Information Sciences 519 (2020) 141–160

Contents lists available at ScienceDirect

Information Sciences

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

Integrating reinforcement learning and skyline computing for adaptive service composition

Hongbing Wang[a,*], Xingguo Hu[a], Qi Yu[b], Mingzhu Gu[a], Wei Zhao[a], Jia Yan[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 12 April 2018
Revised 25 November 2019
Accepted 18 January 2020
Available online 20 January 2020

Keywords:
Service composition
QoS
Reinforcement learning
Skyline computing
Adaptability

ABSTRACT

In service computing, combining multiple services through service composition to address complex user requirements has become a popular research topic. QoS-aware service composition aims to find the optimal composition scheme with the QoS attributes that best match user requirements. However, certain QoS attributes may continuously change in a dynamic service environment, so service composition methods need to be adaptive. Furthermore, the large number of candidate services poses a key challenge for service composition, where existing service composition approaches based on reinforcement learning (RL) suffer from low efficiency. To deal with the problems above, in this paper, a new service composition approach is proposed which combines RL with skyline computing where the latter is used for reducing the search space and computational complexity. A WSC-MDP model is proposed to solve the large-scale service composition within a dynamically changing environment. To verify the proposed method, a series of comparative experiments are conducted, and the experimental results demonstrate the effectiveness, scalability and adaptability of the proposed approach.

1. Introduction

Service composition is a widely used software engineering paradigm to build complex and value-added software [1,2]. Due to its inter-operability, reusability, deployability, service composition has become one of the key technological choices to deal with complex user requirements by combining multiple atomic services [3]. Services provided by multiple service providers usually have different Quality of Service (QoS), such as price, reliability, reputation, throughput, and response time. In service composition, corresponding QoS constraints need to be considered, leading to QoS-aware service composition, which aims to generate optimal or near-optimal composite services that meet user requirements.

In a dynamic service environment, certain QoS attributes may continuously evolve. As a result, service composition method needs to adapt to the changing environment. Moreover, things may become more complicated if a service composition involves complex service workflows (e.g., WSC-MDP in Fig. 2) and a large number of candidate services, which becomes common for enterprise applications [4]. Nowadays, there are a large number of services on the Internet. For ex-

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

https://doi.org/10.1016/j.ins.2020.01.039
0020-0255/

Table 1
Notations.

Notation	Description
QoS	Quality of Service
RL	Reinforcement Learning
MDP	Markov Decision Process
WSC	Web Service Composition
SC	Skyline Computing
BPEL	Business Process Execution Language
SOA	Service-Oriented Architecture
EDA	Event Driven Architecture
HTN	Hierarchical Task Network
SDN	Software-defined Networks

ample, *programmableweb.com*[1] has documented over 22,770 APIs by September 2019, and the number of APIs is growing at an alarming rate every year. Thus, efficiency is an important and urgent aspect that must be taken into account. To solve large-scale problems, there are some existing methods like multi-agent in [5], multi-level index technology in [6]. In [5], multiple agents work together and speed up the convergence rate of the algorithm; In [6], authors propose a multi-level index model to expedite Web service discovery and composition. In general, these two methods do not reduce the number of candidate services, so there will be some unnecessary explorations in the process of learning. In this paper, we utilize skyline computing [7,8] to address the above limitations. Because the skyline chooses high-quality services from a large candidate pool, it can significantly reduce the search space, leading to efficient computation.

To deal with a dynamic environment, we leverage the advantage of reinforcement learning (RL), which learns by trial-and-error interactions with the dynamic environment and thus has good adaptability. Thus, introducing the RL into the process of service composition can optimize the service composition solution and adapt to the dynamic environment. RL is a major type of machine learning method that has become a useful technique to solve sequential decision making problems [9]. In an RL system, a learning agent learns an optimal policy via interactions with an uncertain environment. In each step, the learning agent chooses and executes the optimal action to maximize the long-term reward, instead of being told which action to take. Afterwards, the agent receives a scalar reward and the current state transits to its successive state. Finally, the agent evaluates the effect of this state transition.

In the context of service composition, on the one hand, the environment is constantly changing and certain QoS may continuously evolve. On the other hand, there exist increasing complex composition flows and a huge number of candidate services. Hence, how to adapt to dynamic environment and how to achieve high efficiency are nontrivial. In order to cope with the above two challenges, we combine reinforcement learning and skyline computing in this paper. RL is to respond to dynamic environment and achieve good adaptability. Skyline computing is used to reduce the search space and improve efficiency. More specifically, skyline computing extracts data points which are not dominated by any other point on all QoS dimensions.

In this paper, we develop a service composition approach that combines RL with skyline computing. The main contributions are summarized as follows:

- In the process of service composition, we present a new method to reduce the search space and computational complexity by exploiting skyline computing.
- A WSC-MDP model is designed to solve the large-scale service composition problems, which can also deal with a dynamically changing environment.
- We conduct a series of experiments to demonstrate the effectiveness, scalability and adaptability of the proposed approach.

Table 1 summarizes the main notations used in the rest of the paper. The remainder of this paper is organized as follows. Section 2 gives an overview of related work. Section 3 presents the preliminaries that lay the foundation of the proposed approach. Section 4 introduces the Web Service Composition Markov Decision Process (WSC-MDP) model and service composition algorithm. Section 5 details the experimental evaluation and comparison with other related works. Finally, Section 6 draws our conclusions and identifies some future directions.

2. Related work

Adaptive service composition has received significant attention. A large number of adaptive service composition methods have been proposed in recent years.

In service computing, the dynamic and uncertain environment is a big challenge for Web service composition. Cao et al. [10] proposed a concept of context service that is able to perceive and adapt to changes in the environment. They

[1] http://www.programmableweb.com/category .

also presented a framework that can detect the changes of the environment and adjust dynamically the execution of the service by using BPEL and agent technologies. Wang et al. [11] modeled context ontologies by using Web Ontology Language (OWL), and proposed a self-adaptive and context-aware service composition system. Though this system can respond to changing contexts, contexts need to be predefined. In [12], Cheng et al. proposed to integrate SOA with EDA for service coordination, where SOA was used to resolve interoperability issues among heterogeneous services and physical entities as well as application reusability issues, and the EDA was used to support context awareness. In [13], a framework was proposed to implement a self-adaptive service composition. The proposed framework, named as AIS–CSF, could adapt to the changes of the environment. When the user's context changes, the recomposition of a composite service will be triggered.

Besides, in a dynamic Web environment, the QoS of Web services becomes unstable and the invocation of Web services can be uncertain. In [14], Yan et al. developed a service composition model based on AI planning. They used a preparing approach to tackle the changes in the QoS of Web services. However, the replacement of individual services lacks an overall consideration of the entire composition solution. In some other works, the problem of service composition is modeled using MDP. In [15], MDP is combined with HTN planning for adaptive service composition. However, HTN is not suitable for a large-scale scenario and susceptible to local-optimal solutions. Uc-Cetina et al. proposed to integrate MDP with dynamic programming to solve the service composition problem, but it also suffers from the local-optimum [16].

As a branch of machine learning, reinforcement learning has become an important research in many fields. In [17], RL was used for adaptive video transmission control system. They proposed an RL based Actor-critic model for adjusting the key parameters based on the information generated from the network and the feedback of video environment dynamically. In [18], a method based on RL was proposed for the robust control in uncertain continuous-time linear systems. In their method, the online policy iteration algorithm was used to find a feedback control law. In [19], Mahboob et al. applied RL to the SDN, and proposed an RL based Q-learning routing method for unicast routing, in which the Q-learning mechanism was used to update the routing path dynamically. Their experiments demonstrated the effectiveness of the proposed method.

Reinforcement learning has also been applied to service composition. To resolve ambiguity in service composition, Jungmann et al. applied RL and a recommendation system to expand state-space based service composition [20]. Liu et al. modeled the service selection problem as a Markov Decision Process, and then utilized RL to solve the problem [21]. In [22], to address the low efficiency of traditional reinforcement learning, a Multi-Agent Reinforcement Learning Algorithm was developed to achieve service composition. However, when facing a very large candidate service space, how to maintain high efficiency becomes highly nontrivial. In this paper, we use skyline computing to reduce the computation complexity and improve efficiency.

Based on many previous studies, skyline analysis has received wide attention in decision making applications. Stephan et al. [8] adopted the skyline operation to extend database systems. The skyline operation can weigh personal preferences and help make the final decision. In [23], Rahman et al. studied the skyline discovery on categorical attributes. They proposed two effective methods to address the skyline query problems without the precomputed indices. The experimental results demonstrated the effectiveness of their method. Liu et al. introduced the skyline query into the encrypted data [24]. They proposed a new secure skyline query protocol based on the semantically-secure encryption. To reduce the computation load further, the two optimizations, data partitioning and merging, were proposed. Skyline queries have attracted considerable attention to assist multi-criteria analysis in large-scale datasets. Lee et al. focused on multidimensional subspace skyline computation and narrowed down a full space skyline, which helps users consider the subspace skyline to reflect their interests [25]. Pei et al. developed an approach to compute skyline cubes [26]. The approach utilizes a skyline group lattice on a small subset of objects and avoids unnecessary search in all proper subspaces. Jureta et al. [27] presented a novel multicriteria-driven reinforcement learning method in order to adapt to dynamic Web services. Benouaret et al. [28] investigated the problem of skyline on uncertain QoS with possibility theory. They utilized a possibility distribution to represent each QoS attribute of a candidate service and improved the efficiency and effectiveness of the skyline based service selection algorithm. Zhao et al. [29] considered the skyline problem as a multi-objective optimization problem and solved it with an evolutionary algorithm.

In general, RL is an effective method to solve adaptive service composition. However, this method will become inefficient when facing a large number of candidate services. Skyline computing is a powerful tool to reduce exploratory space and computational complexity. In this paper, we propose a new service composition approach based on reinforcement learning and skyline computing to cope with these challenges outlined above, where the former is to achieve adaptability and the latter is to reduce the computational complexity.

3. Preliminaries

We present some preliminaries in this section, focusing on Reinforcement Learning, Skyline Computing, and Web service composition MDP. We also summarize the mathematical notations in Table 2.

3.1. Reinforcement learning

In the field of machine learning, key machine learning techniques can be divided into four categories, supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning (RL).

Table 2
Mathematical notations.

Notation	Description
S	The Finite Set of States of The World
s_0	The Initial State
s_τ	The Finite Set of Terminal State
$A(s)$	The Set of Services Can Be Executed In State $s \in S$
P	The Probability Distribution Function
R	The Reward Function
V_i'	The Standardized Value of $i-th$ QoS Attribute

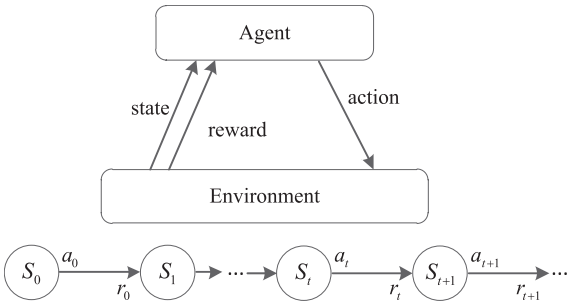


Fig. 1. The RL framework.

Benefiting from the research of reflection mechanism in bionics, RL is a machine learning technique that takes environmental feedback as input and adapts to the environment. In the process of learning, the agent adjusts its learning policy according to positive or negative reinforcement information from the executed action. The process is a heuristic process in which the agent's goal is to maximize its long-term reward value [9].

Fig. 1 shows the framework of reinforcement learning. The agent interacts with the external environment by its actions, and each action is followed by a corresponding feedback information. The feedback may be reward (positive feedback signal), or punishment (negative feedback signal). When the agent executes an action, it will update its policy based on the feedback information of the environment. At the same time, this action will lead to environmental changes from state S_t to S_{t+1}. In the process of learning, if an action gets a positive feedback (e.g., reward), the trend that the agent will execute the action will be strengthened; if an action gets a negative feedback (e.g., punishment), the agent will tend to reduce the likelihood of executing the action in the future. This is what "reinforcement" implies, i.e., strengthening its own judgment about action trend according to the feedback information. The purpose of reinforcement learning is to acquire an action policy by learning: $\pi: S \rightarrow A$. According to this policy, the action of the agent can receive the maximum reward.

The Q-learning algorithm is an effective RL method [9]. The algorithm was proposed by Watkins [30] and in this paper we inherit the original work [31] that utilizes Q-learning to solve service composition problems. Q-learning usually refers to single-step Q-learning, and its basic form is as follows:

$$Q^*(s,a) = R(s,a) + \gamma \sum_s T_{ss'}(a) \max_{a'} Q^*(s',a') \tag{1}$$

where $T_{ss'}(a)$ is the transition probability from state s to state s', γ is the discount factor which makes adjacent rewards more important than future ones and $Q^*(s, a)$ represents the maximal reward value by executing action a in the state s. The Q-learning algorithm keeps an estimation of Q^* (denoted as $\hat{Q}^*$) and adjusts $\hat{Q}^*$ based on the action performed and reward. The specific equation is as follows:

$$\hat{Q}^*(s,a) = (1-\alpha)\hat{Q}^*(s,a) + \alpha(r + \gamma \max_{a'} \hat{Q}^*(s',a')) \tag{2}$$

where r is the reward and s' is the resulting state of the action a. $\alpha \in (0, 1]$ is the learning rate and is used to control the learning speed.

When the estimated value is updated in each environment state, Q-learning doesn't point out which action the agent should perform. In fact, the agent can choose and execute any action. So the same algorithm could run again and the result may be different. To obtain the optimal Q function, the agent needs to execute all actions many times in each state. When Eq. (2) runs enough times, $\hat{Q}^*$ will converge to Q^*.

3.2. Skyline computing

A Web service and its own QoS can be seen as a multidimensional data object. Extracting a collection of data objects that are not dominated by other ones from a multidimensional data object set is the purpose of skyline computing [8,32].

Table 3
Example of three services.

	response time (ms)	throughput (invoke/second)	reliability (%)	price ($)
API-1	107	1.9	73	100
API-2	450	1.6	73	50
API-3	255	1.3	67	60

Let's consider a typical skyline query that a user wants to know about tomorrow's weather condition. However, there are a great variety of services with the same function but different QoS, including response time, price, throughput, and reliability. Assuming that there are three services with QoS given in Table 3. Ideally, we want to find the service with the best performance among all QoS dimensions. Unfortunately, such service does not exist. Therefore, skyline analysis is useful to get a balance among different QoS dimensions.

Definition 1 (Dominance relation)**.** Consider two Web services, WS_1 and WS_2 and N QoS attributes. If WS_1 is not worse than WS_2 in all the dimensions of QoS and better than WS_2 in one dimension, service WS_1 dominates service WS_2. Formally, it is defined as,

If $\forall i \in [1, N]$: $q_i(WS_2) \leq q_i(WS_1)$, and $\exists k \in [1, N]$: $q_k(WS_2) < q_k(WS_1)$, then service WS_1 dominates service WS_2, denoted as: $WS_2 \prec WS_1$, service WS_2 is dominated by service WS_1.

Skyline computing extracts Web services that are not dominated by other services, which constitute the skyline services. There are some skyline algorithms, including Divide and Conquer, Bitmap, Nearest Neighbour, Branch-and-Bound Skyline et al [7]. The Branch-and-Bound method is based on the nearest neighbor search and uses an R tree to index data points. The skyline algorithm starts from the root of the R tree. Nodes in the R tree will be discarded if they are dominated by the services in the skyline, otherwise they will be added to the skyline. Then their children nodes will be checked. In what follows, we give the Branch-and-Bound skyline computing as shown in Algorithm 1. We choose Branch-and-Bound skyline algorithm for several reasons. It is a correct, efficient and fair algorithm. The algorithm has high efficiency and scalability in both computing time and space. The R tree is convenient for users to get the skyline result, which is highly efficient and flexible. Users can define their own distance formula so that the algorithm can be personalized based on users' needs.

Algorithm 1: Branch-and-Bound skyline for service composition.

```
Initialize SP = ∅ //Skyline initialization
Insert all Web services of the root R into the heap
repeat
  for heap not empty do
    Remove top Web service ws_top
    if ws_top is dominated by some point in SP then
      Discard ws_top
    else
      //ws_top is not dominated
      if ws_top is an intermediate entry then
        for each child ws_i of ws_top do
          if ws_i is not dominated by some point in SP then
            Insert ws_i into heap
          else
            //ws_top is a data point
            Insert ws_i into SP
          end if
        end for
      end if
    end if
  end for
until heap become empty
```

4. Model and algorithm

Firstly, we give a problem description with respect to our service composition scenario. The model in [33], referred to as WSC-MDP, will be used to describe the scenario. Fig. 2 shows a transition graph to illustrate this model. The notes in Fig. 2 will be explained after Definition 3.

The main task is to select proper services for every state node (hollow circle in Fig. 2) and combine them to form an optimal service composition. In this paper, we use RL to find the optimal service composition. Our method has good adaptability because the agent learns by trial-and-error interactions with the dynamic environment. For example, considering a ticket-booking service, when many people visit the service at the same time, the service will block, which will cause a long delay. So the agent will select a service with a shorter delay in next exploration. Furthermore, to speed up the convergence of the algorithm, skyline computing is adopted to reduce the number of candidate services and explore the search space more efficiently.

In this model, the agents will choose different services according to the environment, policy and reward feedback in every state transition until reaching the terminal state. After these processes, we obtain the composite service. The key issues in the learning process include the definition of the reward feedback function for environment awareness and the algorithms in selecting the optimal service composition, which will be introduced in Section 4.2 and 4.3 respectively.

4.1. Model

In this section, we refer to the definitions in [33] and add some explanations to make them more clear.

Definition 2 (Web Service)**.** A Web service is modeled as a 6-tuple $WS =< ID, In, Out, Pr, E, QoS >$, where

- ID is the identifier of the Web service.
- In is the input of Web service.
- Out is the output of Web service.
- Pr is the precondition state that guarantees the successful invocation of the Web service.
- E is the effect to the environment after invoking and running the Web service.
- QoS is an n-tuple$< attr_1, attr_2, \ldots, attr_n >$, where each $attr_i$ denotes a QoS attribute (e.g., Throughput, Reliability, Availability, and Response Time) of a WS.

Example 1: Consider a phone register service. A user first inputs a phone number. Then the service will send the check code message to the user. At last, the user inputs the check code to finish the registration. The QoS attributes may include the price per message, the average response time, and so on.

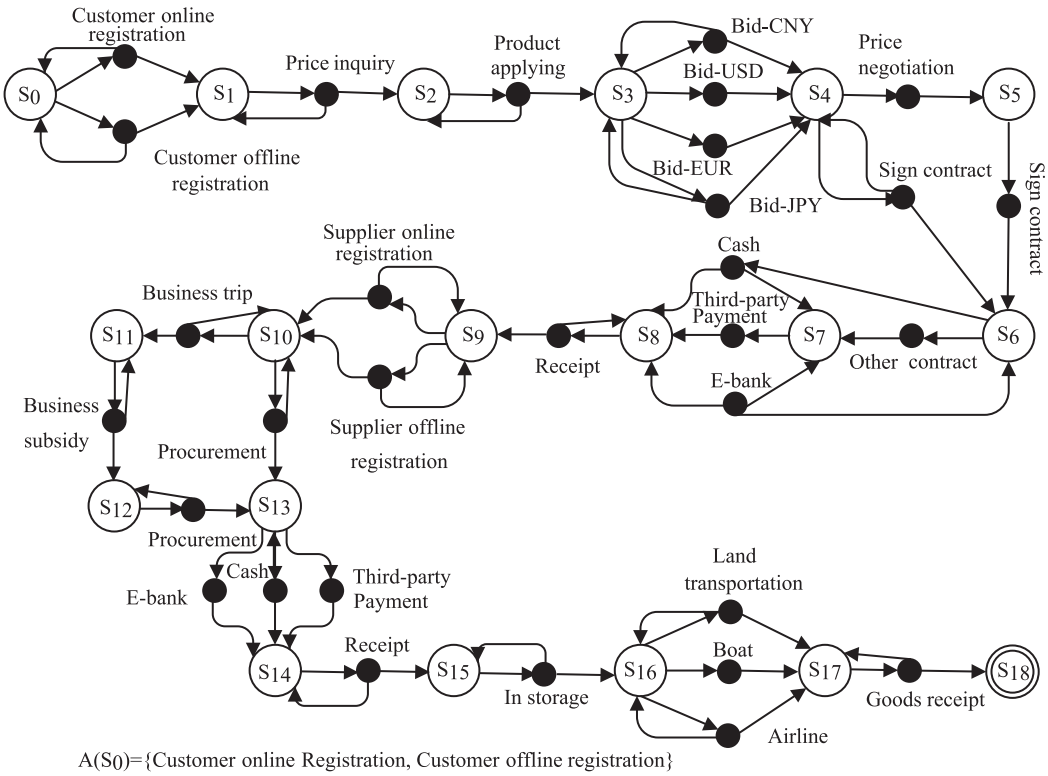
Definition 3 (WSC-MDP)**.** A Web Service Composition MDP is a 6-tuple $WSC - MDP =< S, s_0, s_\tau, A, T, R >$, where

- S is the discrete set of environment states.
- $s_0 \in S$ is the initial state. The execution of the service composition starts from s_0.
- $s_\tau \subset S$ is the set of terminal states. Upon reaching one state in this set s_τ, an execution of the service composition terminates.
- $A(s)$ is the set of services that can be executed in state $s \in S$, and A is the set of services that can be executed in all states.
- T is the probability distribution function. When a service $a \in A(s)$ has been invoked, the environment makes a transition from its current state s to a succeeding state s'. The probability for this transition is $T(s'|s, a)$.
- R is the immediate reward function. When the current state is s and a service a is selected, we get an immediate reward $r = R(s, a)$ from the environment after executing the action.

The MDP model for service composition can be visualized as a transition graph. We use a case study in Fig. 2 to describe this model [33]. This comprehensive case study consists of a Sales Management of Cargo (SMC) that aims at the development of a Service-Oriented Architecture (SOA) solution to effectively process the activities within the Sales Management of Cargo. The MDP model for Sales Management of Cargo is a generic and universal business process, which purchases raw materials from suppliers and provides products to customers. The SMC essentially consists of three big modules: procurement and supplier chain, commodity sales management, and financial management. In this scenario, SMC acquires simpler individual parts or even raw materials from its suppliers, combines them to produce the merchandises, and works with its distributors to ensure the products to reach its final customers.

In Fig. 2, the hollow circles represent state nodes. The solid circles represent the abstract services, which are defined as the set of services with the same functional attributes but different non-functional attributes. The service in an abstract service set can be replaced by another service in the same set. The new service may be better or worse than the original one in non-functional attributes. The initial state is the beginning node of the graph, which is represented with s_0. The last node with a double circle is the terminal state. A state node can be followed by a number of abstract service nodes, representing the multiple possible services that can be invoked in this state, labeled with the transition probability $T(s'|s, a)$. For every $T(s'|s, a)$, a reward r can be calculated.

The service composition model based on Markov decision process (WSC-MDP) can express the control flow of a business process [34]. The solution of service composition based on WSC-MDP is a deterministic service selection policy π^{WSC} that defines which services in A are invoked in each state to obtain the optimal composition. After determining the specific service in each state through the policy, the agent learns an optimal or near optimal service composition result. Therefore, the solution of WSC-MDP is to learn optimal or near optimal polices and determine the appropriate specific services according



A(S0)={Customer online Registration, Customer offline registration}
A(S1)={Price inquiry} A(S2)={Product applying}
A(S3)={Bid-CNY, Bid-USD, Bid-EUR, Bid-JPY} A(S4)={Price negotiation, Sign contract}
A(S5)={Sign contract} A(S6)={Other contract, Cash, Third-party Payment, E-bank}
A(S7)={Cash, Third-party Payment, E-bank} A(S8)={Receipt}
A(S9)={Supplier online Registration, Supplier offline registration}
A(S10)={Business Trip, Procurement} A(S11)={Business subsidy}
A(S12)={Procurement} A(S13)={Cash, Third-party Payment, E-bank}
A(S14)={Receipt} A(S15)={In storage}
A(S16)={Land transportation, Boat, Airline} A(S17)={Goods receipt}

Fig. 2. A MDP model for Sales Management of Cargo.

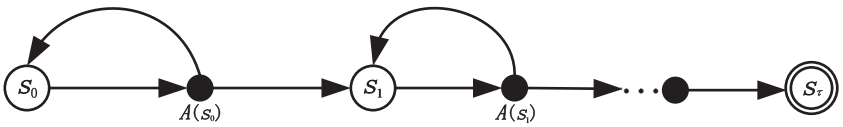


Fig. 3. One Web Service Workflow in Fig. 2.

to these polices. The solution of a WSC-MDP service composition model is a deterministic decision policy denoted by π. This deterministic decision policy maps state s to an action a: $\pi: S \rightarrow A$. We give a formal definition of policy as below.

Definition 4 (Policy)**.** A policy π is a mapping from state $s \in S$ to a concrete Web service $WS \in A(s)$, where $A(s)$ represents a set of specific services available for the state s, $\pi(s, a)$ denotes the probability of performing the action a in the state s. A deterministic policy represents the probability of executing a service is 1 in a given state.

Given a deterministic policy, the agent chooses a concrete Web service according to the policy and combines these Web services to form a service composition result. We define a Web service workflow as follows:

Definition 5 (Web Service Workflow)**.** *WF* is a Web service workflow iff there is at most one action that can be invoked at each state. More formally,

$\forall s \in WF, |A(s) \cap WF| \leq 1$.

Fig. 3 shows a web service workflow of the WSC-MDP in Fig. 2. The opposing arrows mean that a service's execution may fail so that the agent will stay in the current state. A workflow represents a whole process of the business which is similar to the traditional service composition.

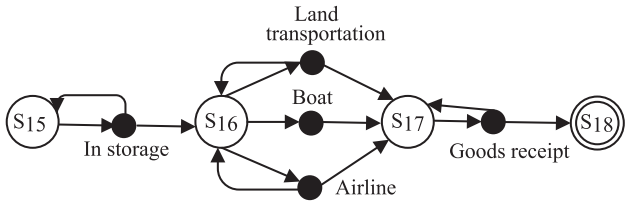


Fig. 4. A part of Fig. 2.

The Web service workflow corresponds to a specific Web service composition scheme. There are a number of possible service workflows in a WSC-MDP. According to the policy, the agent obtains a workflow, through executing services under the different states. The user will get a reward value, which is the cumulative reward of the execution of the services.

In the model, the agent aims to find the optimal policy to obtain maximum cumulative reward. In the learning process, the state transition function T and the reward function R keep updating to adapt to the changing environment of service composition (e.g., when services are deleted or added.).

4.2. Reward assessment

Reward function is an important issue in reinforcement learning. This section will explain how to calculate reward in the WSC-MDP.

It is obvious that the better service will meet user requirements better. So the user satisfaction can be used to describe the reward. QoS is commonly used for modeling the user satisfaction. The QoS attribute information can be easily gathered and processed after every invocation. For these reasons, we use QoS attributes to construct the reward function.

In this paper, we use the QoS attribute method based on service quality to design the reward function. For our WSC-MDP model, we use the RL algorithm to learn the optimal or near optimal policy set. The RL algorithm learns a policy and updates it based on the learning agent's interaction with the environment. The agent performs an action followed by a corresponding feedback signal. Based on the type of the feedback signal and the succeeding environment state, it decides the next action. In the Web service composition model, the learning agent executes a certain service and the current environment transfers to the following environment state and the agent receives the feedback signal given by environment at the same time. Environmental feedback signal is calculated according to the QoS attributes of the invoked service, and the action selection at the next step is based on the current state and the feedback signal. The process is repeated until the terminal state is reached.

A Web service has more than one QoS attributes and the range of various attributes are not the same, which make them not directly comparable. Therefore, there needs to be a process of standardization for the values of different attributes and map them into the interval [0,1] (so that they are comparable to each other). In addition, some attributes (e.g., reliability and availability) are positively correlated with the quality of service whereas others (e.g., price and response time) are negatively correlated. These need to be considered in the standardization process, which are given by formulas (3) and (4). Formula (3) is used for the positively correlated attributes and the formula (4) is for the negatively correlated attributes.

$$v_{normPos} = \begin{cases} \frac{v-\min}{\max-\min}, \max \neq \min \\ 1, \max = \min \end{cases} \tag{3}$$

$$v_{normNeg} = \begin{cases} \frac{\max-v}{\max-\min}, \max \neq \min \\ 1, \max = \min \end{cases} \tag{4}$$

4.3. Algorithm

There is an effective solution for a sequential decision making problem: reinforcement learning. In our Web service composition model, we combine Q-learning with skyline to find optimal service composition.

We have introduced skyline computing to reduce the exploration space and improve the efficiency of the composition. We apply skyline computing to each abstract service set, which is represented by a solid node in Fig. 4, to extract non-dominated Web services, as concrete services in the same set have the same functionality. We will give a service composition solution: Q-learning with skyline which will exploit the Branch-and-Bound Skyline for Service Composition (BBSFWS) given in Algorithm 1.

4.3.1. Q-Learning with skyline

Q-learning with skyline method aims to obtain the optimal policy, which meets user's requirements. An important problem is how to represent the user preference for each QoS attribute as users may care certain QoS attributes more seriously than others. Furthermore, it is usually difficult to give a clear quantitative comparison between different attributes. Therefore, for the N dimension of service quality, we construct a N-dimensional matrix, where the user fills in the relative

importance of the objectives (i.e., QoS attributes). Then, after dealing with the matrix, quantitative preference information for each objective can be obtained. We divide the importance into 10 levels, and users fill in each importance level of the dimension and the importance relative to other objectives. For example, in our Web service model, we use formula (5) to record the relative importance between each objective. Among them, there are N objective dimensions, and under the 10 levels of importance evaluation standard, a_{ij} is the relative importance of objective i to objective j. Through this matrix, we can calculate the relative importance of each QoS attribute of Web service, and get the quantitative weights by calculating. We set the relative importance of an objective to its own as 0, that is all the values of the matrix diagonal are 0. After knowing the matrix of relative importance between the QoS attributes of a service, we use the formula (6) to calculate the relative importance of each objective.

$$Matrix_{relative}\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \tag{5}$$

$$w_i = \frac{\sum_{j=1}^{N} a_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}} \tag{6}$$

In the following, we present the Q-learning with skyline method, which is built upon our service composition model (WSC-MDP). We refer to one complete process from the initial state to a terminal state by a sequence of actions in an episode.

The agent monitors the environment. As for an existing composition, we rerun the algorithm when some services' QoS deteriorates to a lower level compared to a user-defined threshold. It would appear that not only the algorithm but also the composition itself is adaptive. The computational complexity of the algorithm is growing in a polynomial order and the average number of iterations is $O(A * S^2 + S^3)$, where A and S are the numbers of actions and states, respectively. There are a large number of composition schemes being considered in the learning process. Different composition schemes correspond to different composite service results, different software architectures, and different software design patterns. In the learning process, the agent chooses different composition schemes and executes different concrete services. Until the convergence of the algorithm, we get a service composition result.

The whole process of Q-learning with skyline method is shown in Algorithm 2, where r denotes the reward, γ is the discount factor, and α denotes the learning rate controling the learning speed. When α increases, the algorithm's convergence speed is accelerated, but it is easy to fall into local optimal solutions. When α decreases, the local optimization can be reduced, but the disadvantage is that the convergence becomes slow. Therefore, when choosing α, we need consider both the convergence rate and effectiveness. For a random iterative algorithm, Eq. (7) can be used to ensure the convergence of the algorithm [30]. In Q-learning with skyline method, the reward r is computed by the weighted sum of QoS values. With the agent's learning, the accumulated Q value will converge to an optimal value in the end and the algorithm will stop.

Algorithm 2: Q-learning with skyline Method Based on WSC-MDP.

```
Initialization:
∀s, ∀a is initialized arbitrarily
Initialize SP = ∅ //Skyline initialization
Insert all Web services of the root R into the heap
repeat
  for each episode ep do
    Initialize state s
    repeat
      for each step of episode ep do
        Get skyline services SP using the skyline algorithm (Algorithm 1)
        Select a service a in SP by ϵ-greedy policy
        Execute service a, observe reward r and state s'
        Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
        s ← s'
      end for
    until state s reach one terminal state
  end for
until convergence condition is satisfied, algorithm converges
```

$$\sum_{k=1}^{\infty}\alpha_k(s,a) = \infty, \sum_{k=1}^{\infty}\alpha_k^2(s,a) < \infty. \tag{7}$$

Table 4
Some E-bank services.

Name	Thoughput (invokes/second)	Reliablity (%)
*WS*1	7	73
*WS*2	8	71
*WS*3	9	69
*WS*4	10	67
*WS*5	8	69
*WS*6	9	67

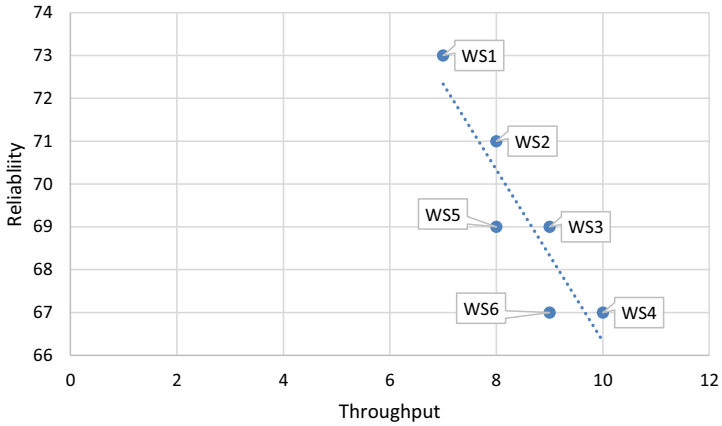


Fig. 5. The Result of Skyline Algorithm.

4.3.2. Example and discussion

In this section, we will give an example to illustrate the skyline and RL algorithms. In Fig. 2, the hollow circles represent state nodes and the solid circles represent the abstract services. At first, let's consider skyline algorithm. Supposing the current state is S_{13}, there are three abstract services, *Cash, Third-party Payment* and *E-bank*, respectively. The abstract service *E-bank* may have many concrete services or candidate services. For example, there are 6 available e-bank services in Table 4, and every service has two QoS attributes. As shown in Fig. 5, the skyline algorithm is to find the line "*WS*1 - *WS*2 - *WS*3 - *WS*4". Note that the points from the line are non-dominated by each other. Obviously, *WS*5 and *WS*6 are eliminated from candidate services. So when facing large scale scenarios, the skyline algorithm can reduce the number of candidate services effectively.

As for the RL algorithm, we use the MDP model in Fig. 2 as an example here. Algorithm 2 runs from the initial state s_0 to the terminal state s_{18}, and the agent needs to choose and execute a service from candidate services, and get a feedback (reward) in every state. When discounted cumulative reward reaches a steady state, the algorithm will stop and the agent will get the maximum reward. Let's consider the state s_{13} again. The agent selects a service (e.g., *WS*1) using the ϵ-greedy policy, and gets a reward r, where $r = 40$ (i.e., $0.5 * 7 + 0.5 * 73$). Then we update the Q matrix according to the equation in Algorithm 2. At last, we need to compute the discounted cumulative reward and determine whether the algorithm should stop. The discounted cumulative reward (DCR) is defined below:

$$DCR = \sum_i Q(s_i, a_i), i = 0, 1 \ldots t \tag{8}$$

In the Eq. (8), t stands for the terminal state index, a_i stands for the service selected by the agent in the state s_i. For example, *WS*1 is selected by the agent in the state S_{13}. When the discounted cumulative reward does not grow any more and reaches a stable state, we think the algorithm can stop.

4.3.3. Travel plan scenario

To illustrate how the algorithm works, a Travel Plan Scenario (TPS) is considered. The TPS describes a trip from the place A to place C. The plan is to arrange transportation and hotel affairs during the trip from the Internet, which is depicted by Fig. 6. The plan consists of two parts. One is choosing a convenient way of transportation in which there are three vehicles: Airplane, Train and Ship as the abstract services. Each abstract service has a set of candidate services with the same functionality and different performance. The other part is to choose the proper hotel. There are also three abstract services to be chosen with many candidate services. These two parts can be modeled as a WSC-MDP which is presented by Fig. 7.

In the service composition based on RL, the agent uses ϵ-greedy strategy to select actions in each state to obtain cumulative reward value and enters the next state through the interaction with the environment. The optimal state-action policy is obtained through multiple interactions with the environment.

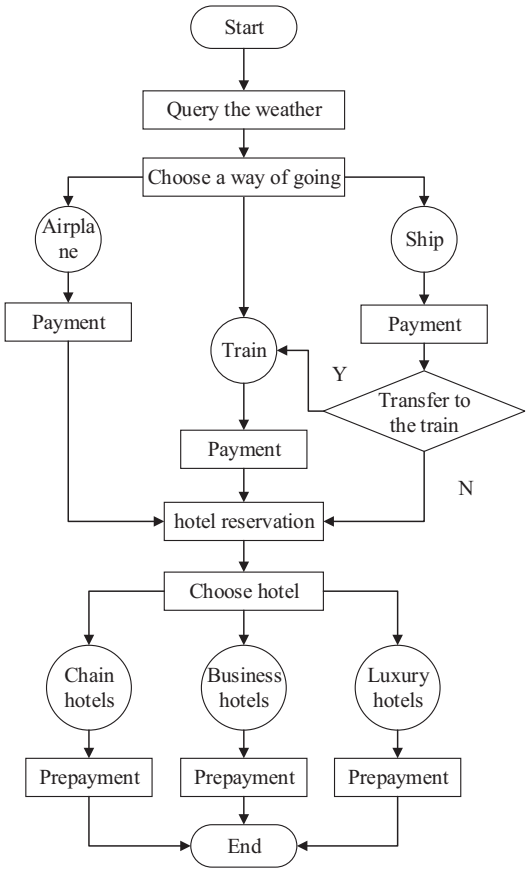
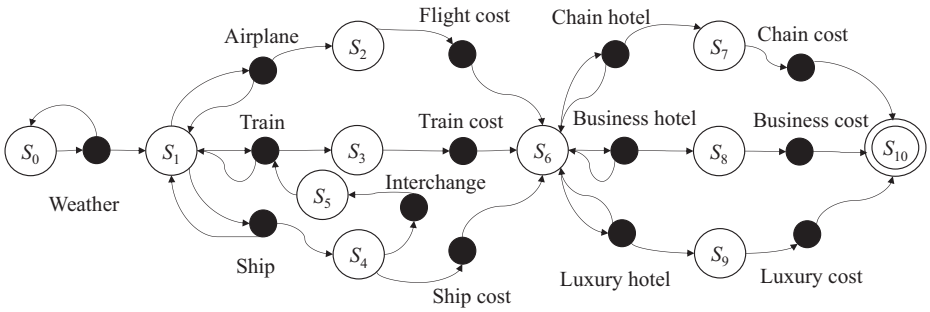


Fig. 6. Travel plan scenario.



A(S_0)={Weather} A(S_5)={Train}
A(S_1)={Airplane, Train, Ship} A(S_6)={Chain hotel, Business hotel, Luxury hotel}
A(S_2)={Flight cost} A(S_7)={Chain cost}
A(S_3)={Train cost} A(S_8)={Business cost}
A(S_4)={Ship cost, Interchange} A(S_9)={Luxury cost}

Fig. 7. The WSC-MDP model of TPS.

According to our algorithm, assuming that the current state of the agent is s_1, the agent will observe whether the skyline has been calculated in the current state and whether candidate services and their attributes fluctuate greatly. If not, an action is selected in skyline using the ϵ-greedy strategy to obtain the corresponding reward value and the agent proceeds to the next state for further exploration. Otherwise, the skyline of the abstract services will be calculated. Do the same if the candidate services have experienced significant volatility.

Supposed that the agent is located in state s_1 for the first time, the skyline will be calculated to reduce the space of action. All of the candidate services with three kinds of attributes (Response time, Throughput and Reliability) for Airplane

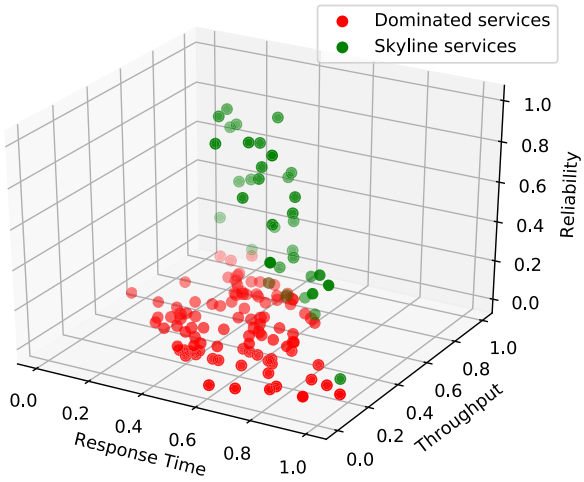


Fig. 8. The skyline of airplane services.

are as the input of skyline algorithm. Fig. 8 shows the candidate services after skyline computing, where the red points represent the dominated services, and the green points represent the skyline services. Through the skyline computing, about 60% of services are pruned and the action space is greatly reduced. Then, the agent selects a service from the skyline by the ϵ-greedy strategy and turns to the next state. The agent will continue to explore until the terminal state s_{10} is reached. The above process will be iterated until the algorithm reaches convergence. The algorithm integrates the advantages of skyline and Q-learning for large-scale adaptive service composition, in which skyline for reducing the action space and Q-learning for adapting to the dynamic changes.

5. Experiments and analysis

In this section, we conduct a series of experiments to verify the effectiveness of the model and the solution method. We mainly focus on evaluating (1) the effectiveness of skyline; (2) the effectiveness of Q-learning with skyline method; (3) the adaptability of the algorithm; (4) the scalability of the algorithm, in terms of the number of candidate services and the number of state nodes; (5) the statistical significance tests of different algorithms.

5.1. Experiment settings

At present, there is no standard platform in Web service composition, so we carry out the experiments through the simulation framework. For the experimental datasets, we use the QWS[2] dataset [35] and WS-DREAM[3] dataset [36]. These datasets are commonly used in service composition research. The data was collected after studying Web services on the Internet. In the QWS dataset, the record is more structured and easier to use, where one record corresponds to the QoS attributes of a service. The dataset consists of 2507 Web services, each of which has 11 QoS attributes, such as availability, response time, success rate, and so on. The QWS dataset is small, so we have adopted a larger dataset WS-DREAM and pruned it to obtain structured data. Table 5 shows some data samples, where one record corresponds to one Web service.

The experimental input is similar to the WSC-MDP transition graph in Fig. 2. For the WSC-MDP transition graph, unless specifically stated, the number of state nodes is set to 100, and in each state node, the number of candidate services is set to 100. After one state, there are some abstract service sets, which consist of candidate services similar in functionality but different in QoS attributes. We selected several commonly used QoS attributes to calculate the reward values, which include availability, response time, reliability and throughput. In order to facilitate the comparison with the service composition methods in the literature [15,16,31,37], the discount factor is set the same as in these approaches: $\gamma = 0.9$ and the learning rate α is set to 0.6.

The hardware/software platform configuration is as follows: Intel Core i7-2600 3.4GHz CPU, 8GB RAM, Operating system: Windows 7 x64, the programming language is Java, JDK version is 1.7.0. and the development platform is Eclipse 3.7.0.

[2] http://www.uoguelph.ca/~qmahmoud/qws/ .
[3] http://wsdream.github.io/ .

Table 5
Samples of the dataset.

response time (ms)	availibility (%)	throughput (invokes/ second)	success rate (%)	reliability (%)	consistency (%)	best practical (%)	delay (ms)	integrity (%)
302.75	89	7.1	90	73	78	80	187.75	32
482	85	16	95	73	100	84	1	2
3321.4	89	1.4	96	73	78	80	2.6	96
126.17	98	12	100	67	78	82	22.77	89
107	87	1.9	95	73	89	62	58.33	93
107.57	80	1.7	81	67	78	82	18.21	61
255	98	1.3	99	67	100	82	40.8	4
136.71	76	2.8	76	60	89	69	11.57	8
102.62	91	15.3	97	67	78	82	0.93	91
93.37	96	13.5	99	67	89	58	41.66	93

Table 6
The reduction rate of the candidate services with skyline.

	The number of candidate services				
	100	200	300	400	500
1	0.403	0.494	0.532	0.582	0.614
2	0.407	0.476	0.534	0.586	0.598
3	0.401	0.487	0.549	0.583	0.613
4	0.404	0.506	0.536	0.593	0.617
5	0.398	0.504	0.547	0.589	0.605
6	0.403	0.494	0.528	0.569	0.616
7	0.395	0.499	0.552	0.584	0.598
8	0.410	0.498	0.544	0.584	0.616
9	0.407	0.487	0.554	0.580	0.608
10	0.391	0.485	0.532	0.585	0.608
Average reduction rate	0.402	0.493	0.541	0.584	0.609

5.2. Algorithm comparison

5.2.1. The effectiveness of the skyline

In this section, to demonstrate the effectiveness of the skyline, we set the original number of state nodes to 100 and the number of candidate services of each state node is set from 100 to 500. Thus, the total number of services is from 10,000 to 50,000. And we execute our method for 10 times. The experimental results are shown in Table 6 where the reduction rate is calculated by the mean of all states.

From the Table 6, the reduction rate of the candidate services is about from 40% to 61% when the number of candidate services increases from 100 to 500, which demonstrates that the skyline in our method could effectively reduce the search space for our service composition environment. Further, it shows that with the increase of the services, the reduction rate also increases. The reason is that when the number of QoS attributes is determined, more services will be under skyline as the number of services increases.

5.2.2. Validation of effectiveness

In the experiments of this section, we set the number of state nodes to 100 and each state node has 100 available candidate services. Thus, the total number of services is $100 \times 100 = 10000$. The evaluation criterion of the effectiveness of the composition method is the discounted cumulative reward and the number of learning episodes until the convergence of the algorithm. The higher discounted cumulative reward denotes better service composition results. Less episodes used to converge denotes the better efficiency of the algorithm. In this set of experiments, we compare the Q-learning with skyline method with the single agent Q-learning method (Q-learning without Skyline) [31], dynamic programming [16], Hierarchical Task Network [15], and the Multi-agent Q-learning method [37] to verify the effectiveness of the Q-learning with skyline method.

The effectiveness is shown in Fig. 9. As can be seen, in the beginning, the Multi-agent Q-learning method is better than the other four methods. Discounted cumulative reward of the Q-learning with skyline method increases the slowest in these five methods. The reason is that our method takes a certain amount of time to calculate the skyline, so its exploration efficiency will be greatly affected compared with other algorithms. However, when our method explores enough times, the skyline of all states has been calculated, and will no longer need to be calculated in the process of subsequent exploration. After that, its exploration efficiency will be greatly improved. Due to the removal of those dominated services, the probability of choosing better services in exploration is obviously higher than that of other methods, leading to a higher cumulative reward value. As shown in Fig. 9, with the continuous learning, the Q-learning with skyline method shows its advantages. After about 1200 episodes, the Q-learning with skyline method exceeds the standard Q-learning method and dynamic pro-

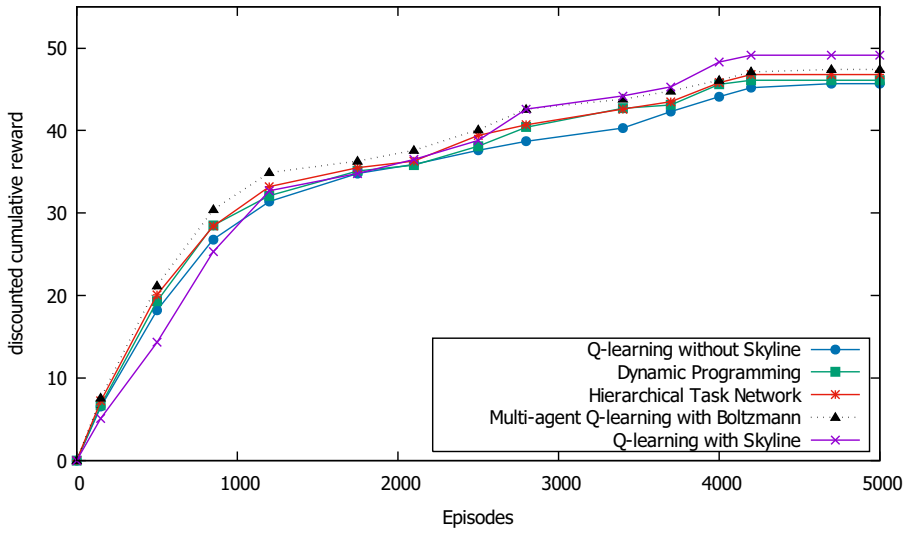


Fig. 9. Validation of effectiveness.

gramming and gets more discounted cumulative reward. At about the 2800-th episode, the Q-learning with skyline method exceeds the Hierarchical Task Network and Multi-agent Q-learning method. This is because that with the continuous exploration of the agent, the skyline in all states is calculated step by step, so that the agent can learn from the reduced candidate services, reduce unnecessary exploration, and get a higher discounted cumulative reward. Through continuous learning, the five methods have finally converged. We see that the Q-learning with skyline method converges along with the dynamic programming and Hierarchical Task Network after about 4200 episodes. Multi-agent Q-learning and the Q-learning without skyline converge after about 4700 episodes. After convergence, the discounted cumulative reward of the Q-learning with skyline method is higher than those of the other methods. The results verify the effectiveness of the Q-learning with skyline method and show the advantage in efficiency.

5.2.3. Validation of adaptability

The experiment setting in this section (the number of states and the number of candidate services) is the same as the above one. Adaptability means software evaluates and adjusts its behavior in response to the changes when the evaluation result indicates that its behavior does not accomplish what the software is intended to do, or there is a chance to reach better performance [38,39].

To simulate the dynamic environment, we periodically and randomly varied the QoS values of existing services, and also selected them randomly, based on a certain frequency to validate adaptability of the algorithm. We assumed that the system had no knowledge about the services' QoS attributes, and let it rely on the learning method to learn the optimal execution policy. When the QoS attributes change and the original optimal policy is no longer an optimal one, the algorithm will re-learn the policy and eventually converge.

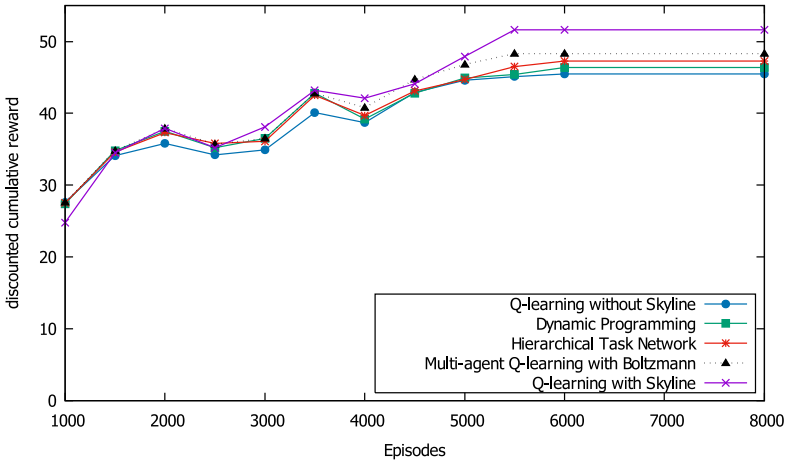
In Fig. 10(a)–(c), we randomly changed 1%, 5%, and 10% services' QoS values at a fixed time. It shows the results of the five methods when facing dynamic changes in the environment. When changing the QoS values, the current optimal policy may not be the optimal result anymore. Since it is possible to add more explorations, it leads to the decline of the algorithm. Through the re-learning process, the method will converge again. For each change, the Q-learning with skyline method need to use skyline computing to extract non-dominated services. Thus, the change has more influence on Q-learning with skyline method compared with other four methods. Along with the learning, the Q-learning with skyline method's convergence accelerates, and the discounted cumulative reward increases. As can be seen in Fig. 10 (a), (b) and (c), all five methods converge at last, and the Q-learning with skyline method has good adaptability to the environment, which can help provide more reliable and flexible service composition.

5.2.4. Validation of scalability

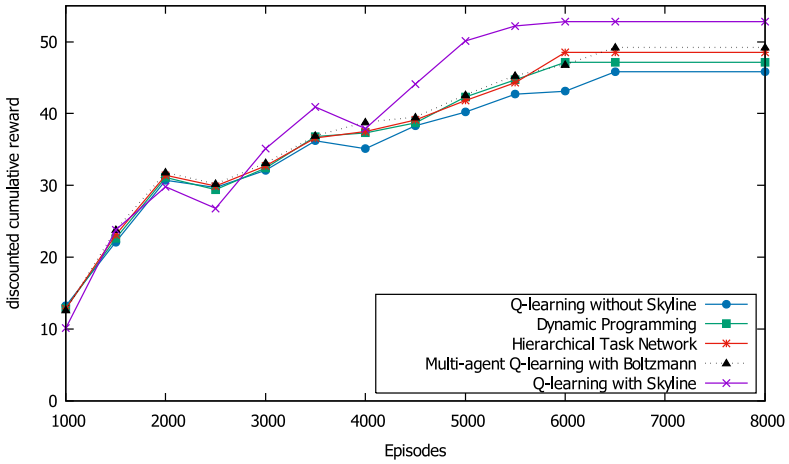
Scalability is an important indicator for the generalization of a method. Therefore, the main purpose of the following experiments is to verify the influence of the number of candidate services and states on the efficiency of the algorithm.

First, we carry out a number of experiments with respect to different candidate services. In Fig. 11, the number of state nodes of the service composition is set as $|S|=100$, and the number of candidate services of each state node is changed from 200 to 400.

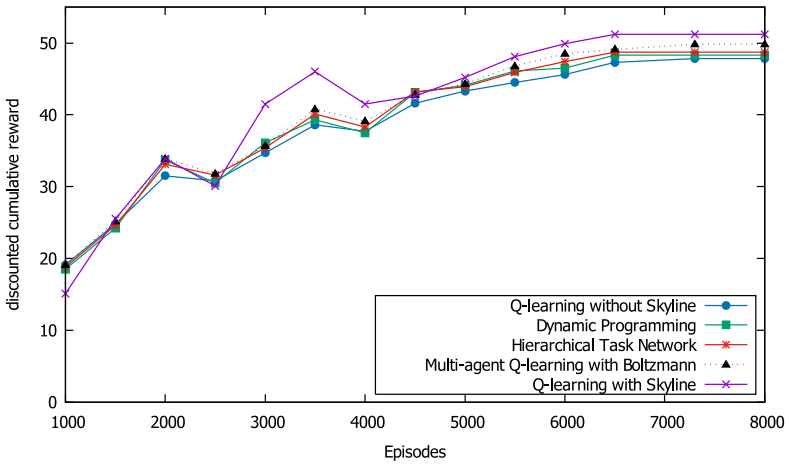
As shown in Fig. 11, the Q-learning with skyline method begins with the use of skyline computing to process the service space, resulting in low efficiency at the beginning. With the continuous improvement of the learning process, the efficiency of the Q-learning with skyline method has been improved, showing better efficiency and higher reward. When the number



(a) Changes 1% in a fixed period

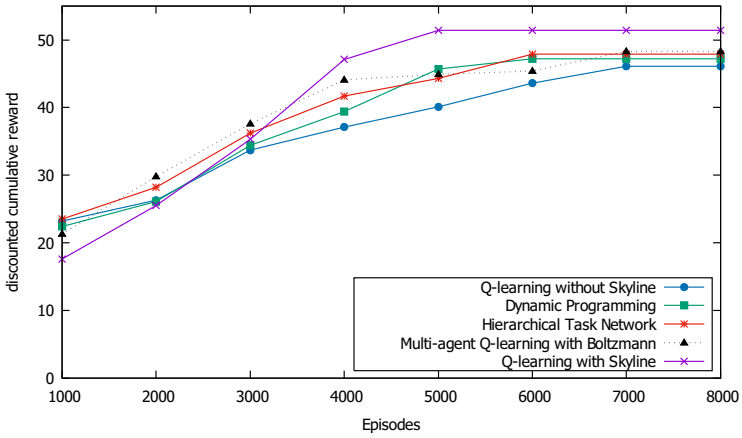


(b) Changes 5% in a fixed period

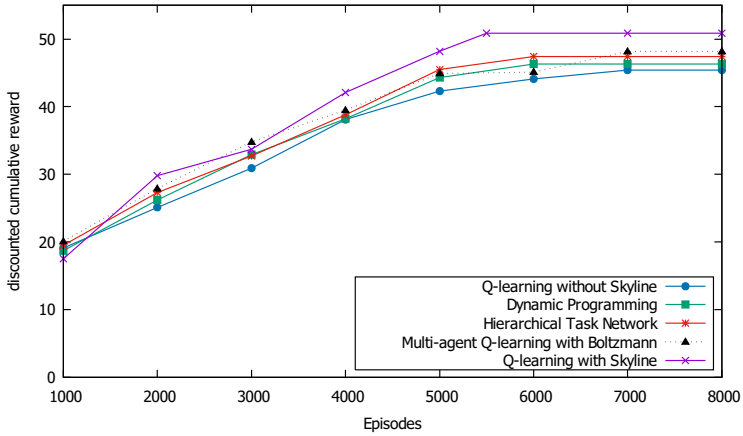


(c) Changes 10% in a fixed period

Fig. 10. Validation of adaptability.



(a) 200 services for each state node



(b) 400 services for each state node

Fig. 11. Influence ofdifferent numbers of services.

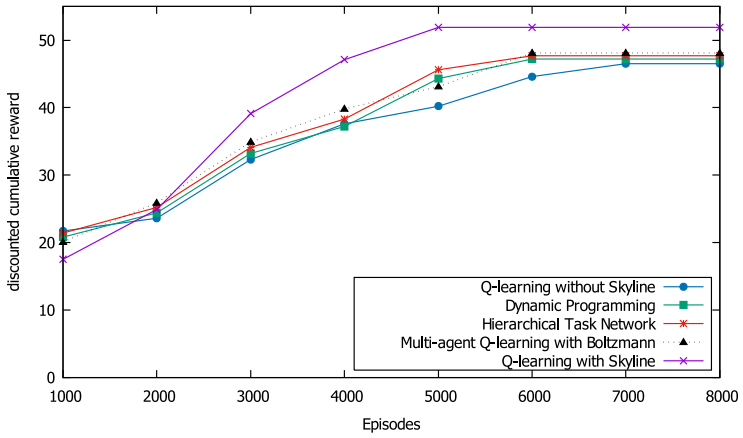
of services in each state node changes from 200 to 400, the cumulative reward is not greatly improved, or even declines. The reason is that new services are not necessarily superior to the existing ones. On the other hand, with the increase in the number of candidate services, convergence time increases.

Second, we carry out experiments with respect to different numbers of states, and explore the performance of the Q-learning with skyline algorithm to deal with the different numbers of states. The states are set to 200 and 400, and the number of candidate services in each state node is set to 100. As can be seen from Fig. 12, with the increase of the number of states, the convergence time of the algorithm becomes longer, because the increase of the state leads to the increase of possible compositions. Through above two sets of experiments and comparisons with the existing methods, it is verified that the Q-learning with skyline method has good scalability with respect to the number of candidate services and the number of states.

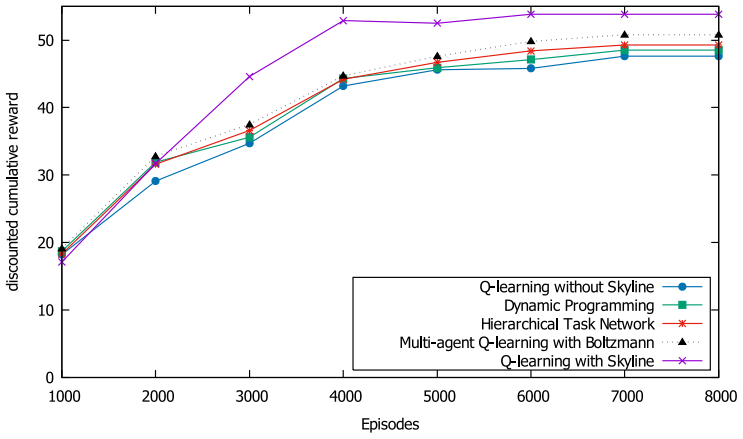
5.3. Statistical significance test

The experimental results show that Q-learning with skyline obtains the best reward. However, since RL is a stochastic algorithm, enough statistical evidence needs to be collected to make a convincing result. Therefore, we conduct statistical significance tests to confirm this.

In this section, we compare our method with multi-agent Q-learning, Hierarchical Task Network, Dynamic Programming and Q-learning without skyline by Wilcoxon test and sign test. Each algorithm is executed 15 times to obtain their final discounted cumulative rewards. Then, the differences between reward values will be computed and ranked by absolute value (if some differences are equal, their ranked value should be the average of their rank). The sign of difference indicates which algorithm obtains a better discounted cumulative reward. Tables 7–10 show the results of these experiments.



(a) 200 states



(b) 400 states

Fig. 12. Influence of different numbers of states.

Tables 7–9 compare multi-agent Q-learning, Hierarchical Task Network and Dynamic Programming with Q-learning with skyline, respectively. The rank sum of positive difference equals 7, 7 and 6. The counts of positive difference equal 3, 2 and 1. For Wilcoxon test, with 15 times contrast and significance level of 0.01, the critical value is 16. Our rank sum value 7 or 6 is smaller than the critical value, so we can conclude that the differences between these three pairs of algorithms are unlikely to occur by chance. In other words, these results of Q-learning with skyline are better than that of other algorithms. As for sign test, a similar conclusion can be reached. The critical value of sign test is 3 (with significance level of 0.05) and our counts of positive difference equal 3, 2 and 1. But with significance level of 0.01, the critical value of sign test is 2. So, with sign test, we can not be sure that Q-learning with skyline is better than multi-agent Q-learning, but it is better than Hierarchical Task Network and Dynamic Programming. The result of sign test means that sometimes multi-agent Q-learning will obtain a better reward than Q-learning with skyline. But from the result of Wilcoxon test, we can see that the reward obtained by Q-learning with skyline is significantly better than multi-agent Q-learning. According to the above observation, it is reasonable to come to the conclusion that Q-learning with skyline algorithm is better than other three algorithms.

Table 10 compares Q-learning without skyline with Q-learning with skyline. We can see that all difference values are negative. This result means Q-learning with skyline is significantly better than Q-learning without skyline.

5.4. Remarks

We conduct a series of experiments to evaluate the service composition model and algorithms proposed in this paper from four aspects: effectiveness, adaptability, scalability, and statistical significance. For the Q-learning with skyline method, we compare it with the Q-learning without skyline, Dynamic Programming, Hierarchical Task Network and the multi-agent Q-learning method. The evaluation criterion of the effectiveness of the composition method is discounted cumulative reward and the number of learning episodes until convergence. Five methods are executed in the same simulation environment re-

Table 7
Statistical Significance Test - with Multi-agent Q-learning.

	Multi-agent Q-learning	Q-learning with skyline	difference	rank	sign
1	46.01	47.54	−1.53	7	−
2	46.34	50.34	−4.00	14	−
3	47.92	50.88	−2.96	10	−
4	48.64	47.88	0.76	4	+
5	47.51	48.23	−0.72	3	−
6	48.39	49.19	−0.80	5	−
7	47.88	47.75	0.13	2	+
8	46.64	50.97	−4.33	15	−
9	48.04	50.99	−2.95	9	−
10	47.2	50.87	−3.67	12.5	−
11	48.22	49.9	−1.68	8	−
12	47.42	50.92	−3.50	11	−
13	47.47	47.44	0.03	1	+
14	46.52	50.19	−3.67	12.5	−
15	46.91	48.19	−1.28	6	−

Table 8
Statistical Significance Test - with Hierarchical Task Network.

	Hierarchical Task Network	Q-learning with skyline	difference	rank	sign
1	47.86	47.54	0.32	3	+
2	46.93	50.34	−3.41	12	−
3	48.21	50.88	−2.67	9	−
4	47.22	47.88	−0.66	5	−
5	48.14	48.23	−0.09	2	−
6	47.25	49.19	−1.94	7	−
7	48.22	47.75	0.47	4	+
8	47.04	50.97	−3.93	15	−
9	47.29	50.99	−3.70	14	−
10	47.63	50.87	−3.24	10	−
11	47.79	49.9	−2.11	8	−
12	47.38	50.92	−3.54	13	−
13	47.37	47.44	−0.07	1	−
14	46.9	50.19	−3.29	11	−
15	47.51	48.19	−0.68	6	−

Table 9
Statistical Significance Test - with Dynamic Programming.

	Dynamic Programming	Q-learning with skyline	difference	rank	sign
1	47.92	47.54	0.38	1	+
2	47.31	50.34	−3.03	10	−
3	47.26	50.88	−3.62	11	−
4	47.44	47.88	−0.44	2	−
5	47.5	48.23	−0.73	3.5	−
6	47.19	49.19	−2.00	7	−
7	46.72	47.75	−1.03	5	−
8	46.96	50.97	−4.01	13	−
9	47.13	50.99	−3.86	12	−
10	45.74	50.87	−5.13	15	−
11	47.61	49.9	−2.29	8	−
12	45.95	50.92	−4.97	14	−
13	46.71	47.44	−0.73	3.5	−
14	47.19	50.19	−3.00	9	−
15	46.21	48.19	−1.98	6	−

spectively to verify the effectiveness. For the adaptability, we modify the services' QoS values randomly to simulate the dynamic environment. The results of five methods in the dynamic environment can verify the adaptability. For the scalability, we carry out a number of experiments with more candidate services and more states. Finally, we compare different algorithms with Wilcoxon test and sign test to make sure that our study does not occur by chance. The comparative study shows that the Q-learning with skyline method is superior to the common Q-learning method, Dynamic Programming, Hierarchical Task Network and multi-agent Q-learning method.

Table 10
Statistical Significance Test - with Q-learning without skyline.

	Q-learning without skyline	Q-learning with skyline	difference	rank	sign
1	47.39	47.54	−0.15	1	–
2	45.95	50.34	−4.39	11	–
3	47.62	50.88	−3.26	8	–
4	45.45	47.88	−2.43	7	–
5	47.98	48.23	−0.25	2	–
6	47.47	49.19	−1.72	3	–
7	45.38	47.75	−2.37	6	–
8	47.29	50.97	−3.68	9	–
9	46.47	50.99	−4.52	12.5	–
10	46.99	50.87	−3.88	10	–
11	45.38	49.9	−4.52	12.5	–
12	45.63	50.92	−5.29	15	–
13	45.15	47.44	−2.29	5	–
14	45.11	50.19	−5.08	14	–
15	46.23	48.19	−1.96	4	–

6. Conclusion and future work

In this section, we first summarize the paper and describe some ongoing works that try to improve the proposed framework. Then we identify some future directions.

6.1. Conclusions

As an important way to realize the reuse of services, service composition has become a new paradigm for building complex software. The increase of homogeneous services promotes QoS-based service selection. The continuous evolution of the internal and external environment requires that the service composition paradigm can adapt to these changes. The complex user requirements combined with the increasing candidate services demand a service composition to have good scalability. We combine skyline computing with Q-learning RL for service composition, where skyline computing is applied to improve the efficiency of the composition computation. The main results of this paper are summarized as follows:

- We use the WSC-MDP model for the large-scale service composition scenarios, which can cope with the dynamic changing environment.
- Skyline computing is used to improve the efficiency of the composition scheme combined with the reinforcement learning method, which can represent user preferences.
- We conduct a series of experiments to show that the proposed methods are effective, adaptive and scalable.

6.2. Future work

We identify a number of interesting and important directions that we plan to further explore in future research.

- First, our approach requires that the environment can be fully observed, which may not be true in real scenarios. We aim to use more general decision models, such as Partially Observed Markov Decision Process (POMDP) to model service composition.
- Second, we will use the multi-agent technology to integrate with skyline computing to further improve the efficiency.
- Third, function approximation is an effective method in the face of continuous and large-scale state space. It is promising to consider combining function approximation with skyline for solving large-scale and adaptive service composition.
- Fourth, since Web services are based on the Internet, which may lead to QoS fluctuations, we will consider some QoS prediction mechanisms to improve the reliability and adaptability of service composition.

Declaration of Competing Interest

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

CRediT authorship contribution statement

Hongbing Wang: Conceptualization, Formal analysis, Writing - original draft, Writing - review & editing. **Xingguo Hu:** Investigation, Writing - original draft. **Qi Yu:** Writing - review & editing. **Mingzhu Gu:** Investigation. **Wei Zhao:** Investigation. **Jia Yan:** Investigation. **Tianjing Hong:** Investigation.

Acknowledgments

This work was partially supported by National Key Research and Development Program (no. 2018YFB1003800) and NSFC Projects (nos. 61672152, 61232007, 61532013), and Collaborative Innovation Centers of Novel Software Technology and Industrialization and Wireless Communications Technology. Qi Yu was supported in part by an NSF IIS award IIS-1814450 and an 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] L. Baresi, S. Guinea, Self-supervising bpel processes, IEEE Trans. Software Eng. 37 (2) (2011) 247–263.
[2] C. Ye, H.-A. Jacobsen, Whitening soa testing via event exposure, IEEE Trans. Software Eng. 39 (10) (2013) 1444–1465.
[3] S. Dustdar, W. Schreiner, A survey on web services composition, Int. J. Web Grid Serv. 1 (1) (2005) 1–30.
[4] H. Wang, C. Yu, L. Wang, Q. Yu, Effective bigdata-space service selection over trust and heterogeneous qos preferences, IEEE Trans. Serv. Comput. (4) (2018) 644–657.
[5] H. Wang, X. Wang, A novel approach to large-scale services composition, in: Asia-Pacific Web Conference, Springer, 2013, pp. 220–227.
[6] Y. Wu, C. Yan, Z. Ding, G. Liu, P. Wang, C. Jiang, M. Zhou, A multilevel index model to expedite web service discovery and composition in large-scale service repositories, IEEE Trans. Serv. Comput. 9 (3) (2016) 330–342.
[7] D. Papadias, Y. Tao, G. Fu, B. Seeger, An optimal and progressive algorithm for skyline queries, in: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM, 2003, pp. 467–478.
[8] S. Borzsony, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings of the 17th international conference on data engineering, IEEE, 2001, pp. 421–430.
[9] R.S. Sutton, A.G. Barto, et al., Introduction to reinforcement learning, 2, MIT press Cambridge, 1998.
[10] Z. Cao, X. Zhang, W. Zhang, X. Xie, J. Shi, H. Xu, A context-aware adaptive web service composition framework, in: 2015 IEEE International Conference on Computational Intelligence & Communication Technology, IEEE, 2015, pp. 62–66.
[11] B. Wang, X. Tang, Designing a self-adaptive and context-aware service composition system, in: 2014 IEEE Computers, Communications and IT Applications Conference, IEEE, 2014, pp. 155–160.
[12] B. Cheng, M. Wang, S. Zhao, Z. Zhai, D. Zhu, J. Chen, Situation-aware dynamic service coordination in an iot environment, IEEE/ACM Trans. Netw. (TON) 25 (4) (2017) 2082–2095.
[13] E. Khanfir, R.B. Djmeaa, I. Amous, Self-adaptive goal-driven web service composition based on context and qos, in: 2017 IEEE 14th International Conference on e-Business Engineering (ICEBE), IEEE, 2017, pp. 201–207.
[14] 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.
[15] K. Chen, J. Xu, S. Reiff-Marganiec, Markov-htn planning approach to enhance flexibility of automatic web service composition, in: 2009 IEEE International Conference on Web Services, IEEE, 2009, pp. 9–16.
[16] V. Uc-Cetina, F. Moo-Mena, R. Hernandez-Ucan, Composition of web services using markov decision processes and dynamic programming, Scientif. World J. 2015 (2015).
[17] B. Cheng, J. Yang, S. Wang, J. Chen, Adaptive video transmission control system based on reinforcement learning approach over heterogeneous networks, IEEE Trans. Autom. Sci. Eng. 12 (3) (2015) 1104–1113.
[18] D. Xu, Q. Wang, Y. Li, Robust control of uncertain linear systems based on reinforcement learning principles, IEEE Access 7 (2019) 16431–16443.
[19] T. Mahboob, Y.R. Jung, M.Y. Chung, Optimized routing in software defined networks–a reinforcement learning approach, in: International Conference on Ubiquitous Information Management and Communication, Springer, 2019, pp. 267–278.
[20] A. Jungmann, F. Mohr, B. Kleinjohann, Applying reinforcement learning for resolving ambiguity in service composition, in: 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, IEEE, 2014, pp. 105–112.
[21] Q. Liu, Y. Sun, S. Zhang, A scalable web service composition based on a strategy reused reinforcement learning approach, in: 2011 Eighth Web Information Systems and Applications Conference, IEEE, 2011, pp. 58–62.
[22] 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. (TAAS) 12 (2) (2017) 8:1–8:42.
[23] M.F. Rahman, A. Asudeh, N. Koudas, G. Das, Efficient computation of subspace skyline over categorical domains, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, ACM, 2017, pp. 407–416.
[24] J. Liu, J. Yang, L. Xiong, J. Pei, Secure and efficient skyline queries on encrypted data, IEEE Trans. Knowl. Data Eng. 31 (7) (2018) 1397–1411.
[25] J. Lee, S.-W. Hwang, Toward efficient multidimensional subspace skyline computation, VLDB J. Int. J. Very Large Data Bases 23 (1) (2014) 129–145.
[26] J. Pei, A.W.-C. Fu, X. Lin, H. Wang, Computing compressed multidimensional skyline cubes efficiently, in: 2007 IEEE 23rd International Conference on Data Engineering, IEEE, 2007, pp. 96–105.
[27] I.J. Jureta, S. Faulkner, Y. Achbany, M. Saerens, Dynamic web service composition within a service-oriented architecture, in: IEEE International Conference on Web Services (ICWS 2007), IEEE, 2007, pp. 304–311.
[28] K. Benouaret, D. Benslimane, A. Hadjali, Selecting skyline web services from uncertain qos, in: 2012 IEEE Ninth International Conference on Services Computing, IEEE, 2012, pp. 523–530.
[29] X. Zhao, L.W. Shen, X. Peng, W. Zhao, Finding preferred skyline solutions for sla-constrained service composition, in: 2013 IEEE 20th International Conference on Web Services, IEEE, 2013, pp. 195–202.
[30] C.J. Watkins, P. Dayan, Q-Learning, Mach. Learn. 8 (3–4) (1992) 279–292.
[31] 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.
[32] Z. Huang, C.S. Jensen, H. Lu, B.C. Ooi, Skyline queries against mobile lightweight devices in manets, in: 22nd International Conference on Data Engineering (ICDE'06), IEEE, Washington, DC, USA, 2006.
[33] H. Wang, X. Wang, X. Hu, X. Zhang, M. Gu, A multi-agent reinforcement learning approach to dynamic service composition, Inf. Sci. (Ny) 363 (2016) 96–119.
[34] P. Doshi, R. Goodwin, R. Akkiraju, K. Verma, Dynamic workflow composition using markov decision processes, in: Proceedings of the IEEE International Conference on Web Services(ICWS), IEEE, 2004, pp. 576–582.
[35] E. Al-Masri, Q.H. Mahmoud, Discovering the best web service, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 1257–1258.
[36] Z. Zheng, Y. Zhang, M.R. Lyu, Investigating qos of real-world web services, IEEE Trans. Serv. Comput. 1 (7) (2014) 32–39.
[37] H. Wang, Q. Wu, X. Chen, Q. Yu, Z. Zheng, A. Bouguettaya, Adaptive and dynamic service composition via multi-agent reinforcement learning, in: 2014 IEEE International Conference on Web Services, IEEE, 2014, pp. 447–454.
[38] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, ACM Trans. Auton. Adapt. Syst. (TAAS) 4 (2) (2009) 14:1–14:42.
[39] T. Vogel, H. Giese, Model-driven engineering of self-adaptive software with eurema, ACM Trans. Auton. Adapt. Syst. (TAAS) 8 (4) (2014) 18:1–18:33.