Intelligent Automation & Soft Computing

Tech Science Press

DOI: 10.32604/iasc.2023.039380

Article

Hyper-Heuristic Task Scheduling Algorithm Based on Reinforcement Learning in Cloud Computing

Lei Yin[1], Chang Sun[2], Ming Gao[3], Yadong Fang[4], Ming Li[1] and Fengyu Zhou[1,*]

[1]School of Control Science and Engineering, Shandong University, Jinan, 250061, Shandong, China
[2]School of Software, Shandong University, Jinan, 250101, Shandong, China
[3]Academy of Intelligent Innovation, Shandong University, Shunhua Road, Jinan, 250101, Shandong, China
[4]Inspur Cloud Information Technology Co., Ltd., Inspur Group, Jinan, 250101, Shandong, China

*Corresponding Author: Fengyu Zhou. Email: zhoufengyu@sdu.edu.cn

Received: 25 January 2023; Accepted: 13 April 2023; Published: 23 June 2023

Abstract: The solution strategy of the heuristic algorithm is pre-set and has good performance in the conventional cloud resource scheduling process. However, for complex and dynamic cloud service scheduling tasks, due to the difference in service attributes, the solution efficiency of a single strategy is low for such problems. In this paper, we presents a hyper-heuristic algorithm based on reinforcement learning (HHRL) to optimize the completion time of the task sequence. Firstly, In the reward table setting stage of HHRL, we introduce population diversity and integrate maximum time to comprehensively determine the task scheduling and the selection of low-level heuristic strategies. Secondly, a task computational complexity estimation method integrated with linear regression is proposed to influence task scheduling priorities. Besides, we propose a high-quality candidate solution migration method to ensure the continuity and diversity of the solving process. Compared with HHSA, ACO, GA, F-PSO, etc, HHRL can quickly obtain task complexity, select appropriate heuristic strategies for task scheduling, search for the the best makspan and have stronger disturbance detection ability for population diversity.

Keywords: Task scheduling; cloud computing; hyper-heuristic algorithm; makespan optimization

1 Introduction

The development of cloud computing provides stable and efficient solutions for the information industry, which attracts many researchers to study application problems. The resources in a cloud environment are shared among users through virtualization technology [1,2], which is one of the essential functions of cloud computing as well as realizes the dynamic sharing of physical resources and the execution of multiple programs in different virtual machines (VMs) on a physical server [3]. Cloud providers can achieve personalized customization of different users with lower energy consumption and maintenance costs via virtualization technology.

Based on virtualization technology, cloud resource providers will create multiple Virtual Machines(VM) [4] based on physical resources to process tasks submitted by users. Tasks are assigned to a designated virtual machine by a scheduling algorithm. Reasonable allocation of tasks on each VM not only makes full use of cloud computing resources but also improves task completion efficiency. Besides, it has a significant impact on the stability of cloud services, user satisfaction, and providers' operating costs. Therefore, efficient scheduling algorithms are indispensable for cloud computing centers.

Currently, traditional algorithms or heuristic algorithms are generally used to achieve task scheduling in the cloud environment. However, the above algorithms still have two problems.

(a) Traditional scheduling algorithms have the advantages of stable performance and easy implementation. However, large-scale task scheduling problems in complex cloud environments are often an NP-hard problem. Common non-heuristic task scheduling algorithms such as First Come First Serve (FCFS), max-min [5], linear-programming [6], Round Robin, Weighted Round Robin, RASA, Segmented Min-Min algorithm are static algorithms following a predefned approach of scheduling the tasks in the computing environment. These algorithms have poor parameter dynamic performance and high computational resource consumption. Amir presented the cTMvSDN to improves resource management based on combination of Markov-Process and Time Division Multiple Access (TDMA) protocol, which does not apply to resource scheduling tasks with different task attributes [7].The heuristic intelligence-based task scheduling is adaptive, intelligent, collective, random, decentralized, self-collective, stochastic and is based on biologically inspired mechanisms than the other conventional mechanisms. Researchers also apply meta-heuristic algorithms [8] to large-scale task scheduling problems in a complex cloud environment, such as particle swarm optimization algorithm (PSO) [9], ant colony algorithm (ACO) [10], chicken swarm optimization algorithm (CSO) [11], etc. Due to different solving strategies, meta-heuristic algorithm can search results quickly, but it is easy to fall into local optimal in different tasks. Recently, based on the advantages of different heuristic algorithms, Hyper-heuristic algorithms have been applied in task scheduling problems [12] and get a more comprehensive search range and better scheduling results. Tsai et al. [13] proposed algorithm provides balanced scheduling solutions by employing the honey bee load balancing and improvement detection operator to conclude which low-level heuristic is to be utilized to search improved candidate solutions. However, the high-level selection strategy still relies on human experience and does not dynamically adjust the strategy along with the calculation process.

(b) The task's computational complexity, that is, the time consumed in the unit computing power, cannot be accurately estimated in practical application scenarios. The computational complexity of different tasks cannot be expressed by a single linear relationship, which makes it difficult for us to obtain the computational complexity as accurate as in the simulation environment. Moreover, task complexity is critical to the selection of underlying algorithm and the overall algorithm effect, especially in real application scenarios.

This work proposes a hyper-heuristic algorithm based on reinforcement learning (HHRL). HHRL uses a high-level heuristic based on a reward table updated with iterations. The makespan and the population diversity form the state information. PSO [14], Fuzzy PSO (F-PSO) [15], Genetic algorithm (GA) [16], and ACO [17] are used to form the action set. The high-level heuristic strategy is based on the reward table, which is updated with iteration. The increase of population diversity and the decrease of makespan will increase the reward value of this action. The high-level heuristic strategy tends to choose the action with the highest reward value. A candidate solution migration mechanism based on random perturbation is also proposed to ensure the continuity of the solving process. HHRL can effectively improve the population diversity, expand the search range, and get better scheduling results.

To accurately predict the computational complexity of tasks, this paper proposes a task complexity estimation method based on linear regression. Although it is difficult to estimate the computational complexity of different categories of tasks, there is an obvious linear relationship in the similar tasks. We record the execution of 100 tasks in each category and explore the linear relationship among them. The results show that the linear relationship exists obviously and can significantly improve task complexity estimation accuracy.

The remainder of the paper is organized as follows. The related work of task scheduling in cloud computing is given in Section 2. Section 3 provides the introduction of the scheduling problem in cloud environment and the relevant technologies of hyper-heuristic algorithms. Section 4 describes the proposed algorithm in detail. The experiment results, on both CloudSim and the real cloud server, are discussed in Section 5. Finally, Section 6 concludes the this work and presents the future research.

2 Related Work

Effective scheduling of tasks submitted by users can effectively reduce the cost and reduce resource consumption of the cloud computing center. From rule-based algorithms to meta-heuristic algorithms, and then to hyper-heuristic algorithms, scheduling algorithm has a wider search range, and gradually has the evolutionary ability.

Plenty of scheduling algorithms is rule-based. Zhu et al. [18] proposed an energy-aware scheduling algorithm based on rolling-horizon scheduling architecture named EARH for real-time, aperiodic, independent tasks. EARH is easy to implement and has strong versatility. However, it does not take into account the differences in the deadlines of each task. Javadpour et al. [19] proposed a scheduling algorithm that dynamically provisions the resource according to the deadline, but it cannot effectively reduce makespan. Javadpour et al. [20] proposed an algorithm that prioritizes the tasks regarding their execution deadline and reduce the consumed energy of the machines processing the low-priority tasks using the DVFS method. To solve this problem, Coninck et al. [21] proposed an adaptive modular nonlinear job scheduling algorithm with parallelism in the meteorological cloud. Compared with the previous methods, the average and optimal makespan values obtained by this method are reduced by 10%. Besides, some algorithms can achieve multi-objective optimization. GIoTDVFS_mGA [22] based on GA was proposed to to balance workload and reduce energy consumption using Dynamic Voltage Frequency Dcaling and microgenetic algorithm. Yu et al. [23] proposed a Runtime Balance Clustering Algorithm (RBCA) based on backtracking approach to improve the load balance of each cluster. Zhu et al. [24] proposed a Multi-stage scheduling algorithm based on LSHP and ALSHP search techniques. This algorithm minimizes the renting VMs and increases or decreases the resources according to upcoming requests. The above-mentioned rule-based methods are stable and easy to implement. However, they cannot change the scheduling strategy according to the different environment information. As a result, there is still much room for improvement.

Representative heuristic algorithms are also used in task scheduling. Chaudhary et al. [25] proposed Hybrid Genetic-Gravitational Search Algorithm (HG-GSA) to reduce the total cost of computation. Jacob et al. [26] proposed CPSO by combining the advantages of Cuckoo Search (CS) and PSO. CPSO is effective in reducing the makespan of the workflow. Gu et al. [27] proposed a bat-based algorithm called Energy Aware, Time, and Throughput Optimization heuristic (EATTO).

EATTO can effectively save energy consumption and minimize the makespan of computation-intensive workflows. Xie et al. [28] proposed a directional and non-local-convergent particle swarm optimization (DNCPSO), which minimize the execution time and cost dramatically. By the directional search process based on non-linear inertia weight with selection and mutation operation, DNCPSO acquires a balanced result. Although these mata-heuristic algorithm of PSO [29–31], GA [32–34] and ACO [35–37] have a wider search range and faster operation speed. Although they obtained better scheduling schemes, there is still much room for improvement. Because a heuristic algorithm is designed for a special problem, it is easy to fall into local optimum when solving problems in other domains.

To get better solutions, Hybrid heuristic algorithms [38], evolutionary algorithms [39], and Q-learning [40] are also used to solve scheduling problems. However, there is a problem with excessive computational overhead. At the beginning of the iteration, the candidate solutions obtained are inferior. Compared with the above methods, hyper-heuristic also has better learning ability, and will not bring too complicated calculations. Tsai et al. proposed a hyper-heuristic scheduling algorithm (HHSA) [12]. By diversity detection and improvement detection operators, HHSA dynamically determine which low-level heuristic will be selected to find better scheduling solutions. In order to optimize scientific workflow scheduling cost in a cloud environment, Alkhanak et al. [41] proposed a Completion Time-Driven Hyper-Heuristic (CTDHH) approach. Panneerselvam et al. [42] discussed the application of hyper-heuristics in resource supply for MapReduce workflow execution in IaaS cloud. Although the above work can obtain better scheduling results than traditional methods, their high-level selection strategies are based on preset rules and do not have the ability of online learning. Besides, Lin et al. [40] proposed a genetic programming hyper-heuristic (GP-HH) algorithm to address the Multi-skill resource-constrained project scheduling problem. Pahlevan et al. [43] proposed a hybrid approach using genetic programming hyper-heuristics combined with rules designed in advance to solve the two-level container allocation problem. Laboni et al. [44] presented a hyper-heuristic (AWSH) algorithm by leveraging the combined powers of A nt Colony, Whale, Sine-Cosine, and H enry Gas Solubility Optimization algorithms at the higher level to adapt to changes in 5G network parameters. Zade et al. [45] presented a hyper-heuristic ALO (HH-ALO-Tabu) that automatically chooses CMs, OBLs, and random walk strategies depending on the differential evolution (DE) algorithm. The above work uses a hyper-heuristic algorithm to solve the task scheduling problem in a cloud environment. However, researchers often set the existing hyper-heuristic algorithms according to their experience, and its selection strategy cannot evolve with iteration.

3 Background and Problem Formalization

This section describes the task scheduling problem and the hyper-heuristic scheduling algorithm in Cloud Computing.

3.1 Task Scheduling Problem

Cloud Computing Center virtualizes physical resources and establishes multiple VMs with different performances. In practical applications, since the creation and initialization of virtual machines will lead to unnecessary consumption of time and energy, users' tasks are generally executed on existing VMs.

Part of the performance of VM_n can be expressed as $\{pesnumber_n, mip_n, bw_n\}$, where bw_n represents bandwidth used to transmit related data, mip_n represents the information processing speed of CPU, and $pesnumber_n$ represents number of CPU cores. The characteristics of the $task_m$ are expressed as

$\{complexity_m, size_m, result_m\}$, where $complexity_m$ represents the computational complexity of the task, $size_m$ represents the related data size of the task, and $result_m$ represents the related data size of the calculated result.

The essence of the task scheduling algorithm in cloud computing is to schedule the task to the appropriate virtual machine and complete the execution in less time. Since VMs and tasks have various characteristics, the results of different scheduling schemes will be significantly different. For example, $task_m$ only needs to upload a small number of related data, but the computational complexity is exceptionally high. The bottleneck of this task is the information processing speed but not bandwidth. The task can be executed quickly on the virtual machine with high information processing speed and small bandwidth, but with low information processing speed and high bandwidth it will consume a lot of time. The scheduling algorithm's function is to search for a better solution for solving the current task sequence in the vast solution space. This paper discusses the scheduling of independent tasks, so there is no transmission between different tasks. The task scheduling problem in cloud computing will be described in detail below.

Cloud Computing System is composed of a large number of Physical Machines (PM).

$$CS = \left[PM_1, PM_2, \ldots, PM_i, \ldots, PM_{N_{pm}}\right]$$

where PM_i represents the physical resources in the Cloud Computing System, and the VMs built on the physical machine can represent as:

$$PM_i = \left[VM_1, VM_2, \ldots, VM_i, \ldots, VM_{N_{VM}}\right]$$

where VM_j represents the VMs created in the PM_i. The performance of VM_j can represent as:

$$VM_n = [pesnumber_n, mip_n, bw_n]$$

where $pernumber_n$ represents the number of CPU cores, mip_n represents the information processing speed of a core, and bw_n is the network bandwidth. Task set T can represent as:

$$T = \left[task_1, task_2, \ldots, task_m, \ldots, task_{N_{task}}\right]$$

The characteristics of the $task_m$ can represent as follows:

$$task_m = [complexity_m, size_m, result_m]$$

where $complexity_m$ represents the computational complexity of the task, $size_m$ represents the related data size of the task, and $result_m$ represents the related data size of the calculated result.

Assuming that $task_m$ starts to upload to VM_n at SUT_m , the upload cost time (UCT_m) can represent as:

$$UCT_m = \frac{size_m}{bw_n} \tag{1}$$

The time finished upload (FUT_m) is:

$$FUT_m = UCT_m + SUT_m \tag{2}$$

The executing cost time (ECT_m) of $task_m$ on VM_n can represent as:

$$ECT_m = \frac{complexity_m}{pesnumber_n \times mip_n} \tag{3}$$

Assuming that $task_m$ needs to wait for WT_m to execute after completing the upload, the execution finished time (FET_m) of $task_m$ can represent as:

$$FET_m = FUT_m + WT_m + ECT_m \tag{4}$$

The download cost time (DCT_m) of the result can represent as:

$$DCT_m = \frac{result_m}{bw_n} \tag{5}$$

The Expect Finish Time (EFT_m) is:

$$EFT_m = FET_m + DCT_m \tag{6}$$

Assuming that $EFT_{m,n}$ represents the Expect Finish Time for $task_m$ to complete the calculation on VM_n, the matrix of size $N_{task} \times N_{vm}$ denotes the Expect Finish Time required to run the task on each virtual machine:

$$EFT = \begin{bmatrix} EFT_{1,1} & EFT_{1,2} & \ldots & EFT_{1,n} & \ldots & EFT_{1,N_{VM}} \\ EFT_{2,1} & EFT_{2,2} & \ldots & EFT_{2,n} & \ldots & EFT_{2,N_{VM}} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ EFT_{m,1} & EFT_{m,2} & \ldots & EFT_{m,n} & \ldots & EFT_{m,N_{VM}} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ EFT_{N_{task},1} & EFT_{N_{task},2} & \ldots & EFT_{N_{task},n} & \ldots & EFT_{N_{task},N_{VM}} \end{bmatrix}$$

A Feasible Solution (FS) in the domain can represent as:

$$makespan = \max \begin{Bmatrix} EFT_{1,n_1} \\ EFT_{2,n_2} \\ \ldots \\ EFT_{m,n_j} \\ \ldots \\ EFT_{N_{task},n_{N_{task}}} \end{Bmatrix}, n \in [1, \mathrm{N}_{vm}], n \in N^+ \tag{7}$$

3.2 Hype-Heuristic Scheduling Algorithm

In 2001, Cowling proposed the Hyper-heuristic algorithm and used it to solve the scheduling problem. Cowling described the Hyper-heuristic algorithm as the heuristic algorithm to find the heuristic algorithm. With the development of research, Hyper-heuristic algorithms are used to manage or manipulate a series of low-level heuristic algorithms to solve various combinatorial optimization problems [46–50]. Differences between the Hyper-heuristic algorithm and the traditional heuristic algorithm are shown in Table 1.

Table 1: Differences between the hyper-heuristic algorithm and the traditional heuristic algorithm

	Hyper heuristic algorithm	Heuristics algorithm
Search space	Set of low level heuristic algorithm	Solution space of problem
Professional knowledge	Less or no problem domain knowledge is required	Knowledge of intelligent computing and problem domain is required

(Continued)

Table 1: Continued

	Hyper heuristic algorithm	Heuristics algorithm
Scope of application	Can be applied to different problems	To solve new problems, redesign is generally needed

Generally speaking, the goal of the Hyper-heuristic algorithm aims to extend the intelligent computing technology to more fields and reduce the difficulty of heuristic algorithm design. A typical Hyper-heuristic algorithm consists of a control domain and problem domain. The problem domain contains the constraints, basic functions, evaluation functions and low-level heuristics (LLH) designed by domain experts. The high-level heuristic in the control domain is designed by the hyper heuristics experts, including how to use the low-level heuristics to construct feasible solutions or improve the quality of solutions. The standard interface between the problem domain and the control domain is defined for information transfer between two layers.

The task scheduling application of generally Hyper-heuristics algorithm in cloud computing is shown in Fig. 1, and the pseudocode is shown in Algorithm1.

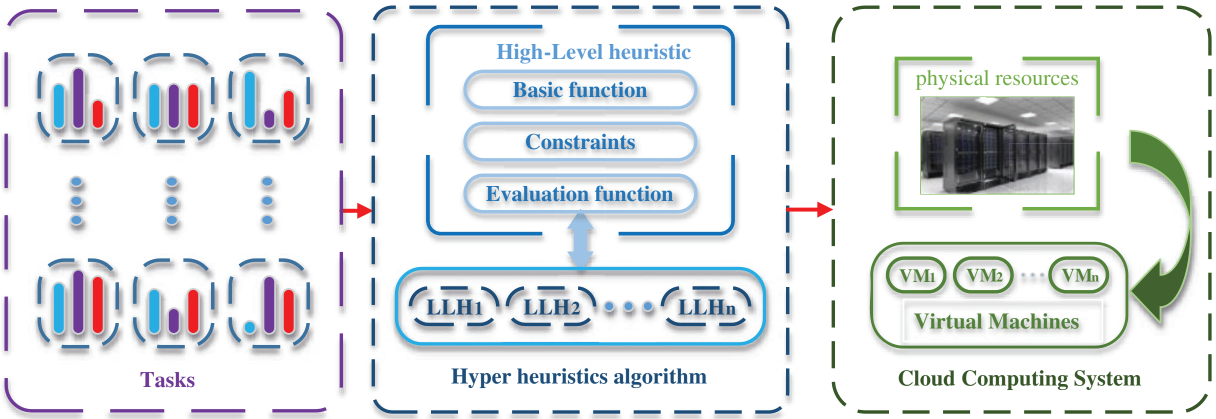


Figure 1: Hyper heuristics algorithm in cloud computing

Algorithm 1: Hyper heuristic algorithm in cloud computing

```
1:  **Input** the scheduling problem
2:  **Output** the best solution as the final solutions
3:  Initialize the population of solution $Z = \{z_1, z_2, \ldots, z_n\}$
4:  Initialize the parameters of Hyper heuristics algorithm
5:  Randomly select a Low-Level heuristic algorithm $LLH_i$
6:  **While** ($t < t_{max}$) **do**
7:     Update the population of solutions$Z$ by $LLH_i$
8:     Select a new $LLH_i$ by the High-level heuristic
9:     Update the High-level heuristic
10: **End while**
```

The flow of the whole algorithm is shown in Fig. 2. Firstly, the population will be initialized and randomly selected LLH for iterative and get the corresponding evaluation function. The high-level selection strategy will get the reward value according to the evaluation function and update the reward table. Then, the high-level selection strategy will select LLH based on the reward table and migrate some high-quality candidate solutions. Finally, the population will be initialized again and iterative to get a solution until the termination condition is reached.

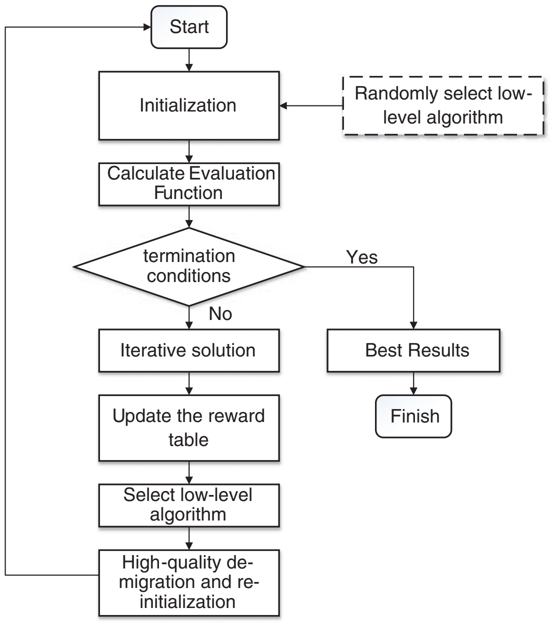

Start
Initialization
Randomly select low-level algorithm
Calculate Evaluation Function
termination conditions
Yes
No
Iterative solution
Best Results
Finish
Update the reward table
Select low-level algorithm
High-quality de-migration and re-initialization

Figure 2: The flow of hype-heuristic SCHEDULING Algorithm

4 The Proposed Approach

In this section, we present the hyper-heuristic algorithm based on reinforcement learning to solve the task scheduling problem in cloud computing. Moreover, task computational complexity estimation method is used to accurately obtain the execution cost time of the task.

4.1 Hyper Heuristic Algorithm Based on Reinforcement Learning

To learn a high-level heuristic method based on current state information, HHRL uses makespan and population diversity to construct reward-table. The pseudocode of HHRL is shown in Algorithm 2.

Algorithm 2: Hyper heuristic algorithm based on reinforcement learning

```
1:  **Input** the scheduling problem
2:  **Output** the best solution as the final solutions
3:  Initialize the population of solution $S = \{X_1, X_2, \ldots, X_i, \ldots, X_{N_{solution}}\}$
4:  Initialize the parameters of high-level heuristic
5:  Randomly select a low-level heuristic algorithm $LLH_i$
```

(Continued)

Algorithm 2: Continued

```
6:      While (Termination condition not satisfied) do
7:          Initialize the parameters of $LLH_i$
8:          While ($t < t_{\max}$) do
9:              Update the population of solutions $S$ by $LLH_i$
10:             Get the makespan of the optimal solutions
11:             Get the population diversity
12:         End while
13:         Calculate the reward value of the current iteration
14:         Update the Reward-table
15:         Select LLH according to Reward-table
16:         Transfer of candidate Solutions
17:     End while
```

4.1.1 Action Set

Action set is composed of ACO, GA, F-PSO and PSO. Although the particle swarm can quickly find the solution, the quality is not stable. Fuzzy-PSO improves the efficiency of conventional PSO by using fuzzy logic systems, and it's less convergent. Although ant colony algorithm can search the optimal solution quickly, it is also easy to fall into local optimal. Because the sequence of candidate solutions generated by GA is very suitable for representing task scheduling, GA has a good effect in task scheduling, which is also proved by experiments.

4.1.2 State Information

The state information consists of makespan and the population diversity. The calculation method of makespan is introduced in Section 3. Because in meta-heuristic algorithm, other candidate solutions will be fitted to the optimal solution. The difference between the optimal solution and other candidate solutions can indicate the diversity of the current candidate solution set. In HHRL, the population diversity is represented by the Hamming distance between the optimal solution and other candidate solutions. The set S of candidate solutions can represent as:

$$S = \{X_1, X_2, \ldots, X_i, \ldots, X_{N_{solution}}\}$$

where the candidate solution X_i can represent as $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,j}, \ldots, x_{i,N_{task}}\}$ and the optimal solution can represent as $B = \{b_1, b_2, \ldots, b_j\}$. Use *popsize* represents the population size of LLH. Then the population diversity can represent as:

$$distance = \sum_{i=1}^{N_{solutiong}} \sum_{j=1}^{N_{task}} (b_j \oplus x_{i,j}) \tag{8}$$

$$diversity = \frac{distance}{popsize \times N_{task}} \tag{9}$$

Eqs. (8) and (9) are utilized to calculate the population diversity and supervise the selection process of the underlying algorithm. In Eq. (8), $x_{i,j}$ denotes i-th solution j-th position. b$_j$ represents i-th position of the best individuals. N_{task} denotes the total number of tasks. $N_{solution}$ denote the number of solutions. $\oplus$ represents exclusive or operation. It is used to calculate the diversity index of the population and supervise the selection process of the underlying algorithm.

4.1.3 Update of Reward-Table

The high-level heuristic method selects a low-level heuristic algorithm (LLH). After the iterations of the LLH, the execution result of the current algorithm can be obtained. The reward value is consists of two parts. The first part of the reward value determined by the makespan of the current iteration result and the makespan obtained from the previous iteration. The reward value of the current LLH iteration can represent as:

$$R_1(s,a) = \frac{1}{1+e^{\lambda}} \qquad \lambda = \frac{-\sigma \times makespan}{N_{num}} \tag{10}$$

where a represents the action information, and s represents the state information, which is composed of the value interval of makespan and population diversity. σ represents the constant can ensure that makespan significantly influences on R1 in the effective range and has a certain marginal effect outside the range. The update formula of reward value can represent as:

$$R_1(s,a) = R_1 + a\left[r + \lambda max_{a'} R_1'(s,a) - R_1(s,a)\right] \tag{11}$$

where $\max_{s'} R'(s', a')$ represents the maximum expect future reward in the new state s'.

$$R_2(s,a) = diversity_n - diversity_{n-1} \tag{12}$$

where $diversity_n$ represents the unit population diversity in the current iteration, and $diversity_{n-1}$ represents the unit population diversity in the last iteration. The total reward value can represent as:

$$R(s,a) = R_1(s,a) + R_2(s,a) \tag{13}$$

4.1.4 Selection of LLH

When selecting the LLH, the high-level heuristic is determined by the reward value of the current reward-table under the state information. LLH with the highest reward value will have a 50% probability to be selected, and other actions will be selected randomly.

4.1.5 Transfer of Candidate Solutions

The candidate solution $X_i = \left\{x_{i,1}, x_{i,2}, \ldots, x_{i,j}, \ldots, x_{i,N_{task}}\right\}$ consists of a sequence of numbers with the length of N_{task}. Where $x_{i,j} \in [1, N_{vm}], x_{i,j} \in N^+$, that is, the value of the sequence indicates that $task_j$ is executed on the $VM_{x_{i,j}}$. After the current LLH completes the iteration, HLH will select the LLH method again. The current candidate solution information needs to be transferred to a new round of LLH iteration to realize the continuation of the solution. However, with the iteration of an LLH, the set of candidate solutions tends to approach the current optimal solution, and the population diversity will also decrease. In order to ensure the population diversity, only the optimal solution and the five suboptimal solutions satisfying the different conditions are reserved when the candidate solutions are migrated. 50% of the candidate solutions will be generated by random perturbation on the above transferred solutions. The remaining candidate solutions will be generated randomly.

4.1.6 The Complexity of Proposed Algorithm

Generally speaking, the computational complexity of the heuristic algorithm is , where represents the iteration round of the algorithm, M represents the length of feasible solutions, and N represents the number of candidate solutions.

The algorithm proposed in this paper is solved by the selected LLH, and the computation cost of high-level selection strategy is very small, so the complexity of the algorithm proposed in this paper is very close to .

4.2 A Task Computational Complexity Estimation Method

The essence of the task scheduling algorithm in cloud computing is to schedule the task to the appropriate virtual machine and complete the execution in a short time. For example, CPU intensive tasks ought to be assigned to high information processing speed resources, but I/O-intensive tasks are more suitable for assignment to resources with large bandwidth.

In the simulation environment, the characteristics of the tasks can be accurately obtained. However, although the category of task and the size of relevant data can be accurately obtained in the real application environment, its computational complexity can't be accurately estimated. Because the computational complexity of a task is related to the task category, it is not linearly related to the size of relevant data. Computational complexity is vital priority information in task scheduling. Inaccurate computational complexity will lead to the degradation of scheduling algorithm performance and failure to obtain high-quality scheduling results. A method to estimate the complexity of three common cloud computing tasks is proposed. For a particular category of task, its computational complexity is often related to the relevant data's size. The regression formula of task complexity can represent as:

$$complexity = a + b \times X + e \qquad (14)$$

where a represents the initialization cost of related resources, b represents the key parameter of the formula, X represents the related data size and e represents error term. After testing in the practical application environment, the method can fit the task's actual calculation cost. The experimental results will be shown in Section 5.

5 Experiments and Discussion

In this section, the proposed method HHRL is compared with other traditional and meta-heuristic algorithms. The empirical results in simulation environment Cloudsim and real tasks show that the proposed HHRL keeps the population diversity effectively and outperforms other algorithms.

5.1 Experiments in CloudSim

In this experiment, CloudSim 4.0 is used to test the algorithms. The performance of HHRL is evaluated and compared with existing meta-heuristics algorithms such as PSO, F-PSO, GA and ACO, and traditional algorithms such as FIFO and Max-min. The parameters of meta-heuristic algorithms are given in Table 2.

Table 2: Parameter setting

Algorithm	Parameters	Value
HHRL	The iterations of LLH	50
	Population size	100
HHSA	The max iterations of LLH	50
	The max iterations of no-improved LLH	5
	Threshold of diversity	2500

(Continued)

Table 2: Continued

Algorithm	Parameters	Value
GA	Crossover problem	0.8
	Mutation rate	0.01
PSO	Inertia weight	0.8
	Cognitive coefficient c_1	1.0
	Cognitive coefficient c_2	1.0
ACO	Pheromone residue coefficient	0.5
	Information elicitation factor	0.5
	Relative influence weights	1

Table 3 shows the Cloudsim experimental settings. We experiment in 500, 1000, 3000, 5000 tasks and compared with other algorithms to verify the scalability of the proposed algorithm. The performance of the algorithm is analyzed by optimal result, average result and variance. The experiment also analyzed the relationship between LLH selection and state information.

Table 3: Cloudsim parameters setting

Entity	Parameters	Value
Cloudlet	No of Cloudlets	500–5000
	length	500–15000
Virtual machine	No of VM	10
	RAM	1024–4096 MB
	MIPS	500–5000
	Bandwidth	1000–5000
	Policy type	Time shared
	VMM	Xen
	Operation system	Linux
	No of cores	1–2
Physical machine	No of PM	2
	RAM	20480 MB
	Storage	100000 MB
	Bandwidth	20000
	Policy type	Time shared

5.1.1 Comparison Between HHRL and Traditional Algorithms

The result of each experiment is the makespan of the current scheduling scheme. The average result and best result obtained by the traditional algorithms and the proposed HHRL algorithm after 30 times are given in Table 4. The convergence process of makespan is shown in Fig. 3, and variance of the results is given in Fig. 4. From these results we can see that HHRL outperforms other scheduling algorithms under different number of tasks.

Table 4: Experiment results in Cloudsim

No.	Algorithm	HHRL	HHSA	GA	ACO	F-PSO	PSO	FIFO	Max-min
500	Best	**155.43**	**155.43**	**155.43**	158.43	157.50	188.47	218.52	162.85
	Average	**158.10**	159.42	160.21	163.40	162.83	203.90	218.52	162.85
1000	Best	**295.61**	300.89	302.07	313.28	310.77	382.58	427.42	324.62
	Average	**300.28**	303.16	307.04	317.72	321.18	395.81	427.42	324.62
3000	Best	**959.60**	962.85	973.24	992.86	1000.85	1022.67	1169.32	1011.53
	Average	**962.17**	966.73	978.17	1002.39	1011.61	1036.27	1169.32	1011.53
5000	Best	**1597.09**	1600.29	1600.02	1623.83	1632.36	1681.83	1826.54	1674.63
	Average	**1600.35**	1603.92	1607.43	1633.69	1646.41	1697.83	1826.54	1674.63

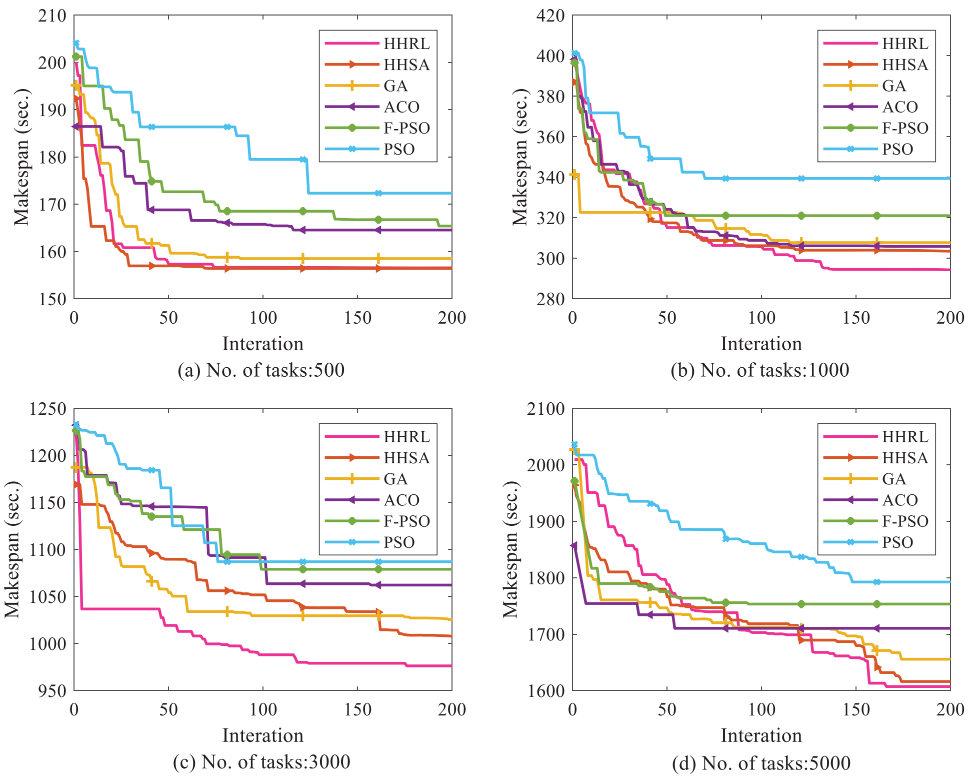


Figure 3: Convergence process of makespan

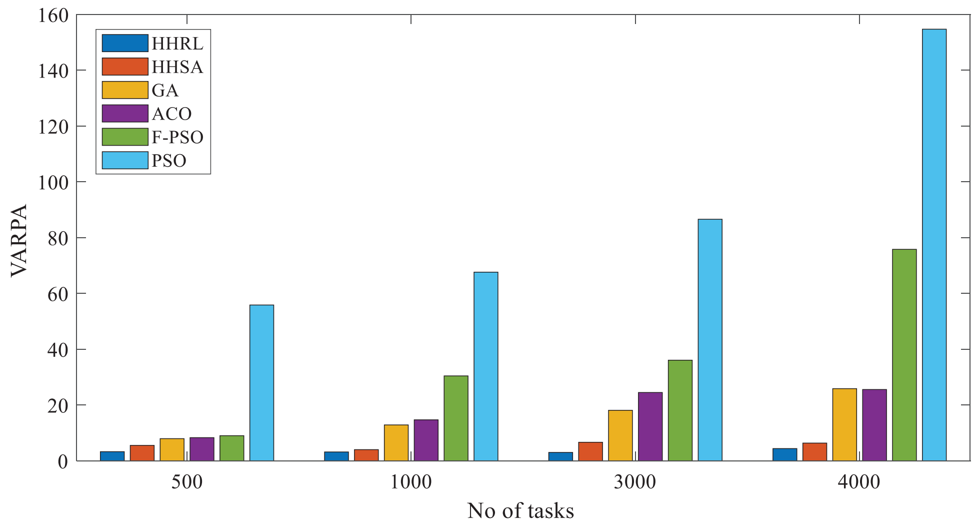


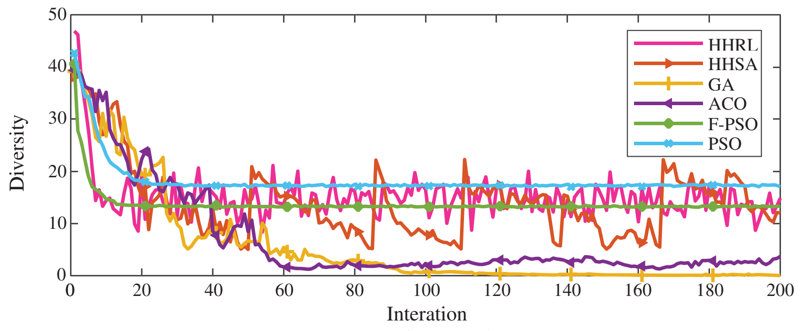
Figure 4: The variance of the different number tasks

As shown in Table 4, although the number of tasks is different, HHRL can almost get smaller best and average results. This means that HHRL can get a better scheduling scheme than several other algorithms. When the number of tasks is 500, HHRL, the space for selecting solutions is small. Thus, HHSA and GA get the same best results. As shown in Fig. 4, the variance of HHRL and HHSA is significantly smaller than other methods. For example, the difference of best results between GA and HHSA is similar, but the average result and variance difference is remarkable. On the other hand, as the number of tasks increases, the variance does not increase significantly as other methods. It indicates that the performance of the two hyper-heuristics is more stable. Since the hyper-heuristics can bring a more comprehensive search range and significantly avoid falling into local optimum, high-quality solutions can be obtained in most iterations.

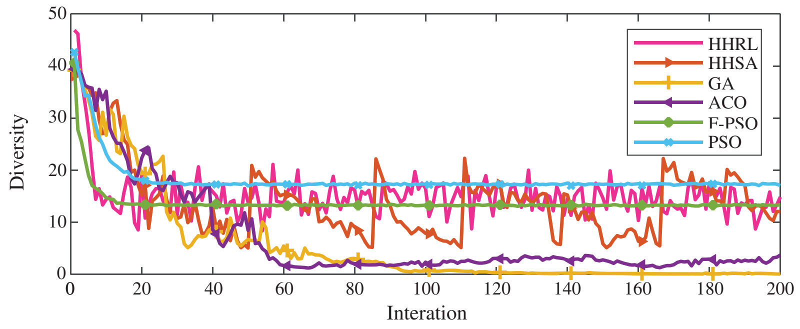
5.1.2 Influence of Population Diversity

In the iteration of the meta-heuristic algorithm, the population diversity significantly impacts the optimal result. With the continuous fitting of candidate solutions to the optimal solutions, the population diversity will decrease. If the population diversity decreases too fast, it is easy to fall into local optimum, and better candidate solutions cannot be obtained. The variation of population diversity of HHRL and other meta-heuristic algorithms is shown in Fig. 5.

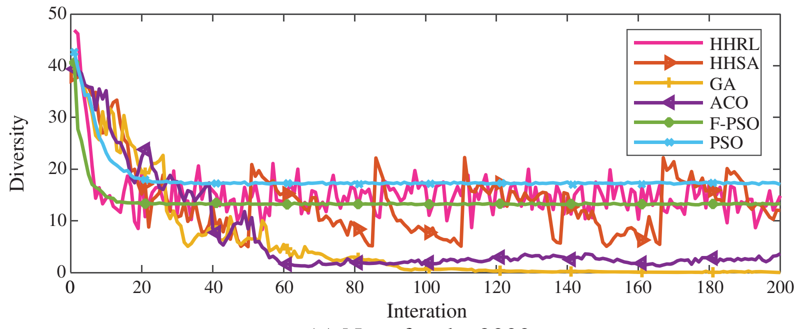
From this result, it can be observed that the population diversity of PSO and F-PSO will gradually decrease, then it tend to be stable and maintain at a high level. The high population diversity means that PSO falls into local optimum in many places, and no high-quality scheduling results are found. On the contrary, the population diversity of ACO and GA can get convergence and better scheduling results. However, the set of candidate solutions is completely fitted to the optimal solution, which leads to serious local optimization. HHSA has a diversity detection operator. When the population diversity is lower than the preset threshold, disturbance information will be introduced to improve the population diversity. As a result, the population diversity varies dramatically. HHRL rewards the population diversity, which effectively avoids the excessive convergence or violent fluctuation of population diversity. This ensures the balance between search range and algorithm efficiency.



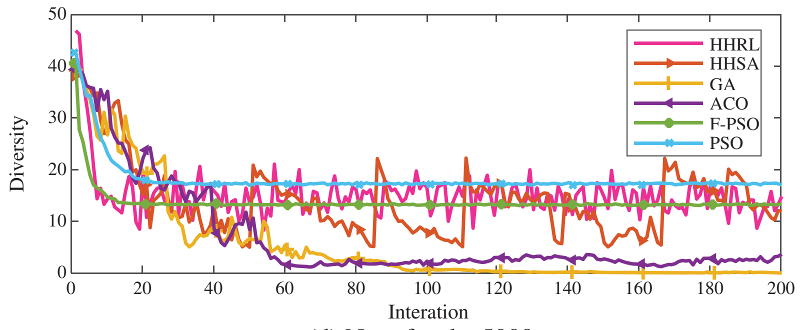
(a) No. of tasks:500



(b) No. of tasks:1000



(c) No. of tasks:3000



(d) No. of tasks:5000

Figure 5: The curve of population diversity

5.1.3 The Relationship Between Population Diversity and LLH Selection

The population diversity as HHRL state information directly affects the selection of LLH. To analyze the influence of the population diversity on LLH selection, this work counted the 6,000 selections of LLH obtained from 30 runs of HHRL, and the results obtained are shown in Fig. 6. Taking the population diversity as the criterion, the population diversity is divided into three cases: high, medium and low. The classification of the three situations can represent as:

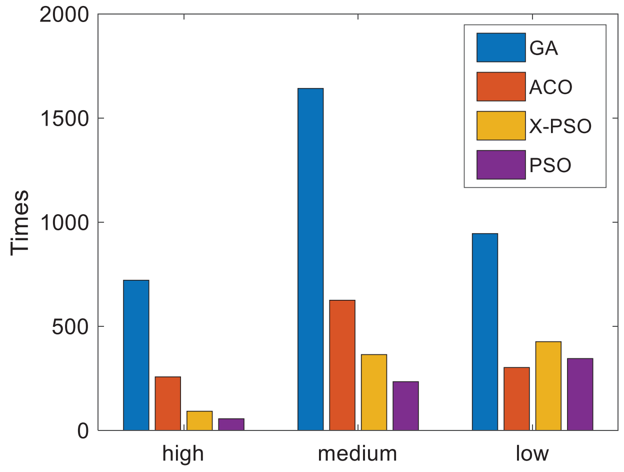


Figure 6: The selected frequency of LLH in each state

$$case = \begin{cases} low & diversity > 10 \\ medium & 5 \le diversity \le 10 \\ high & diversity < 5 \end{cases} \tag{15}$$

In the experiment, there were 1126 high, 2856 medium and 2018 low. When population diversity is in a high state, GA is selected most, while PSO and F-PSO are rarely selected. At the beginning of HHRL iteration, the increase of the population diversity leads to small reward value, while the decrease of makespan can bring more reward value. Therefore, GA, which can significantly reduce makespan, is selected multiple times. On the contrary, when population diversity is in a low state, PSO and F-PSO are selected more times. Because PSO and F-PSO can effectively maintain the population diversity, it can bring large reward value.

5.2 Experiment in Actual Tasks

In the simulation environment, the computational complexity of the task is considered to be accurately obtained. But in practical application, because of the different categories of tasks, it is not easy to estimate the complexity of tasks in practical application scenarios. This significantly affects the scheduling effect. Their computational complexity does not show a simple linear relationship with the size of related data for all tasks because the task categories are different. For example, training the neural network model with a data set of 2000 MB on the virtual machine consumes significantly more computing power than the only test. However, there is a linear relationship between the data size and the computational complexity for a certain task category. This makes it possible for us to estimate the computational complexity. To verify the effect of this method on improving scheduling effect, linear regression analysis was carried out on three categories of tasks: CNN model training, CNN model testing and RNN model testing. The experimental environment is shown in Table 5. The experimental results are shown in Fig. 7 and Table 6.

Table 5: Characteristics of hosts

	Configuration
CPU	Intel Core i5-8500 3.00 GHz
GPU	Nvidia GeForce GTX 1080Ti
RAM	16G
Operating System	Linux
Storage	1 TB
Policy	Time Shared

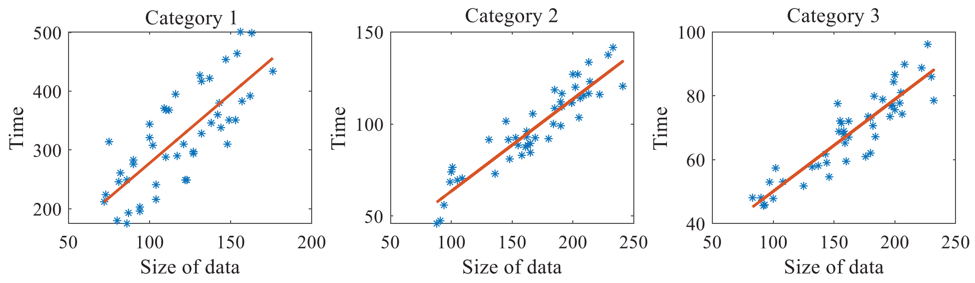


Figure 7: Linear regression of task complexity

Table 6: Analysis of linear regression

		Slope	RMSE	R-square	P-value
Category 1	CNN model training	2.345	54.5	0.599	1.74e-10
Category 2	CNN model testing	0.501	8.73	0.833	3.84e-37
Category 3	RNN model testing	0.290	5.53	0.819	1.67e-18

CNN model includes a classification model and a generative model based on VGG-16 and Resnet-34. RNN model includes text generation based on GRU. As shown in Fig. 6 and Table 6, there is a significant linear relationship between the computational complexity of same type tasks and relevant data size. The three linear relationships have obviously different slopes, and the slope of category1 is exceptionally high. RMSE is sensitive to large or small errors in a group of measurements and can well reflect prediction accuracy. The RMSE value of category 1 is significantly higher than that of other categories. It is due to the randomness of the CNN training process, which leads to the fluctuation of actual computing cost. R-square is a statistical index reflecting the reliability of dependent variables, indicating the degree of linear regression fitting. The R-square of category 2 and category 3 is more significant than 0.8, which indicates that the computational complexity can be accurately predicted. The P-values of the three categories are all less than 0.05, which indicates that the chance of not meeting the above linear relationship is extremely small.

The experimental steps are as follows. Firstly, the execution time of 100 tasks in each category on the host is obtained, which is called complexity 1, and the linear relationship between the computational complexity of different categories of tasks and the size of relevant data is obtained. Then the computational complexity of all tasks will be predicted, which is called complexity 2. In addition, there is a complexity 3 based on the size of the relevant data. In Cloudsim, scheduling result 1 based on complexity 3 and scheduling result 2 based on complexity 2 are obtained. It should be noted that complexity 1 is the real computational power consumption, while complexity 2 and complexity 3 are estimated. Two scheduling results are obtained through complexity2 and complexity 3, but the real makespan of the two scheduling results still needs to be obtained according to complexity 1. The average result and best result of 30 times between the traditional algorithms and the proposed HHRL algorithm are given in Table 7 and Fig. 8.

Table 7: Experiment results of real tasks

		HHRL	HHSA	GA	ACO	F-PSO	PSO	FIFO	Max-min
Result 1	Best	82.57	82.04	83.51	82.63	82.60	82.78	94.56	91.57
	Average	88.12	89.70	91.06	90.46	93.36	93.47	94.56	91.57
Result 2	Best	**80.96**	**80.96**	**80.96**	**80.96**	82.16	82.58	94.38	84.39
	Average	**84.59**	85.80	86.22	87.75	89.42	92.37	94.38	84.39

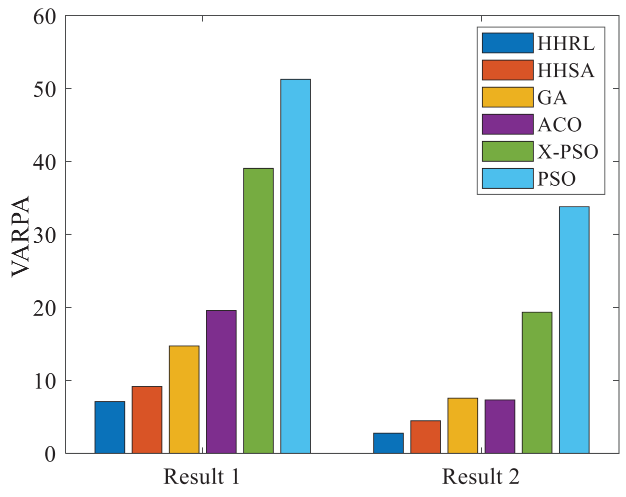


Figure 8: The variance of the real tasks

As shown in Table 1, the computational complexity can't be accurately estimated, which significantly increases the randomness and contingency of scheduling. Several meta-heuristic algorithms get similar optimal values and average values. At the same time, Result 2 in Table 2 is similar to the simulation experiments in Section 5.1. HHRL can always get smaller best values, average values and variance. This means that HHRL can get better scheduling results, and it has a stable effect. Compared with result1 and result2, the scheduling effect of each algorithm is improved. This shows that the proposed method is effective.

From the experiment results in this section it can be concluded that, the proposed HHRL algorithm get better results than the traditional scheduling algorithms. Experiments in cloudsim show that HHRL can effectively reduce makespan and improve the stability of the scheduling scheme. This improvement comes from the reward for the improvement of population diversity. The results of real tasks show that makespan can be effectively improved after the task complexity is estimated, and the experimental conclusion is similar to that in the simulation environment.

6 Conclusion

This work proposed a hyper-heuristic algorithm based on reinforcement learning (HHRL). HHRL obtained reward value by makespan and population diversity. The action set of HHRL was formed by four mata-heuristic algorithms-GA, ACO, PSO, and F-PSO. By rewarding the decrease of makespan and improving population diversity, HHRL kept the population diversity and got a better scheduling scheme. This work also proposed a task computational complexity estimation method in a cloud environment by linear regression to reduce task complexity estimation error. Firstly, we carried out experiments from 500 to 5000 tasks in Cloudsim. The results of 30 times executed of each algorithm showed that HHRL could get better average results and optimal results than other algorithms. This result also showed that the variance of HHRL was the smallest, which indicated that the solution effect of HHRL was the most stable. By analyzing population diversity, we found that HHRL can effectively maintain population diversity by rewarding the improvement of population diversity. Simultaneously, the relationship between the selected frequency of LLH and the population diversity was also explained. On the other hand, the task complexity estimation method proposed in this work was verified to be effective. Experiments show that after the estimation of this method, the effects of each scheduling algorithms are significantly improved.

In the future, we plan to study a hyper-heuristic scheduling algorithm that optimizes multi-objective such as load balancing, energy consumption, and user expenditure in a cloud environment. We also plan to implement another HHRL version that can schedule workflow. At the same time, we are also committed to research a method to achieve synchronous fitting of various task computational complexity in the iterative process of the scheduling algorithm.

Acknowledgement: No further acknowledgments to provide than those already listed in the authors' section. In addition, the authors thank the anonymous reviewers for providing valuable comments to improve this paper.

Funding Statement: This work was supported in part by the National Key R & D Program of China under Grant 2017YFB1302400, the Jinan "20 New Colleges and Universities" Funded Scientific Research Leader Studio under Grant 2021GXRC079, the Major Agricultural Applied Technological Innovation Projects of Shandong Province under Grant SD2019NJ014, the Shandong Natural Science Foundation under Grant ZR2019MF064, and the Beijing Advanced Innovation Center for Intelligent Robots and Systems under Grant 2019IRS19.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] M. Mishra, A. Das, P. Kulkarni and A. Sahoo, “A dynamic resource management using virtual machine migrations,” *IEEE Communications Magazine*, vol. 50, no. 9, pp. 34–40, 2012.

[2] I. Mavridis and H. Karatza, “Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing,” *Future Generation Computer Systems*, vol. 94, no. 1, pp. 674–696, 2019.

[3] F. D. Prieta, S. Rodríguez, J. Bajo and J. M. Corchado, “A multiagent system for resource distribution into a Cloud Computing environment,” in *Int. Conf. on Practical Applications of Agents and Multi-Agent Systems*, Berlin, Heidelberg, Springer, vol. 1, pp. 37–48, 2013.

[4] N. M. Donnell, E. Howley and J. Duggan, “Dynamic virtual machine consolidation using a multi-agent system to optimise energy efficiency in cloud computing,” *Future Generation Computer Systems*, vol. 108, no. 1, pp. 288–301, 2020.

[5] B. Radunovic and J. Y. LeBoudec, “A unified framework for max-min and min-max fairness with applications,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1073–1083, 2007.

[6] M. Patidar, R. Bhardwaj and S. Choudhary, “The study of linear programming approach for optimal scheduling of work in a corporation with different models,” *Materials Today: Proceedings*, vol. 29, pp. 661–667, 2020.

[7] A. Javadpour and G. Wang, “cTMvSDN: Improving resource management using combination of Markov-process and TDMA in software-defined networking,” *The Journal of Supercomputing*, vol. 1, pp. 1–23, 2022.

[8] M. Kumar, S. C. Sharma, A. Goel and S. P. Singh, “A comprehensive survey for scheduling techniques in cloud computing,” *Journal of Network and Computer Applications*, vol. 143, no. 2, pp. 1–33, 2019.

[9] K. Chen, F. Zhou and A. Liu, “Chaotic dynamic weight particle swarm optimization for numerical function optimization,” *Knowledge-Based Systems*, vol. 139, no. 12, pp. 23–40, 2018.

[10] D. Johann and P. Siarry, “Continuous interacting ant colony algorithm based on dense heterarchy,” *Future Generation Computer Systems*, vol. 20, no. 5, pp. 841–856, 2004.

[11] D. Zouache, Y. O. Arby, F. Nouioua and F. B. Abdelaziz, “Multi-objective chicken swarm optimization: A novel algorithm for solving multi-objective optimization problems,” *Computers & Industrial Engineering*, vol. 129, pp. 377–391, 2019.

[12] A. Gupta, H. S. Bhadauria and A. Singh, “RETRACTED ARTICLE: Load balancing based hyper heuristic algorithm for cloud task scheduling,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 12, pp. 5845–5852, 2021.

[13] C. W. Tsai, W. C. Huang, M. H. Chiang, M. C. C.hiang and C. S. Yang, “A hyper-heuristic scheduling algorithm for cloud,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 236–250, 2014.

[14] M. A. Rodriguez and R. Buyya, “Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[15] H. Liu, A. Abraham and A. E. Hassanien, “Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm,” *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1336–1343, 2010.

[16] C. Jatoth, G. R. Gangadharan and R. Buyya, “Optimal fitness aware cloud service composition using an adaptive genotypes evolution based genetic algorithm,” *Future Generation Computer Systems*, vol. 94, no. 4, pp. 185–198, 2019.

[17] C. Jatoth, G. R. Gangadharan and R. Buyya, “An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem,” *IEEE Access*, vol. 7, pp. 20281–20292, 2019.

[18] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin *et al.,* “Real-time tasks oriented energy-aware scheduling in virtualized clouds,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 168–180, 2014.

[19] A. Javadpour, A. K. Sangaiah, P. Pinto, F. Jafari, W. Zhang *et al.,* “An energy-optimized embedded load balancing using DVFS computing in cloud data centers,” *Computer Communications*, vol. 197, no. 1, pp. 255–266, 2023.

[20] A. Javadpour, A. H. Nafei, F. Ja’fari, P. Pinto, W. Zhang *et al.,* “An intelligent energy-efficient approach for managing IoE tasks in cloud platforms,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 1, no. 1, pp. 1–7, 2022.

[21] E. D. Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens *et al.,* “Dynamic auto-scaling and scheduling of deadline constrained service workloads on IaaS clouds,” *Journal of Systems and Software*, vol. 118, no. 1, pp. 101–114, 2014.

[22] Y. Hao, L. Wang and M. Zheng, “An adaptive algorithm for scheduling parallel jobs in meteorological Cloud,” *Knowledge-Based Systems*, vol. 98, no. 1, pp. 226–240, 2016.

[23] D. Yu, Y. Ying, L. Zhang, C. Liu, X. Sun *et al.,* “Balanced scheduling of distributed workflow tasks based on clustering,” *Knowledge-Based Systems*, vol. 199, no. 1, pp. 105930, 2020.

[24] J. Zhu, X. Li, R. Ruiz, X. Xu and Y. Zhang, “Scheduling stochastic multi-stage jobs on elastic computing services in hybrid clouds,” in *2016 IEEE Int. Conf. on Web Services (ICWS)*, San Francisco, CA, USA, IEEE, pp. 1–10, 2016.

[25] D. Chaudhary and B. Kumar, “Cost optimized hybrid genetic-gravitational search algorithm for load scheduling in cloud computing,” *Applied Soft Computing*, vol. 83, no. 1, pp. 105627, 2019.

[26] T. P. Jacob and K. Pradeep, “A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization,” *Wireless Personal Communications*, vol. 109, no. 1, pp. 315–331, 2019.

[27] Y. Gu and C. Budati, “Energy-aware workflow scheduling and optimization in clouds using bat algorithm,” *Future Generation Computer Systems*, vol. 113, no. 9, pp. 106–112, 2020.

[28] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu *et al.,* “A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment,” *Future Generation Computer Systems*, vol. 97, no. 1, pp. 361–378, 2019.

[29] N. Mansouri, B. M. Zade and M. M. Javidi, “Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory,” *Computers & Industrial Engineering*, vol. 130, no. 1, pp. 597–633, 2019.

[30] Y. Zhang, Y. Liu, J. Zhou, J. Sun and K. Li, “Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing,” *Future Generation Computer Systems*, vol. 112, no. 1, pp. 148–161, 2020.

[31] Z. Miao, P. Yong, Y. Mei, Y. Quanjun and X. Xu, “A discrete PSO-based static load balancing algorithm for distributed simulations in a cloud environment,” *Future Generation Computer Systems*, vol. 115, no. 1, pp. 497–516, 2021.

[32] P. Salza and F. Ferrucci, “Speed up genetic algorithms in the cloud using software containers,” *Future Generation Computer Systems*, vol. 92, no. 1, pp. 276–289, 2019.

[33] Y. Xiong, S. Huang, M. Wu, J. She and K. Jiang, “A Johnson’s-rule-based genetic algorithm for two-stage-task scheduling problem in data-centers of cloud computing,” *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 597–610, 2017.

[34] B. Keshanchi, A. Souri and N. J. Navimipour, “An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing,” *Journal of Systems and Software*, vol. 124, no. 1, pp. 1–21, 2017.

[35] A. Ragmani, A. Elomri, N. Abghour, K. Moussaid and M. Rida, “FACO: A hybrid fuzzy ant colony optimization algorithm for virtual machine scheduling in high-performance cloud computing,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 10, pp. 3975–3987, 2020.

[36] Y. Gao, H. Guan, Z. Qi, Y. Hou and L. Liu, “A multi-objective ant colony system algorithm for virtual machine placement in cloud computing,” *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1230–1242, 2013.

[37] A. M. Kumar and M. Venkatesan, “Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment,” *Wireless Personal Communications*, vol. 107, no. 4, pp. 1835–1848, 2019.

[38] M. A. Elaziz, S. Xiong, K. Jayasena and L. Li, “Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution,” *Knowledge-Based Systems*, vol. 169, no. 1, pp. 39–52, 2019.

[39] X. Ye, S. Liu, Y. Yin and Y. Jin, “User-oriented many-objective cloud workflow scheduling based on an improved knee point driven evolutionary algorithm,” *Knowledge-Based Systems*, vol. 135, no. 1, pp. 113–124, 2017.

[40] J. Lin, L. Zhu and K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Expert Systems with Applications*, vol. 140, pp. 112915, 2020.

[41] E. N. Alkhanak and S. P. Lee, "A hyper-heuristic cost optimisation approach for scientific workflow scheduling in cloud computing," *Future Generation Computer Systems*, vol. 86, no. Suppl. C, pp. 480–506, 2018.

[42] A. Panneerselvam and S. Bhuvaneswari, "Hyper heuristic MapReduce workflow scheduling in cloud," in *2018 2nd Int. Conf. on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, IEEE, pp. 1–9, 2018.

[43] A. Pahlevan, X. Qu, M. Zapater and D. Atienza, "Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1667–1680, 2017.

[44] N. M. Laboni, S. J. Safa, S. Sharmin, M. A. Razzaque, M. M. Rahman *et al.,* "A hyper heuristic algorithm for efficient resource allocation in 5G mobile edge clouds," *IEEE Transactions on Mobile Computing*, vol. 1, pp. 1–13, 2022.

[45] B. M. Zade, N. Mansouri and M. J. Mohammad, "A new hyper-heuristic based on ant lion optimizer and Tabu search algorithm for replica management in cloud environment," *Artificial Intelligence Review*, vol. 1, pp. 1–111, 2022.

[46] I. Chana, "Bacterial foraging based hyper-heuristic for resource scheduling in grid computing," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 751–762, 2013.

[47] G. Koulinas, L. Kotsikas and K. Anagnostopoulos, "A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem," *Information Sciences*, vol. 277, no. 1, pp. 680–693, 2014.

[48] M. A. Elaziz and S. Mirjalili, "A hyper-heuristic for improving the initial population of whale optimization algorithm," *Knowledge-Based Systems*, vol. 172, no. 1, pp. 42–63, 2019.

[49] W. Li, E. Özcan and R. John, "A learning automata-based multiobjective hyper-heuristic," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 59–73, 2017.

[50] S. S. Choong, L. P. Wong and C. P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," *Information Sciences*, vol. 436, no. 1, pp. 89–107, 2018.