Research Article

PPDRL: A Pretraining-and-Policy-Based Deep Reinforcement Learning Approach for QoS-Aware Service Composition

Kan Yi, Jin Yang, Shuangling Wang, Zhengtong Zhang, and Xiao Ren

1Science and Technology on Information Systems Engineering Laboratory, Xi’an, China
2School of Computer Science and Technology, XiDian University, Xi’an, China

Correspondence should be addressed to Jin Yang; yangjin@stu.xidian.edu.cn

Received 8 November 2021; Revised 23 March 2022; Accepted 24 March 2022; Published 22 April 2022

Abstract

Service composition is a mainstream paradigm for rapidly constructing large-scale distributed applications. QoS-aware service composition, i.e., selection of the optimal execution plan that maximizes the composition’s end-to-end QoS properties, is an active area of research and development endeavors in service composition. In this paper, we propose PPDRL, a pretraining-and-policy-based deep reinforcement learning approach, to solve the QoS-aware service composition problem. Its significant feature is to incorporate a maximum likelihood estimate and a policy scoring mechanism into a deep reinforcement learning framework. As a result, our approach can balance the exploitation and exploration efforts adaptively and can search for the solution space in a robust and efficient manner. We have executed our approach to solve 6 randomly generated QoS-aware service composition problems with different sizes and structures based on QWS dataset including 2,507 real Web services classified into 233 categories. The results indicate that our approach can find near-optimal solutions within moderate numbers of iteration and has performance superiority in comparison with five state-of-the-art algorithms.

1. Introduction

Modern enterprises require an efficient and flexible scheme for pooling globally available services together to quickly adapt to various customer needs and dynamic market conditions. It has been proved that service composition becomes a convincing computing paradigm for rapidly developing large-scale distributed applications by these services within and across organizational boundaries.

Over the past decade, services composition has become a prevalent area of academic efforts, with a large amount of research work produced [1–7]. Among these works, QoS-aware service composition, i.e., selection of the optimal execution plan that maximizes the composition’s end-to-end QoS properties, is one of the most active areas of research and development endeavors.

QoS-aware service composition is usually modeled as a combinatorial optimization problem in many previous studies [8]. Existing service composition methods include classic algorithm, heuristic algorithms, and learning-based algorithm, as discussed in Section 2. Among them, learning-based algorithm has received much recent attention. The key idea of learning-based algorithm is to construct a QoS-value prediction model using training samples of different solutions and then explore better solutions using some searching algorithms. Among these algorithms, the deep reinforcement learning (DRL) based algorithms have received the most attention, and this is because DRL-based algorithms can solve the large-scale service composition problem adaptively and can adapt to the change of environment automatically [3]. Although previous studies on DRL-based algorithms show promising results, one critical drawback is that it requires a lot of time to train a useful model. Apparently, the assumption of the abundance of time used in many DRL-based methods is invalid considering the constrained time for optimization in the real-world problems. Given the limited optimization time in practice, only a limited number of iteration can be obtained. It typically results in a less accurate model that jeopardizes the exploration for better solutions.
In this paper, we propose PPDRL, a pretraining-and-policy-based deep reinforcement learning approach, to solve the QoS-aware service composition problem. Its key idea is to incorporate a maximum likelihood estimate and a policy scoring mechanism into a deep reinforcement learning framework. As a result, PPDRL can solve the large-scale service composition problems adaptively and can adapt to the change of environment automatically.

In summary, our work makes the following contributions:

We introduce and formulate the QoS-aware service composition (QSC) problem and model it as a combinatorial optimization problem.

We propose PPDRL, a novel approach to address QSC problem, from an unconventional direction. PPDRL can recommend promising solution by combining the maximum likelihood estimate and the policy scoring mechanism into a deep reinforcement learning framework.

We investigate the effectiveness of the proposed algorithm by solving 6 composite service instances with different sizes and structures. The experiment results have also shown the performance superiority of the proposed algorithm in comparison with five state-of-the-art algorithms.

The rest of this paper is organized as follows. Section reviews some related work. Section defines quality criteria for atomic services and the composite service model, and then gives the problem formulation. Section describes our PPDRL for QoS-aware service composition problem in detail. Section introduces our experiment setups and reports the performance evaluation and comparison of different algorithms. The paper is finally concluded in Section.

2. Related Work

QoS-aware service composition has received much attention from both industry and academia. We can classify the previous studies into three categories: classic algorithms, heuristic algorithms, and learning-based algorithms, where the last one is most recent and relevant to our work.

2.1. Classic Algorithms. Many classic algorithms have been proposed to solve the QoS-aware service composition problem, such as integer programming [9], backtracking, and branch-and-bound. Wang et al. [10] proposed a branch-and-bound algorithm based on multi-level graph that finds a feasible path and maximizes the utility of the path while reducing the search space. Fan et al. [11] presented a chained dynamic programming and hybrid pruning algorithm to transform the composition task into an equivalent task with reduced computational complexity. White et al. [12] proposed a collaborative filtering algorithm based on matrix factorization that allows programs to automatically adapt to QoS changes in their component services. Wakrime et al. [13] proposed a formal method based on satisfiability to model and verify Web service composition. Chattopadhyay et al. [4] presented an abstraction refinement-based approach to reduce the search space. Yan et al. [14] proposed a method to combine the system search algorithm with the planning algorithm. Mirandola et al. [15] proposed a set of software metrics that quantify the adaptability of service-oriented applications.

Classic algorithms use different deterministic models to solve the service composition problem. These algorithms have lower time complexity but may contain strict preconditions that may not hold in application scenarios, which severely impairs its performance.

2.2. Heuristic Algorithms. Jatho et al. [1] proposed a MapReduce-based evolutionary algorithm with guided mutations applied in large service environments. Bhushan et al. [16] proposed a hybrid particle swarm optimization technique that combines particle swarms and fruit flies for search and optimization. Rodriguez-Mier et al. [17] presented a hybrid local global search method to extract the optimal QoS value with the least number of services. Siriweera et al. [18] proposed a customizable transaction and QoS-aware service selection approach. Boussalia et al. [19] proposed an approach based on the use of a new Extended Bat Inspired Algorithm. Hammas et al. [20] proposed an architecture that supports dynamic combination and global QoS optimization. Wang et al. [21] conducted a configurability study on the artificial bee colony algorithm (ABC) and implemented a prototype system for configuring ABC parameters and optimization strategies. Da Silva et al. [2] proposed a method that generates a composition scheme based on information stored in a graph database and optimized it with genetic algorithm. Wu et al. [22] studied the transaction attributes of services and optimized the problem by ant colony algorithm. Klein et al. [23] proposed a heuristic approach based on Hill-Climbing that reduces the time complexity by narrowing the search space. Canfora et al. [24] proposed a classical genetic algorithm to provide a more stable algorithm output and get better solutions. As moveable users are very common in current service composition, frequent service re-configurations are carried out after the initial provision to maintain the QoS values, as is studied in [25, 26].

Heuristic algorithms are usually based on evolutionary algorithms. These algorithms are often bound to find a local optimal solutions. However, they have high time complexity and are usually designed only for offline service composition tasks.

2.3. Learning-Based Algorithms. In recent years, a large number of learning algorithms have been applied to solve service composition problems. Wang et al. [3] proposed a deep Q-learning (DQN) algorithm based on LSTM and fully connected layer. Labbaci et al. [27] proposed a deep learning approach for long-term QoS. Kazem et al. [28] used Bayesian network to predict new values of certain QoS attributes. Wang et al. [29] proposed a Q-learning algorithm that uses a Gaussian process to predict Q-valued functions. Wang et al. [30] proposed a multiagent algorithm based on Sarsa to
achieve the maximum possible benefit. Wang et al. [31] proposed an automatic layered reinforcement learning method that replaces manual generation of task graph by systematically integrating automatic task decomposition. Jungmann et al. [32] proposed a recommendation mechanism to expand state-space-based service composition. Elsayed et al. [33] proposed a new method combining Genetic Algorithm (GA) and Q-learning. Zhao et al. [34] proposed a machine learning method using learning-to-rank algorithm to automatically learn user preferences. Shehu et al. [35] used a learning automata-based non-negative matrix factorization algorithm (LANMF) to predict network delay and optimized QoS-aware service composition problems through four evolutionary algorithms. Wang et al. [36] proposed a Q-learning-based algorithm based on Markov decision process. Li et al. [37, 38] model the problem as a Markov decision process (MDP) and propose a hierarchical deep reinforcement learning (DRL) model based on graph neural network (GNN) in [37]. Simulations experiments demonstrate their performance in virtual network function service chaining.

Learning-based algorithms use neural network to learn the optimal strategy. Although these algorithms require a large amount of time to train the models, they may potentially find a better solution due to its model complexity.

3. Problem Statement

This paper focuses on the QSC problem. It aims at finding the best set (in terms of QoS) of atomic services to execute the abstract tasks defined in a composite service. In this section, we first present the quality criteria in the context of atomic services and provide a brief definition for each criterion. After that, we construct the composite service model to facilitate a mathematical formulation of QSC problem.

3.1. Quality Criteria for Atomic Services. We consider six quality criteria for atomic services [5] in this paper, as shown in Table 1:

Mathematically, given a composite service cs containing n atomic services S = {s1, s2, . . . , sn}. Let cs have m service classes C = {C1, C2, . . . , Cm}, and each service class C1 contains k1 different service candidates: C1 = {c1, c2, . . . , cm}. The QSC problem can be formulated as follows:

\[
\max_{\mathbf{x}} (\text{QoS}(\mathbf{cs}, \mathbf{x})),
\]

where (1) states that the goal of QSC problem is to maximize the QoS value for a given composite service cs. x = [x1, x2, . . . , xn] denotes the service candidate selecting result for each atomic service in cs. Note that there is one and only one service candidate can be selected for each atomic service in cs. The constraints of the problem [39] are as follows:

\[
(\alpha_p (\mathbf{cs}, \mathbf{x}), \alpha_d (\mathbf{cs}, \mathbf{x}), \alpha_a (\mathbf{cs}, \mathbf{x}), \alpha_q (\mathbf{cs}, \mathbf{x}), \alpha_r (\mathbf{cs}, \mathbf{x}))
\]

\[
\leq (Q'_p, Q'_d, Q'_a, Q'_q, Q'_r).
\]

(2)

\[
\sum_{j=1}^{k} x_{i,j} i = 1, 2, . . . , n.
\]

(3)

\[
x_{i,j} \in \{0, 1\} i = 1, 2, . . . , n; j = 1, 2, . . . , \max_{j=1}^{n} (|c(s_j)|),
\]

(4)

where (6) state that each QoS attribute must meet the predefined QoS constraints. (3) and (5) guarantee that there is one and only one service candidate selected for each atomic service in cs.

The goal is calculated by the weighted sum of the six QoS criterias:

\[
\text{QoS}(\mathbf{cs}, \mathbf{x}) = \alpha_1 \cdot \frac{\alpha_d (\mathbf{cs}, \mathbf{x}) - \alpha_{\min} (\mathbf{cs})}{\alpha_{\max} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}
\]

\[
\quad + \alpha_2 \cdot \frac{\alpha_a (\mathbf{cs}, \mathbf{x}) - \alpha_{\min} (\mathbf{cs})}{\alpha_{\max} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}
\]

\[
\quad + \alpha_3 \cdot \frac{\alpha_q (\mathbf{cs}, \mathbf{x}) - \alpha_{\min} (\mathbf{cs})}{\alpha_{\max} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}
\]

\[
\quad + \alpha_4 \cdot \frac{\alpha_r (\mathbf{cs}, \mathbf{x}) - \alpha_{\min} (\mathbf{cs})}{\alpha_{\max} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}
\]

\[
\quad + \alpha_5 \cdot \frac{\alpha_{\min} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}{\alpha_{\max} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}
\]

\[
\quad + \alpha_6 \cdot \frac{\alpha_{\min} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}{\alpha_{\max} (\mathbf{cs}) - \alpha_{\min} (\mathbf{cs})}
\]

The QSC problem has been well studied and often modeled as a combinatorial optimization problem and is considered to be NP complete. The NP completeness proof
4. Pretraining-and-policy-Based Deep Reinforcement Learning for QSC Problem

Our proposed pretraining-and-policy-based deep reinforcement learning (PPDRL) is a hybrid framework for MOPs that combines the pretraining-based strategy and the policy-based deep reinforcement learning method. The key idea behind PPDRL is to incorporate a maximum likelihood estimation (MLE) method and a policy scoring mechanism into a deep reinforcement learning framework.

The overall framework of our proposed approach is shown in Figure 1. PPDRL consists of three important components: initialization, pretraining, and RL training. In the initialization module, we use random sampling to generate 50,000 sets of service composition results from a well-known data set and pick the best 64 service composition results as the initial samples. After that, the pretraining module trains the actor through MLE to learn the distribution characteristics of good solutions. The actor network is composed of an embedding layer and an rnn layer, to encode the candidate atomic service and output a probability distribution over candidates. After that, the RL-training module is called to further train the actor through gradient descent in order to find a combination that can obtain the better QoS values, the detailed process is states in Algorithm 1. During this optimization process, we constantly update the optimal QoS values by continually updating the sample set, and repeating the pretraining and RL-training process until the convergence condition is met.

4.1. Maximum Likelihood Estimate. We use MLE here as a statistical method to find the parameters of the correlation probability density function of a sample set. Specifically, the mathematical principle of MLE is to maximize the probability density function [40], which is defined as follows:

$$\text{lik}(\theta) = f_D(x_1, x_2, \ldots, x_n|\theta). \quad (7)$$

As shown in (7), given a probability distribution $D$, a distribution parameter $\theta$, and $n$ values $x_1, x_2, \ldots, x_n$ sampled from distribution $D$, find the $\theta$ that maximizes the probability density function.

4.2. Scores-Based Sampling Strategy. For a service composition problem, PPDRL uses the neural network to score the candidate atomic services for each atomic service. Then it samples the candidate service set for each service multiple times based on the scores to prepare for the mini-batch gradient descent. This approach balances exploration and utilization and improves the efficiency of the algorithm.

4.3. Gradient Descent. Gradient descent is the most commonly used method to optimize neural networks, and its calculation process is to solve the minimum value along the direction of gradient descent. In this paper, we use mini-batch gradient descent to optimize the algorithm to avoid falling into local optimum. Mini-batch gradient descent each time select a part of the sample to train the network, this method overcomes the instability of stochastic gradient descent and the inefficiency of batch gradient descent.

4.4. PPDRL Algorithm. Using the optimization process shown in Figure 1, we now discuss the PPDRL algorithm in Algorithm 1. We first define the state, action, and reward of the algorithm. The state is composed of the DAG of the given composite service $cs$, current service class $C_i$, and its corresponding service candidates $C_i = \{c_1, c_2, \ldots, c_k\}$. The action is to choose one atomic service from the candidates $a_t = 1, 2, \ldots, k$, and the reward is the QoS goal estimated by fill the unassigned service by the median QoS value. The agent chooses an atomic service for a current service class in the topological order and receive a feedback. This procedure will proceed until all the service class of the given composite service is assigned a corresponding service.

In the PPDRL algorithm, we use MLE to pretrain the neural network in order to accelerate convergence and use policy-based deep reinforcement learning to find better QoS values. Our objective function is the expected value of QoS in the initial state. So the gradient is the derivative of the objective function to the network parameters, $\theta$, which is defined as follows:

$$\nabla_{\theta} J(\theta) = E_{\pi_0}[\text{QoS}(\pi_0|\theta_0)]\nabla_{\theta}\log p(\pi_0|\theta_0). \quad (8)$$

### Table 1: Six quality criteria for an atomic service

<table>
<thead>
<tr>
<th>Quality criterion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution price: $q_p(s)$</td>
<td>$q_p(s)$ represents the fee that a service requester has to pay for invoking $s$.</td>
</tr>
<tr>
<td>Execution duration: $q_d(s)$</td>
<td>$q_d(s)$ measures the expected delay in seconds between the moment when a request is sent and the moment when the results are received.</td>
</tr>
<tr>
<td>Availability: $q_s(s)$</td>
<td>$q_s(s) = I_s/I_t$, where $I_s$ is the number of successful invocations of $s$ and $I_t$ is the number of total invocations of $s$, during a constant time interval set by an administrator of the service provider.</td>
</tr>
<tr>
<td>Throughput: $q_t(s)$</td>
<td>$q_t(s) = I_t/\theta$, where $I_t$ is the number of total invocations of $s$ and $\theta$ is the given time interval.</td>
</tr>
<tr>
<td>Successful execution rate: $q_e(s)$</td>
<td>$q_e(s) = N_e(s)/I_t$, where $N_e(s)$ is the number of times that the service $s$ has been successfully completed within the maximum expected time frame and $I_t$ is the total number of invocations.</td>
</tr>
<tr>
<td>Reliability: $q_r(s)$</td>
<td>$q_r(s) = 1 - N_e(s)/N_i(s)$, where $N_i(s)$ is the total number of error response messages.</td>
</tr>
</tbody>
</table>
After determining the gradient, we use Adam optimizer [41] to optimize the network parameters and get the optimized QoS value. Finally, we will choose better results to update the sample set and proceed to the next round of training until the algorithm meets the convergence conditions.

5. Experiments

In this section, we examine the performance of PPDRL-SC by empirically comparing it with five state-of-the-art algorithms. The source code and the data can be found in https://github.com/xdbdilab/ppdrl.

5.1. Experimental Settings

Running Environment. All experiments run on a server equipped with two 8-core Intel XeonE5-2650v2 2.6 GHz processors, 256GiB RAM, 1.5 TB disk, and running CentOS 7.5.

Workloads. We use a well-known synthetic workflow generator [42] to randomly generate composite service instances with six different sizes (number of atomic services): 10, 30, 50, 70, 90, and 100. For each instance, every algorithm is executed in a large number of iterations and then is forced to stop when converges. For each run in our experiments, each workload is executed for five times and calculate the average of these five runs.

Benchmark. We conduct extensive experiments on the QWS data set [43] that includes 2,507 real Web services. To generate the candidate service set for each atomic service, we apply a simple text clustering method to all these service names in the QWS data set and produce 233 service classes [39]. The number of candidate services in each service class ranges from 2 to 128.

Performance metrics. We consider two performance metrics, namely QoS values of composite services (short for QoS) and running times (short for RT) for different algorithms, in our experiments for performance evaluation.

5.2. Baseline Algorithms and Hyperparameters. To evaluate the performance of PPDRL-SC, we compare it with five state-of-the-art algorithms, namely multiconstrained optimal path problem for multistage graph (MCOP_M) [10], genetic algorithm (GA) [24], pointer network (PTR) [44], Q-learning (QLR) [36], and deep Q-learning (DQN) [3]. We provide a brief description for each algorithm as follows.

**MCOP_M** is a two-stage algorithm based on branch-and-bound strategy. It attempts to find a feasible service composition solution subject to multiple constraints simultaneously and maximize the utility of the solution.

**GA** is a genetic algorithm-based approach to solve the QSC problem. It uses the integer array encoding strategy, the standard two-points crossover operator, the random mutation operator, and the elitist selection method.

**PTR** is a framework to tackle combinatorial optimization problems using neural networks and reinforcement learning.

**QLR** applies reinforcement learning to obtain the optimal solution at runtime by directly studying the results of execution.

**DQN** decomposes an MOP into a number of single-objective optimization subproblems. At each iteration, a
predictive distribution model is built for each individual objective in the MOP by using fuzzy clustering and Gaussian stochastic process modeling.

Note that MCOP_M belongs to the classic algorithm category; GA is an mainstream evolutionary algorithm; and PTR, QLR, and DQN are three state-of-the-art reinforcement learning-based algorithm for the QSC problem. For a fair comparison, we adopt the recommended hyperparameter settings for the algorithms that have achieved the best performance reported in the previous literature, and the details are given in Table 2.

5.3. Experimental Results. The statistical results of the QoS values and the corresponding variances achieved by the six algorithms on six different test cases are summarized in Table 3, where the best results are highlighted. We can see from Table 3 that PPDRL outperforms all the other five algorithms on six test cases. For some test cases (i.e., Nodes 30, 70, 90, 100), PPDRL can give significantly better solutions; for the other test cases (i.e., Nodes 10 and 50), PPDRL and GA can give similar solutions. These results indicate that PPDRL can give better mean solution quality in general. Furthermore, PPDRL gives smaller standard deviation of function values than other algorithms in most cases, and hence it has a more stable solution quality.

The detailed execution traces for the second run of six test cases are recorded and shown in Figure 2. In particular, the QoS values are listed with different running times. We can tell from Figure 2 that PPDRL obtains good initial QoS values compared to other algorithms. In addition, PPDRL requires relatively smaller number of iterations to converge

![Image](image.png)

**Algorithm 1: PPDRL(cs,C,T,B).**

**Table 2: Hyperparameters for each algorithm.**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCOP_M</td>
<td>N/A</td>
</tr>
<tr>
<td>GA</td>
<td>population_size: 64, cross_rate: 0.5, mutation_rate: 0.2</td>
</tr>
<tr>
<td>PTR</td>
<td>embedding_hidden_dim: 128, LSTM_hidden_dim: 128, lr1_decay_step: 1000, lr1_decay_rate: 0.96, a: 0.9</td>
</tr>
<tr>
<td>QLR</td>
<td>learning_rate: 0.2, reward_decay: 0.9, e_greedy: 0.6</td>
</tr>
<tr>
<td>DQN</td>
<td>hidden_dim: 30, lr_start: 0.0002, lr_decay_step: 5000, lr_decay_rate: 0.96, max_epsilon: 0.9, min_epsilon: 0.1, epsilon_increment:0.01, memory_capacity: 300</td>
</tr>
<tr>
<td>PPDRL</td>
<td>preipotrain_lr: 0.001, rl_lr: 0.0001, hidden_dim: 128</td>
</tr>
</tbody>
</table>

**Table 3: The QoS values (mean and variance) on six different test cases.**

<table>
<thead>
<tr>
<th>Problem #</th>
<th>PPDRL Mean</th>
<th>Var</th>
<th>MCOP_M Mean</th>
<th>Var</th>
<th>GA Mean</th>
<th>Var</th>
<th>PTR Mean</th>
<th>Var</th>
<th>QLR Mean</th>
<th>Var</th>
<th>DQN Mean</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes 10</td>
<td>0.32</td>
<td>0.00</td>
<td>0.24</td>
<td>2.22e-3</td>
<td>0.32</td>
<td>0.0000</td>
<td>0.29</td>
<td>6.83e-5</td>
<td>0.29</td>
<td>1.19e-4</td>
<td>0.25</td>
<td>2.45e-4</td>
</tr>
<tr>
<td>Nodes 30</td>
<td>0.31</td>
<td>0.00</td>
<td>0.13</td>
<td>4.23e-3</td>
<td>0.28</td>
<td>4.38e-3</td>
<td>0.13</td>
<td>1.55e-5</td>
<td>0.12</td>
<td>5.54e-5</td>
<td>0.05</td>
<td>1.44e-4</td>
</tr>
<tr>
<td>Nodes 50</td>
<td>0.20</td>
<td>0.0000</td>
<td>0.08</td>
<td>1.37e-3</td>
<td>0.20</td>
<td>1.07e-7</td>
<td>0.13</td>
<td>7.12e-6</td>
<td>0.13</td>
<td>4.83e-6</td>
<td>0.06</td>
<td>3.32e-4</td>
</tr>
<tr>
<td>Nodes 70</td>
<td>0.15</td>
<td>6.64e-12</td>
<td>0.06</td>
<td>2.89e-4</td>
<td>0.14</td>
<td>1.02e-6</td>
<td>0.07</td>
<td>1.49e-5</td>
<td>0.07</td>
<td>2.22e-5</td>
<td>0.02</td>
<td>1.72e-4</td>
</tr>
<tr>
<td>Nodes 90</td>
<td>0.14</td>
<td>0.0000</td>
<td>0.05</td>
<td>4.30e-5</td>
<td>0.13</td>
<td>4.56e-6</td>
<td>0.04</td>
<td>6.50e-5</td>
<td>0.03</td>
<td>7.36e-5</td>
<td>-0.02</td>
<td>3.48e-5</td>
</tr>
<tr>
<td>Nodes 100</td>
<td>0.20</td>
<td>7.54e-7</td>
<td>0.09</td>
<td>4.30e-4</td>
<td>0.18</td>
<td>2.31e-6</td>
<td>0.09</td>
<td>2.57e-5</td>
<td>0.08</td>
<td>2.31e-4</td>
<td>0.05</td>
<td>5.18e-4</td>
</tr>
</tbody>
</table>

Security and Communication Networks
than most of other algorithms, and it hence has a faster convergence speed.

Finally, we plot the QoS values of MCOP_M, GA, PTR, QLR, DQN, and PPDRL on six different test cases in Figure 3. In each figure, x-axis lists the test cases and y-axis represents the QoS values. We observe that PPDRL achieves an average of 129.00% improvement over MCOP_M, 6.11% improvement over GA, 114.86% improvement over PTR, 142.25% improvement over QLR, and 421.89% improvement over DQN. We can conclude from Figure 3 that PPDRL achieves stable and significant improvements compared with the other five algorithms. During online training, PPDRL can converge to a better solution faster than other algorithms, which also indicates that PPDRL can adapt to a new network topology and service requests quickly. Another interesting observation is that the GA achieves surprisingly good results in our experiments. This is consistent with the findings of Jula in [8].

5.4. Threats to Validity. Internal validity: to increase the internal validity, we performed controlled experiments by...
executing each test case five times and calculate the average of these five runs. Such method can avoid misleading effects of specifically selected test cases and ensures the stability of the results. In addition, we set the values of hyperparameters for each compared algorithm as suggested by their authors (see Table 2). Finally, we have tried multiple values of hyperparameters for PPDRL in our experiments and observed that the good values of these hyperparameters that can lead to better results are almost the same from test case to test case.

**External validity:** we increase the external validity by choosing six different composite service with different sizes. Furthermore, we are aware that because our PPDRL is a general black-box approach and is independent to the composite service structures and service classes, the results of our evaluations are transferable to other different service composition scenarios.

### 6. Conclusion and Future Work

In this paper, we propose PPDRL—a novel service composition solution based on deep reinforcement learning with pretraining-and-policy strategy for adaptive and large-scale service composition problems. The key idea of PPDRL is to incorporate a maximum likelihood estimate and a policy scoring mechanism into a deep reinforcement learning framework. More specifically, PPDRL uses the MLE to pretrain the neural network in order to accelerate convergence. After that, it uses the neural network to score the candidate atomic services for each atomic service and samples the candidate service set for each service multiple times based on the scores to prepare for the mini-batch gradient descent. As a result, such approach balances exploration and utilization and improves the efficiency of the algorithm. In depth experiments on PPDRL demonstrate its superior performance to five state-of-the-art optimization algorithms on six different testing scenarios.

Our further work includes refining our PPDRL approach by supporting the automatic selection of the appropriate hyperparameters given a specific composite service and the corresponding service classes. We would compare our algorithms with latest benchmarks and with different network topology and service request. More performance metrics would be evaluated in our future work. In addition, we hope to abstract the proposed algorithm and release it as an automatic tool for different service composition scenarios.

**Data Availability**

No data were used to support this study

**Conflicts of Interest**

The authors declare that they have no conflicts of interest.

**References**


