end{align*}
\]

(15)

According to the Eq. (15), when the agent is in the state \(s \in S_k\), the selection of optimal policy is only related to the current subtask \(k\). Therefore, we can consider each subtask as an independent entity. The service composition task will find the optimal policy when each of its subtask finds the optimal sub policy. It make subtask-assistance strategy possible. As shown in Fig. 4, there are \(n\) service composition tasks (T1, T2… Tn), they could contain the same subtasks.
although they are different tasks. As illustrated in this figure, the same color circles represent the same subtask. In the process of multi-Q learning, the same subtasks coming from different service composition tasks can exchange information to accelerate the process of finding the optimal policy for all involving tasks (T1, T2... Tn). The subtask-assistance strategy is the core of multi-Q learning algorithm. This process is represented as Algorithm 1.

For multiple service composition tasks that achieve different functions may contain same subtasks. These same subtasks share the same state and action spaces. Suppose that, we can get $D$ different subtasks by decomposing multiple service composition tasks totally, and the $d$th ($1 \leq d \leq D$) subtask may have $E_d$ same tasks and $S_d$ states. $A_{sd}$ is a collection of candidate services for the $s_d$th ($1 \leq s_d \leq S_d$) state. For example, in Fig. 4, the same color circles represent the same subtask. Therefore, the subtask-assistance strategy can be realized by Algorithm 1.

Based on Q-learning algorithm and subtask-assistance strategy, we propose multi-Q learning algorithm for multiple service composition task (see Algorithm 2). Figure 5 shows the procedure of multi-Q learning algorithm.

### 4. Experiments

In this simulation study, we mainly shows that effectiveness and advantages of our strategy. First, compare with Q-learning algorithm and multi-agent algorithm based on Q-learning, our algorithm can get faster learning speed. Secondly, for one specific task, we study the effect of the number of the same subtasks from different tasks on learning speed of our algorithm. Thirdly, we discuss if the number of subtasks that one task contains could affect learning speed of our algorithm. At last, we verify the advantages of our algorithm while other algorithm are not available.

In this paper, the learning speed of algorithm is measured by the number of episodes when the algorithm converges, it also shows that the speed of the task to learn the optimal policy. We set the threshold of discounted accumulative rewards to 5% when algorithm converges. In order to simulated dynamic environment, we change the QoS values of 10% services per 500 episodes. We set some parameter values: learning rate $\alpha = 0.3$, $\epsilon$-greedy strategy $\epsilon = 0.6$ and reward discount $\gamma = 0.8$. Experiment platform: CPU: Intel Xeon E5-2609 2.4GHz. RAM: 16GB and windows server 2008 operating system.

In current paper, three QoS attributes are mainly considered as reward assessment, which are Response Time, Throughput and Latency based on the QWS Dataset [23, 24]. We simulated the three QoS attributes mentioned above by reference to the QWS Dataset and created a maximum of 20000 individual web services. The distribution of QoS values is shown in Table 1. Take response time for example, it means that there are 10% of all web services that their response time is between 20ms and 200ms. In our experiments, each action is a finite web service that could return finite result by invoking it. We could get every state through observing the finite result. Therefore, we can obtain the discounted cumulative rewards by Eq. (1).

#### 4.1 Learning Speed of Multi-Q Learning

The aim of this experiment is to study the learning speed of our algorithm for large-sale service composition. We focus on one service composition task, which contains 100 states and 80 candidate services for each state. This task contains three subtasks. We suppose each subtask has three same subtasks when our algorithm is in the learning process. In order to demonstrate the advantages of our algorithm, we compare with traditional Q-learning algorithm and Multi-agent algorithm based on Q-learning. We set the number of agent for Multi-agent algorithm is three that share the same set of states and actions. We define $\pi^j_s(a) (1 \leq j \leq 3)$ as current optimal policy that each agent obtain when an episode end. Agents choose the better one from three current optimal policy for their next episode. The result of this experiment shows in Fig. 6. It is obvious that the learning speed of our algorithm is significantly faster when compared with the Q-learning algorithm. Moreover, our algorithm has a faster learning speed than Multi-agent algorithm with three agents.

![Comparison of different algorithms](image)

**Fig. 6** Comparison of different algorithms

| Attributes     | Units          | Distribution of QoS values |
|----------------|----------------|---------------------------|
| Response       | ms             | 20-200 200- 500- 1000- 1500- |
| Time           | invokes/second | 1-2 2-3 3-8 8-10 10-20 |
| Throughput     |                | 1-20 20-40 40-100 100-200 200-300 |
| Latency        | ms             | 1-20 20-40 40-100 100-200 200-300 |

#### 4.2 Learning Performance of Multi-Q Learning

In the learning process, we mainly shows the learning performance of our algorithm compared with traditional Q-learning algorithm and Multi-agent algorithm. In order to demonstrate the advantages of our algorithm, we compare with traditional Q-learning algorithm and Multi-agent algorithm based on Q-learning. We set the number of agent for Multi-agent algorithm is three that share the same set of states and actions. We define $\pi^j_s(a) (1 \leq j \leq 3)$ as current optimal policy that each agent obtain when an episode end. Agents choose the better one from three current optimal policy for their next episode. The result of this experiment shows in Fig. 6. It is obvious that the learning speed of our algorithm is significantly faster when compared with the Q-learning algorithm. Moreover, our algorithm has a faster learning speed than Multi-agent algorithm with three agents.

| Attributes     | Units          | Distribution of QoS values |
|----------------|----------------|---------------------------|
| Response       | ms             | 20-200 200- 500- 1000- 1500- |
| Time           | invokes/second | 1-2 2-3 3-8 8-10 10-20 |
| Throughput     |                | 1-20 20-40 40-100 100-200 200-300 |
| Latency        | ms             | 1-20 20-40 40-100 100-200 200-300 |

#### 4.3 Learning Speed of Multi-Q Learning

In this simulation study, we mainly shows that effectiveness and advantages of our strategy. First, compare with Q-learning algorithm and multi-agent algorithm based on Q-learning, our algorithm can get faster learning speed. Secondly, for one specific task, we study the effect of the number of the same subtasks from different tasks on learning speed of our algorithm. Thirdly, we discuss if the number of subtasks that one task contains could affect learning speed of our algorithm. At last, we verify the advantages of our algorithm while other algorithm are not available.

In this paper, the learning speed of algorithm is measured by the number of episodes when the algorithm converges, it also shows that the speed of the task to learn the optimal policy. We set the threshold of discounted accumulative rewards to 5% when algorithm converges. In order to simulated dynamic environment, we change the QoS values of 10% services per 500 episodes. We set some parameter values: learning rate $\alpha = 0.3$, $\epsilon$-greedy strategy $\epsilon = 0.6$ and reward discount $\gamma = 0.8$. Experiment platform: CPU: Intel Xeon E5-2609 2.4GHz. RAM: 16GB and windows server 2008 operating system.

In current paper, three QoS attributes are mainly considered as reward assessment, which are Response Time, Throughput and Latency based on the QWS Dataset [23, 24]. We simulated the three QoS attributes mentioned above by reference to the QWS Dataset and created a maximum of 20000 individual web services. The distribution of QoS values is shown in Table 1. Take response time for example, it means that there are 10% of all web services that their response time is between 20ms and 200ms. In our experiments, each action is a finite web service that could return finite result by invoking it. We could get every state through observing the finite result. Therefore, we can obtain the discounted cumulative rewards by Eq. (1).

#### 4.1 Learning Speed of Multi-Q Learning

The aim of this experiment is to study the learning speed of our algorithm for large-sale service composition. We focus on one service composition task, which contains 100 states and 80 candidate services for each state. This task contains three subtasks. We suppose each subtask has three same subtasks when our algorithm is in the learning process. In order to demonstrate the advantages of our algorithm, we compare with traditional Q-learning algorithm and Multi-agent algorithm based on Q-learning. We set the number of agent for Multi-agent algorithm is three that share the same set of states and actions. We define $\pi^j_s(a) (1 \leq j \leq 3)$ as current optimal policy that each agent obtain when an episode end. Agents choose the better one from three current optimal policy for their next episode. The result of this experiment shows in Fig. 6. It is obvious that the learning speed of our algorithm is significantly faster when compared with the Q-learning algorithm. Moreover, our algorithm has a faster learning speed than Multi-agent algorithm with three agents.

| Attributes     | Units          | Distribution of QoS values |
|----------------|----------------|---------------------------|
| Response       | ms             | 20-200 200- 500- 1000- 1500- |
| Time           | invokes/second | 1-2 2-3 3-8 8-10 10-20 |
| Throughput     |                | 1-20 20-40 40-100 100-200 200-300 |
| Latency        | ms             | 1-20 20-40 40-100 100-200 200-300 |

#### 4.2 Learning Performance of Multi-Q Learning

In the learning process, we mainly shows the learning performance of our algorithm compared with traditional Q-learning algorithm and Multi-agent algorithm. In order to demonstrate the advantages of our algorithm, we compare with traditional Q-learning algorithm and Multi-agent algorithm based on Q-learning. We set the number of agent for Multi-agent algorithm is three that share the same set of states and actions. We define $\pi^j_s(a) (1 \leq j \leq 3)$ as current optimal policy that each agent obtain when an episode end. Agents choose the better one from three current optimal policy for their next episode. The result of this experiment shows in Fig. 6. It is obvious that the learning speed of our algorithm is significantly faster when compared with the Q-learning algorithm. Moreover, our algorithm has a faster learning speed than Multi-agent algorithm with three agents. In order to implement subtask-assistance strategy, we redefine the global Q function expression for multiple service composition task (see Eq. (14)). Therefore, our algorithm has a higher converged value than the other methods. In our paper, we mainly focus on learning speed of algorithm.
4.2 Impact of the Number of Same Subtasks

The aim of this experiment is to study the effect of the number of the same subtasks from different tasks on learning speed of our algorithm. We focus on one service composition task, which contains 100 states and 80 candidate services for each state. We decompose this task into three subtasks and vary the number of the same subtasks for each subtask from two to six. It means that each subtask has two same subtasks when the number is two. As shown in Fig. 7, we can obviously see that our algorithm has faster learning speed with the increase of the number of the same subtasks. The more the same subtasks update corresponding Q function in the learning process, the faster learning speed the subtasks have. According to Eq. (14), it also will lead to find optimal policy of the whole task faster.

Though the number of the same subtasks can accelerate learning speed of our algorithm, is it the more the better? As shown in Fig. 8, with the number of the same subtasks from different tasks increases, our algorithm has faster convergence at the beginning, but when the number exceeds seven, the learning speed of our algorithm is little change. We can conclude that, there exists a suitable number of assistant subtasks for one specific task. It will lead to an optimal learning speed.

4.3 Impact of the Number of Subtasks

The aim of this experiment is to study the effect of the number of subtasks that a single task contains on the learning speed of our algorithm. We consider the service composition tasks which contain 100 states, and each state has 80 candidate services. These tasks contain different number of subtasks from 3 to 10, and each subtask has 5 same subtasks while learning the optimal policy. As shown in Fig. 9, we can see that the number of episodes is between 500 and 700 when our algorithm converges. It shows that the number of subtasks that a single task contains has little effect on the learning speed of our algorithm. Therefore, we can only focus on the number of the same subtasks from different tasks.

4.4 Multi-Q Learning for Subtask-Assistance

The aim of this experiment is to validate our algorithm can improve learning speed of different service composition tasks having the same subtasks each other simultaneously in real time. We conduct this experiment by using four different service composition tasks, as shown in Table 2. These four different tasks contain different numbers of states and subtasks, but having one or more the same subtasks each other. The same subtask is marked as the same subtask ID. We compare the learning speed of subtask-assistance (SA) strategy and no subtask-assistance (NSA) strategy for each
different tasks improve learning speed simultaneously. In the pattern of SA, the same subtasks from different tasks can update the corresponding Q function between each other to accelerate the learning speed, but not in the pattern of NSA.

Through our experiment, we shown the comparative data in Fig. 10. We can conclude that all involved tasks which using SA can improve their learning speed simultaneously compared with using NSA. In the process of NSA, only one subtask updates its own Q function for single task, it will take a longer time for the algorithm converges.

5. Conclusion

In this paper, we mainly investigate a subtask-assistance strategy for adaptive services composition. Considering the practical service composition environments, there could be many service composition tasks at a time. We decompose service composition task into multiple subtasks according to the graph theory. Different tasks with the same subtasks can assist each other to improve their learning speed. For this purpose, we propose a multi-Q learning algorithm to realize subtask-assistance strategy. The same subtasks from different tasks implement subtask-assistance strategy between each other, which will accelerate their learning speed. In the learning process, more same subtasks will lead to faster learning speed for all involved tasks. The experiments show that, our algorithm is better than traditional Q-learning algorithm and Multi-agent algorithm. Meanwhile, we take the relationship between different tasks into account, which is not considered in previous study. Our strategy can improve their speed of learning optimal policy simultaneously for all involved service composition tasks that have the same subtasks between each other in real-time.

Acknowledgments

This paper was supported by National Natural Science Foundation of China (Key Project, No. 61432004), National Key Research and Development Program (No. 2016YFB1001404), Beijing Natural Science Foundation (No. 4164091), China Postdoctoral Science Foundation (No. 2015M580048), and Fundamental Research Funds for the Central Universities (No. FRF-TP-15-034A1).

References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” Ad Hoc Networks, vol.10, no.7, pp.1497–1516, 2012.
[2] R. Perrey and M. Lycett, “Service-oriented architecture,” Applications and the Internet Workshops, vol.44, no.4, pp.116–119, 2012.
[3] R. Richards, “Universal Description, Discovery and Integration,” Pro Php Xml & Web Services, pp.751–780, 2006.

[4] G. Alonso, et al., Web Services - Concepts, Architectures and Applications, DBLP, 2004.

[5] B. Srivastava and J. Koehler, “Web Service Composition - Current Solutions and Open Problems,” Icaps Workshop on Planning for Web Services, pp.28–35, 2003.

[6] J. Rao and X. Su, “A survey of automated web service composition methods,” Proc. First International Workshop on Semantic Web Services & Web Process Composit, vol.3387, pp.43–54, 2004.

[7] Q. Yu and A. Bouguettaya, “Framework for Web service query algebra and optimization,” ACM Transactions on the Web, vol.2, no.1, p.6, 2008.

[8] D.-H. Shin, K.-H. Lee, and T. Suda, “Automated generation of composite web services based on functional semantics,” Web Semantics Science Services & Agents on the World Wide Web, vol.7, no.4, pp.332–343, 2009.

[9] W. Jiang, S. Hu, D. Lee, S. Gong, and Z. Liu, “Continuous Query for QoS-Aware Automatic Service Composition,” IEEE, International Conference on Web Services, vol.7, pp.50–57, 2012.

[10] Y. Yan and M. Chen, “Anytime QoS-aware service composition over the GraphPlan,” Service Oriented Computing and Applications, vol.9, no.1, pp.1–19, 2015.

[11] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya, “Adaptive Service Composition Based on Reinforcement Learning,” Service-Oriented Computing, International Conference, ICSOC 2010, San Francisco, pp.92–107, 2010.

[12] A. Gao, D. Yang, S. Tang, and M. Zhang, “Web Service Composition Using Markov Decision Processes,” Advances in Web-Age Information Management, International Conference, WAIM 2005, Hangzhou, China, pp.308–319, 2005.

[13] Y. Lei, Z. Jiantao, W. Fengqi, G. Yongqi, and Y. Bo, “Web Service Composition Based on Reinforcement Learning,” IEEE International Conference on Web Services, pp.731–734, 2015.

[14] C.J.C.H. Watkins and P. Dayan, “Q-learning,” Machine Learning, vol.8, no.3/4, pp.279–292, 1992.

[15] L. Yu, W. Zhili, M. Lingli, W. Jiang, L. Meng, and Q. Xue-song, “Adaptive Web Services Composition Using Q-Learning in Cloud,” IEEE Ninth World Congress on Services IEEE Computer Society Conference, pp.393–396, 2013.

[16] R. Wang, X. Zhou, X. Zhou, W. Liu, and W. Li, “Adaptive and Dynamic Service Composition Using Q-Learning,” IEEE International Conference on TOOLS with Artificial Intelligence, vol.1, pp.145–152, 2010.

[17] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, and A. Bouguettaya, “Integrating On-policy Reinforcement Learning with Multi-agent Techniques for Adaptive Service Composition,” Service-Oriented Computing, pp.154–168, 2014.

[18] L. Yu and J. Zhang, “Service composition based on multi-agent in the cooperative game,” Future Generation Computer Systems, vol.68, pp.128–135, 2017.

[19] T.G. Dietterich, “Hierarchical reinforcement learning with the MAXQ value function decomposition,” Artif. Intell. Res. (JAIR), vol.13, pp.227–303, 2000.

[20] R. Makar, S. Mahadevan, and M. Ghavamzadeh, “Hierarchical multi-agent reinforcement learning.” Proc. 5th Int’l Conf. on Autonomous Agents, ACM, pp.246–253, 2001.

[21] Y. Shi and X. Chen, “A Survey on QoS-aware Web Service Composition,” Third International Conference on Multimedia Information NETWORKING and Security, pp.283–287, 2011.

[22] C.J.C.H. Watkins, “Learning with Delayed Rewards,” Robotics & Autonomous Systems, vol.15, no.4, pp.233–235, 1989.

[23] E. Al-Masri and Q.H. Mahmoud, “Discovering the best web service,” International Conference on World Wide Web (WWW), pp.1257–1258, 2007.

[24] E. Al-Masri and Q.H. Mahmoud, “QoS-based Discovery and Ranking of Web Services,” IEEE 16th International Conference on Computer Communications and Networks (ICCCN), pp.529–534, 2007.

Li Quan is a PhD student in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research is in Internet of things and service computing.

Zhi-liang Wang is a professor in School of Computer and Communication Engineering, University of Science and Technology Beijing, China. He is a senior board member of China Artificial Society. He received his PhD degree in Harbin Institute of Technology, China. His current research interests include artificial psychology and the Internet of things.

Xin Liu is a postdoctoral lecturer in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. She received a PhD degree in control science and technology from University of Science and Technology Beijing. Her current research is in cognitive affective computing and intelligent information processing.