




## Research Article

# DAG Scheduling with Communication Delays Based on Graph Convolutional Neural Network

Zhijun Zhang,<sup>1,2,3,4</sup> Qiang-Sheng Hua ,<sup>1,2,3,4</sup> Xiaohui Zhang,<sup>1,2,3,4</sup> Hai Jin ,<sup>1,2,3,4</sup>  
and Xiaofei Liao <sup>1,2,3,4</sup>

<sup>1</sup>National Engineering Research Center–Big Data Technology and System Lab, Wuhan, China

<sup>2</sup>Key Laboratory of Services Computing Technology and System, Wuhan, China

<sup>3</sup>Key Laboratory of Cluster and Grid Computing, Wuhan, China

<sup>4</sup>School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China

Correspondence should be addressed to Qiang-Sheng Hua; [qshua@hust.edu.cn](mailto:qshua@hust.edu.cn)

Received 7 May 2022; Accepted 26 May 2022; Published 5 July 2022

Academic Editor: Wei Li

Copyright © 2022 Zhijun Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In vehicular edge computing (VEC), tasks and data collected by sensors on the vehicles can be offloaded to roadside units (RSUs) equipped with a set of servers through the wireless transmission. These tasks may be dependent of each other and can be modeled as a directed acyclic graph (DAG). The DAG scheduling problem is aimed at scheduling the tasks to the servers to minimize the scheduling length (makespan), i.e., the maximum finish time of all tasks. The conventional heuristic algorithms only utilize partial information of the DAG, so the performance of these algorithms is not stable. The state-of-the-art scheduling method employs the graph neural network to further reduce the makespan. However, this method ignores the fact that there are communication delays between tasks scheduled on different servers. In this paper, we tackle the DAG scheduling problem considering communication delays which makes the problem much more challenging. Our method is based on graph convolutional neural network and reinforcement learning. Experimental results show that our scheduling method reduces the DAG scheduling length by 8% to 15% compared with the representative scheduling strategies based on graph neural network models (GAT, GraphSAGE) and 15% to 25% compared with the conventional algorithms (HEFT, LC, and CPOP) and the sequence-to-sequence model.

## 1. Introduction

With the maturity of cloud computing technology, VCC (vehicular cloud computing) [1] is considered to be a promising method to improve vehicular services. Vehicles with limited resources can offload computing-intensive tasks to the cloud through VCC. However, cloud computing servers may be far away from vehicles running on the road. It may take a long time to transfer tasks from the vehicle to the cloud server and return the calculation results from the cloud server to the vehicle. Thus, VCC might not be suitable for delay-sensitive tasks.

In order to cope with the above issue, researchers proposed vehicular edge computing (VEC) [2, 3]. VEC is a distributed deployment service that extends the computing and storage capacity to the edge of the network. In VEC, a large number

of roadside units (RSUs) will be deployed near the road where the vehicles are driving. Thus, the computing tasks and the data collected by sensors on vehicles no longer need to be offloaded to the cloud servers but directly offloaded to the roadside units equipped with a set of servers through the wireless transmission such as 5G. The remaining issue is how to schedule the tasks to the servers to minimize the scheduling length.

At present, some works [2, 4] have discussed the scheduling problem in VEC. However, these works assume that the tasks to be scheduled are independent of each other. In fact, there may be dependencies between different tasks from the same application. Liu et al. [3] also pointed out the issue, and they proposed a dependency-aware task scheduling algorithm where the tasks are modeled as a directed acyclic graph (DAG). However, they assume different tasks require the same

computing time and the communication time between tasks is ignored.

With the development of artificial intelligence and machine learning, the neural network is also used to solve the DAG scheduling problem. Mao et al. [5] utilized neural networks and reinforcement learning (RL) [6] to learn job-specific scheduling algorithms. However, they ignore the fact that there are nonnegligible communication delays between tasks scheduled on different servers (processors). Note that the communication delay will become zero if the tasks are scheduled on the same processor. Thus, minimizing the DAG scheduling length needs a tradeoff between placing all the tasks on one processor and placing them on all available processors. In this sense, the DAG scheduling problem with communication delays will be much more challenging than the one without considering them [7].

In this paper, we study the node-weighted and edge-weighted DAG scheduling problem where the node weight represents the task computation time and the edge weight represents the communication time (communication delay) between two tasks.

We design a scheduling method based on a two-layer graph convolutional neural network (TLGC). Similar to [5], we employ reinforcement learning [6] to optimize the training of the strategy, which takes the execution time as the reward for feedback and adjusts the network parameters. However, our reward function is different from [5] since we need to consider communication time between tasks. In addition, when the graph neural network is trained, we also consider the processor network information, which is ignored in [5]. A better scheduling strategy is generated in the process of multiple trainings. More details will be given in Section 4.

Our contributions are as follows:

- (1) We study the DAG scheduling problem considering both the computation and communication time. The node information is encoded through a graph neural network. Combining with reinforcement learning, we train the network to reduce the communication overhead caused by task scheduling
- (2) The scheduling scheme based on graph convolutional neural network is evaluated with the conventional DAG scheduling methods [8, 9] and the sequence to sequence scheduling method [10]. The results show that the DAG scheduling length is reduced by 15% to 25%
- (3) Compared with the state-of-the-art graph neural network models (GAT [11], GraphSAGE [12]), the evaluation shows that the DAG scheduling length of the scheme based on graph convolutional neural network and proximal policy optimization is reduced by 8% to 15%

## 2. Related Work

VEC can be applied in many fields. Hong et al. [13] proposed mobile fog, which helps the police search for and track

target vehicles by using traffic cameras. Wan et al. [14] migrated video processing tasks to computation units deployed by the edge to analyze real-time traffic videos accurately. It can reduce the latency of video analysis and improve the quality of video analysis. Grassi et al. [15] designed ParkMaster, a scheme for detecting open parking spaces based on edge computing. ParkMaster can analyze street videos uploaded to cloud servers to evaluate parking spaces.

There are some research works on how to offload the tasks. Guo et al. [16] modeled the computation offloading problem as a mixed integer nonlinear programming problem. Since the problem is NP-hard, they proposed a suboptimal solution which makes use of particle swarm optimization (PSO) and genetic algorithm (GA). Zhu et al. [17] proposed two approximated algorithms to solve the problem where multiple mobile devices share multiple heterogeneous mobile edge computing servers. Their goal is to minimize energy consumption. Fang et al. [18] designed an approximated offline algorithm to minimize the total response time for finishing all the tasks in edge computing. Zhu et al. [4] proposed a novel scheme named Fog Following Me (Folo), which considers the mobility of vehicles. These vehicles may generate tasks or serve as fog nodes. The privacy and security issues are also considered in vehicular-related networks [19–22].

For the scheduling problem of DAG, conventional solutions define the properties of CP (critical path) [23], bottom-level (BL) [24], and top-level (TL) [25] of DAG. These scheduling algorithms are only based on the properties of one aspect of DAG. They do not consider the global structure information of the graph, so they might not obtain an efficient scheduling strategy. Taking CP as an example, CP is the critical path of DAG, that is, the longest path of the DAG. The goal of scheduling is to reduce the critical path as much as possible. The commonly used algorithms include LC (Linear Clustering) algorithm [8] which repeatedly clusters the critical path directly and CPOP (Critical-Path-on-a-Processor) algorithm [9] which calculates node scheduling priority through the sum of TL and BL values. An illustrating example for scheduling a DAG based on four representative conventional algorithms is shown in Section 5.1.

With the development of neural networks, for the scheduling problem of DAG, more and more researchers often use the model of recurrent neural network (RNN) [26] to read the input information about operations and their dependencies to generate scheduling strategies. Moreover, they use reinforcement learning (RL) to continuously optimize training to generate better strategies [10, 27, 28]. Although the sequence to sequence recurrent neural network model benefits natural language processing, it only serializes the input information. Therefore, in order to extract the information from the graph, researchers proposed graph neural networks [29]. Common graph neural networks include graph convolutional neural network (GCN) [30], graph attention network (GAT) [11], and GraphSAGE [12].

Although GAT introduces the attention mechanism and GraphSAGE generalizes the node information, they cannot accurately extract the key information that affects the

scheduling performance, so the scheduling performance by these neural network methods might not be good enough. In our paper, the purpose of the TLGC method using graph convolutional neural network is to consider both the information such as CP and the global DAG structure to generate a more efficient scheduling strategy (cf. Figure 1).

### 3. System Model

For the scheduling problem of the DAG, the definition of each symbol is shown in Table 1.

Graph  $G = (V, E, w(V), w(E))$  is a node-weighted and edge-weighted DAG, where  $V = \{v_1, v_2, \dots, v_n\}$  represents the set of nodes (tasks), and  $E = \{e_1, e_2, \dots, e_m\} \subset V \times V$  represents the set of directed edges. A directed edge  $e_i = (v_i, v_j)$  means that the task  $v_j$  cannot be executed until task  $v_i$  has been finished.  $w(v_i)$  represents the computation time of task  $v_i$ , and  $w(e_i)$  represents the communication time between the two tasks associated with edge  $e_i$ . Figure 2 gives an illustrating DAG. The value below a node means its computation time, and the value close to a directed edge means the communication time of the two tasks if they are scheduled on different processors.

Let  $P = \{p_1, p_2, \dots, p_k\}$  be the processor group. In order to simplify the system model, it is assumed that all processors are homogeneous and are fully connected. It means that the running time of the same task on different processors is the same, and the communication bandwidth between processors is the same.

The objective of the DAG scheduling problem is to minimize the scheduling length (makespan), which is the maximum finish time of all tasks. Note that the waiting time is counted in each task's finish time calculation. We also assume the tasks are nonpreemptive in the sense that once a task is scheduled to a processor, it cannot be terminated until it finishes its computation on that processor.

### 4. The TLGC Scheduling Scheme

In this paper, by considering both the computation and communication time of tasks, the initial scheduling strategy is generated through a graph neural network. The neural network is trained by reinforcement learning (RL). In the training, by observing the results of the generated scheduling strategy, a corresponding reward will be provided for the network. The reward function is set according to an evaluation mechanism, such as to minimize the DAG scheduling length. The RL algorithm utilizes this reward signal to gradually improve the scheduling scheme.

The design of the TLGC scheme faces the following challenges:

- (1) In DAG scheduling, as mentioned before, the communication delay between tasks cannot be ignored which makes the training much more difficult
- (2) The purpose of the reinforcement learning is to maximize the cumulative rewards. The reward value

directly influences the DAG scheduling length, and it also determines whether the model will converge or not. Therefore, the design of the reward function should comprehensively consider the factors such as communication time of tasks and the degree of parallelism (whether or not to use all the available processors)

We now discuss how to tackle the above challenges in the subsequent subsections.

**4.1. Information Embedding.** In each state observation, the state information (the states of the DAG and the processors) must be transformed into feature vectors to be transmitted to the policy network. One option is to create a planar feature vector containing all state information. However, this method cannot scale to arbitrary size and topology of DAGs. In addition, processing high-dimensional feature vectors will require a huge size policy network. It will be difficult to train.

Therefore, the scalability can be achieved by using a graph neural network, which encodes or “embeds” state information (e.g., the running time of tasks, the dependency structure between nodes, the communication time between tasks, and the state of processing units) into a set of embedding vectors. The method adopted in this paper is based on a graph convolutional neural network [31] but customized for scheduling. The notations used in this paper and their descriptions are shown in Table 1.

The graph embedding takes the DAG as the input whose nodes have a set of stage attributes (such as task computation time) and outputs two different types of embeddings:

- (1) Node embeddings capture information about nodes and their child nodes (for example, including the aggregated information along the critical path from the node)
- (2) DAG embeddings can summarize the information in the DAG and the processor's information during execution

It is important that the information embedded and stored is not hardcoded. It will automatically learn the statistically significant content and how to calculate the information from the input DAG through end-to-end training. In other words, embedding can be regarded as a feature vector, and a graph neural network can learn and calculate without manual feature engineering.

Given the feature vector  $x_v$  of node  $v$  in a DAG, the embedding  $(G, x_v) \rightarrow e_v$  of each node is established where  $e_v$  is a vector containing the information of all nodes ( $v$ 's child nodes and their descendants) reaching node  $v$ . In order to calculate these vectors, starting from the leaves of the DAG, the information propagates from the child nodes to the parent nodes according to a series of information passing steps (Figure 3). In each information passing step, the embedding of a node  $v$  (the shadow nodes in Figure 3) whose child nodes have aggregated information from its

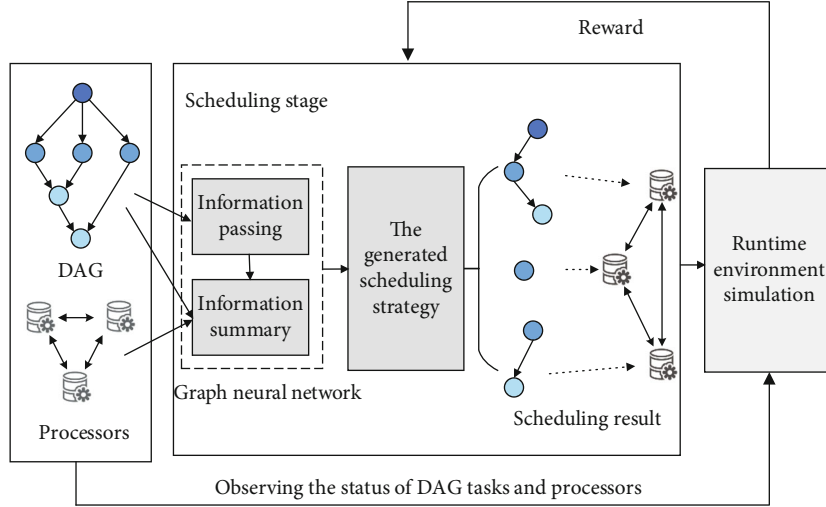


FIGURE 1: In the DAG scheduling process, the network will generate the task (node) to be scheduled in each scheduling event.

TABLE 1: Notation table.

Notations	Descriptions
$v$	A node in the DAG
$w_v$	Computation time of $v$
$\xi(v)$	Child node set of $v$
$\zeta(v)$	Parent node set of $v$
$x_v$	Feature vector of $v$
$e_v$	Embedding information of $v$
$y$	Embedding information of the DAG
$c_{u,v}$	Communication time from $u$ to $v$
$\tau_v$	Finish time of $v$
$\phi_t$	The ready node set in step $t$
$f, g, q$	Nonlinear functions realized by neural networks
sinks	The nodes without outgoing edges
sources	The nodes without ingoing edges

whole child nodes is calculated as

$$e_v = g \left[ \sum_{u \in \xi(v)} f(e_u, c_{v,u}) \right] + x_v, \quad (1)$$

where  $f(\cdot)$  and  $g(\cdot)$  are nonlinear transformations on input vectors, which are realized by graph neural network, and  $\xi(v)$  represents child node set of  $v$ . The first is the general nonlinear aggregation operation, which summarizes the embedding of  $v$ 's child nodes and the communication overhead to  $v$ 's child nodes. Adding the summary item from this aggregation to the feature vector  $x_v$  of  $v$  can obtain the embedding of  $v$ . The same nonlinear transformations  $f(\cdot)$  and  $g(\cdot)$  are used repeatedly in all nodes and information passing steps.

When calculating the node embedding of node  $v$  through nonlinear transformation, it is usually calculated in the form of a nonlinear transformation  $f(\cdot)$ . In our scheme, the second nonlinear transformation  $g(\cdot)$  is added. The reason is that the graph neural network cannot calculate some valuable features for scheduling without  $f(\cdot)$  [5]. For example, it cannot calculate the critical path of the DAG, which requires a series of steps operations on nodes during information passing. Note that the communication delays play an important role in calculating this kind of critical path.

We add a summary node to the DAG to calculate the embedding of the DAG. The summary node takes all nodes in the DAG as child nodes and takes the state of processing units (processors) as its feature vector to calculate the embedding of the DAG. Similarly, the embedding of summary node is also calculated by Equation (1). That is, each aggregation step has its own nonlinear transformations  $f(\cdot)$  and  $g(\cdot)$ .

**4.2. The Design of the Scheduling Network.** The TLGC scheduling scheme constructs the generation of scheduling policy into the Markov Decision Process (MDP). In each decision-making process, the scheduling policy of one node is generated. The scheduling process is illustrated in Figure 1, which is built upon [5]. However, as mentioned before, the processor network information and communication delay are considered which are ignored there.

Determining the next task (node) to be scheduled is based on the assigned score for each task. For task  $v$  in the DAG, the score of node  $v$  is  $q_v = q(e_v, y)$ , where  $q(\cdot)$  is a nonlinear function for calculating the score which is realized by the two-layer fully connected neural network. Note that, at each step  $t$ , only the ready tasks can be scheduled, i.e., the tasks satisfying all the precedence constraints. We denote this kind of ready tasks at step  $t$  as  $\phi_t$ . Then, the normalization (softmax operation) is used to calculate the probability

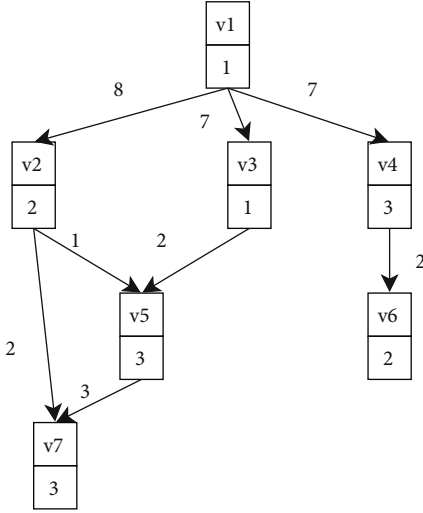


FIGURE 2: An illustrating DAG.

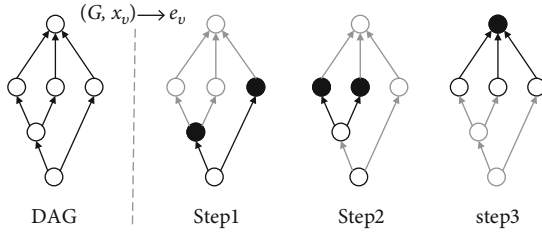


FIGURE 3: The process of information passing and aggregation in the DAG. The example shows the steps of information passing from the child nodes to the parent nodes. The shadow nodes represent calculating the information.

of selecting task  $v$  based on the priority scores:

$$P(\text{task} = v) = \frac{\exp(q_v)}{\sum_{u \in \phi_t} \exp(q_u)}. \quad (2)$$

It should be emphasized that  $\phi_t$  changes in real-time according to the execution process of the DAG and limits the output of the normalization operation.

In order to gradually improve the task selection process, we need the following reinforcement learning with the carefully designed reward function.

**4.3. The Design of Reward Function and Training Process.** We use reinforcement learning (RL) to train the neural network through many offline (simulation) experiments. In these experiments, rewards are provided by observing each decision-making process's operation. The rewards are set through the evaluation mechanism of the DAG scheduling (such as minimizing the makespan and maximizing parallelism). RL algorithm uses this reward signal to improve the scheduling strategy gradually. Therefore, the design of the reward signal is essential to the training effect of the network.

The finish time of each task can only be obtained after the task is scheduled on a certain processor. Therefore, we adopt two calculation methods for the reward function:

- (1) We take the communication delay of the tasks as the negative signal of the reward. Considering the situation that the task  $v$  is scheduled to the processor  $p_i$ , there are two cases to calculate the finish time of task  $v$ . The first case is all of  $v$ 's parent nodes (tasks) have also been scheduled on processor  $p_i$ . In this case, there is no communication cost between  $v$  and its parent tasks. Denote  $t_1^{v,i}$  as the starting time of  $v$  on processor  $p_i$  in this case; i.e., the time processor  $p_i$  has finished executing all the tasks already placed on it. The second case is some of  $v$ 's parent tasks have been scheduled to another processor. In this case, there will exist communication time. Denote  $t_2^{v,i}$  as the starting time of  $v$  on processor  $p_i$  in this case

$$t_2^{v,i} = \max_{u \in \zeta(v)} (\tau_u + c_{u,v}). \quad (3)$$

At this time, reward is calculated as follows:

$$\text{reward}_v^i = t_1^{v,i} - t_2^{v,i}. \quad (4)$$

After the DAG task is completed, the average reward of each decision is calculated according to the scheduling length of the DAG (the maximum finish time of all tasks) and the above  $\text{reward}_v^i$ . We now can adjust the neural network based on Equation (5). For this equation,  $k$  in the numerator means the number of processors. The denominator  $\max_{p \in P}(t_p)$  means the scheduling length of the DAG where  $t_p$  means the finish time of executing all tasks placed on the processor  $p$ .

$$\text{avg}_{\text{reward}} = \frac{\sum_{i=1}^k (\text{reward}_v^i)}{\max_{p \in P}(t_p)}. \quad (5)$$

Similar to [5, 32], the TLGC scheme is then trained by reinforcement learning and proximal policy optimization [33] with the strategy gradient method. The method is to learn by gradient descent of neural network parameters using the above rewards observed during training. These are commonly used methods, and we omit the details.

The above describes how to select the scheduled task. Then, we need to allocate it to some processor that satisfies its earliest start time (EST) [34]. For a task  $v$ , we need to calculate all the earliest start time  $\text{EST}_i(v)$  on each processor  $i$  and then pick the processor with the smallest  $\text{EST}_i(v)$  value. A detailed example for calculating  $\text{EST}_i(v)$  can be found in the TLGC processor selection example (cf. the last paragraphs of Section 5.2). Note that the communication delay between two tasks will become zero if they are placed on the same processor.

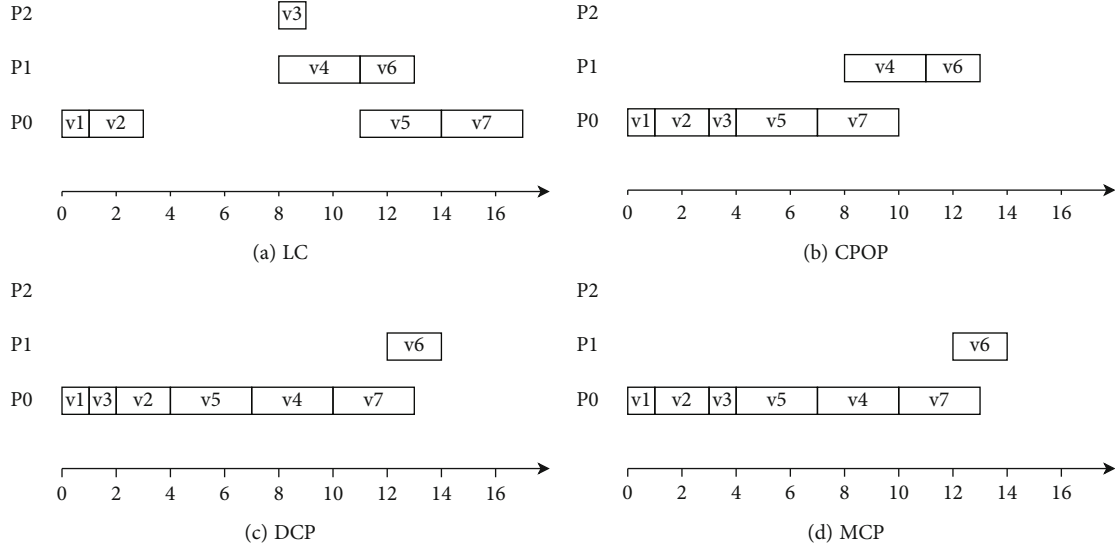


FIGURE 4: The scheduling length of the DAG in Figure 2 based on four representative conventional scheduling algorithms. The  $x$ -axis means the time and the  $y$ -axis stands for the three available processors.

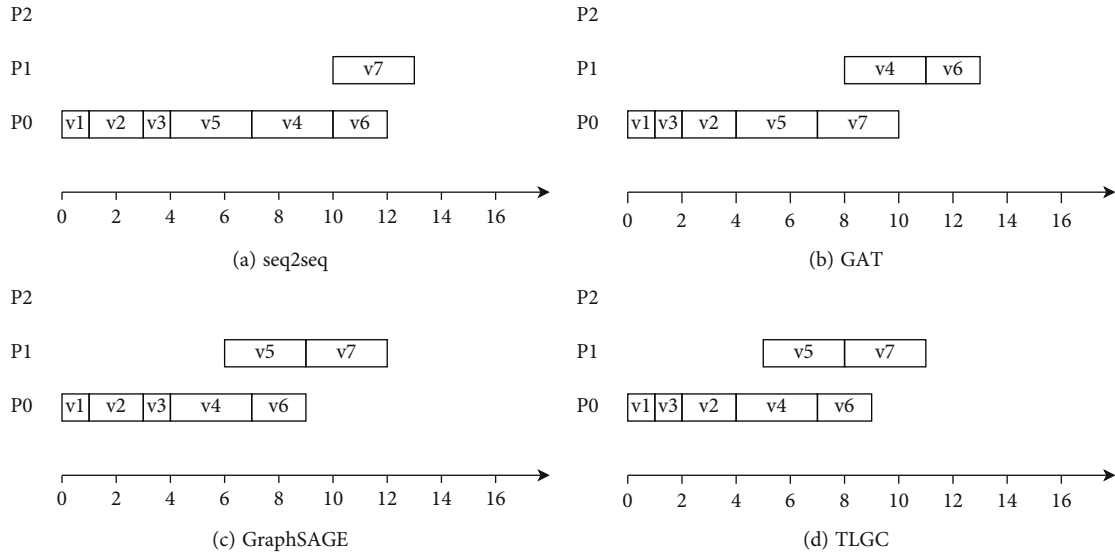


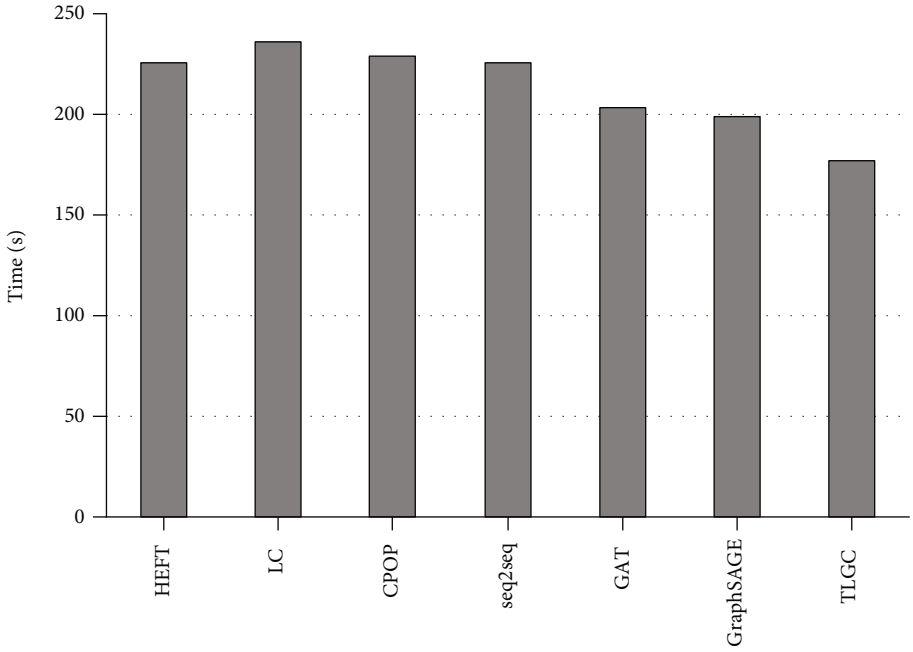
FIGURE 5: The scheduling length of the DAG in Figure 2 based on three representative neural networks and our TLGC method. The  $x$ -axis means the time and the  $y$ -axis stands for the three available processors.

TABLE 2: The earliest start time for each task on different processors.

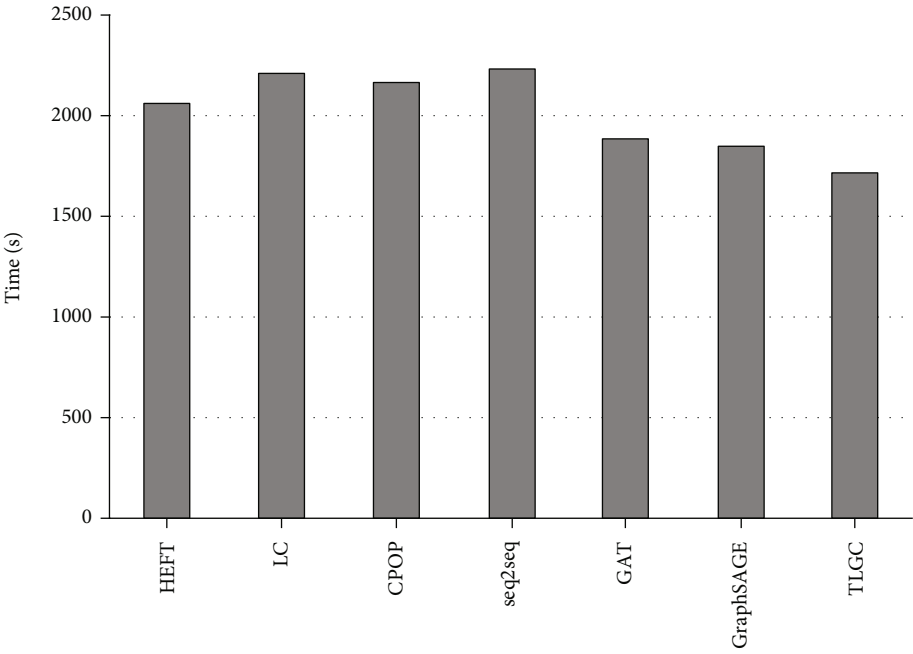
Tasks	$EST_0()$	$EST_1()$	$EST_2()$	$EST()$	Processor
$v1$	0	0	0	0	$p0$
$v3$	1	8	8	1	$p0$
$v2$	2	9	9	2	$p0$
$v4$	4	8	8	4	$p0$
$v5$	7	5	5	5	$p1$
$v6$	7	9	9	7	$p0$
$v7$	11	8	11	8	$p1$

TABLE 3: The attributes of the six real workflow DAGs.

Application	$ V $	$ E $	CCR
Cycles	1322	1940	14.84
1000Genome	902	1166	12.93
Epigenomics	863	1068	12.38
Montage	472	12840	2.72
Seismology	1101	1100	5.30
Soykb	546	1344	2.46



(a) Node number = 100



(b) Node number = 1000

FIGURE 6: Continued.

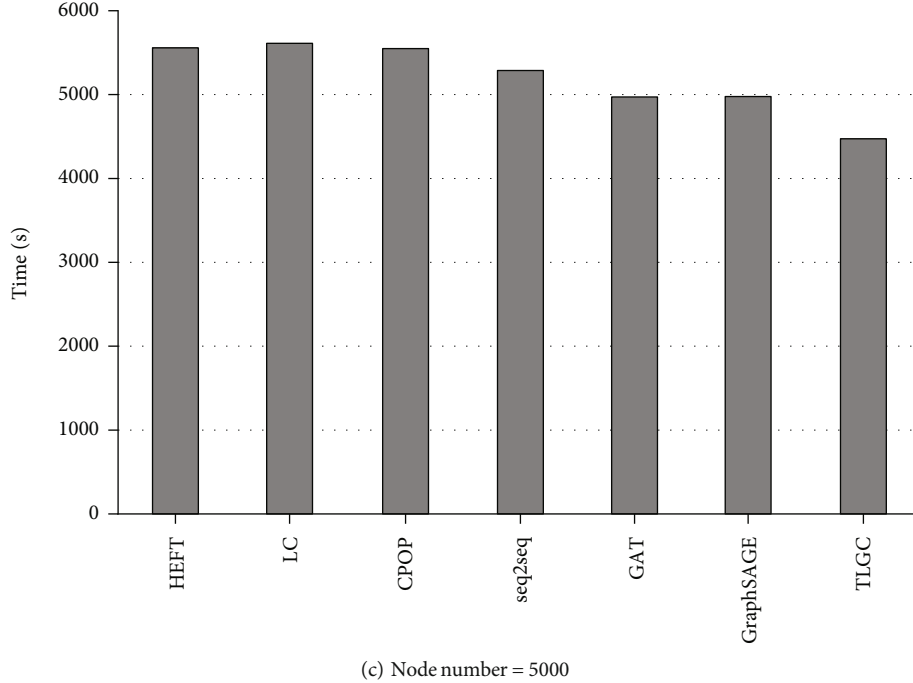


FIGURE 6: When CCR = 0.1, the scheduling lengths of the randomly generated DAGs with 100, 1000, and 5000 nodes under different scheduling methods.

## 5. An Illustrating Experiment for DAG Scheduling with Communication Delays

In order to give the readers a concrete feeling of DAG scheduling with different methods, taking Figure 2 as an example, we will show the corresponding scheduling results based on both the conventional scheduling algorithms and the neural network-based methods.

*5.1. Scheduling Results with Conventional Algorithms.* For the scheduling problem of a DAG, conventional solutions are to define the properties of CP (critical path), BL (bottom level, as defined in Equation (6)), and TL (top level, as defined in Equation (7)) of a DAG. Conventional scheduling algorithms often only consider these properties but lack the global information of DAGs. Thus, the scheduling results might not be stable.

$$blevel(v) = \begin{cases} w_v, & v \in \text{sinks}, \\ \max_{u \in \zeta(v)} (blevel(u) + w_v + c_{u,v}), & \text{otherwise}, \end{cases} \quad (6)$$

$$tlevel(v) = \begin{cases} 0, & v \in \text{sources}, \\ \max_{u \in \xi(v)} (tlevel(u) + w_u + c_{u,v}), & \text{otherwise}. \end{cases} \quad (7)$$

The critical path of a DAG is the longest path in the DAG. The goal of scheduling is to reduce the critical path as much as possible. The commonly used algorithms include LC algorithm [8] which repeatedly clusters the critical paths, DCP (Dynamic Critical Path) algorithm [23] which orders

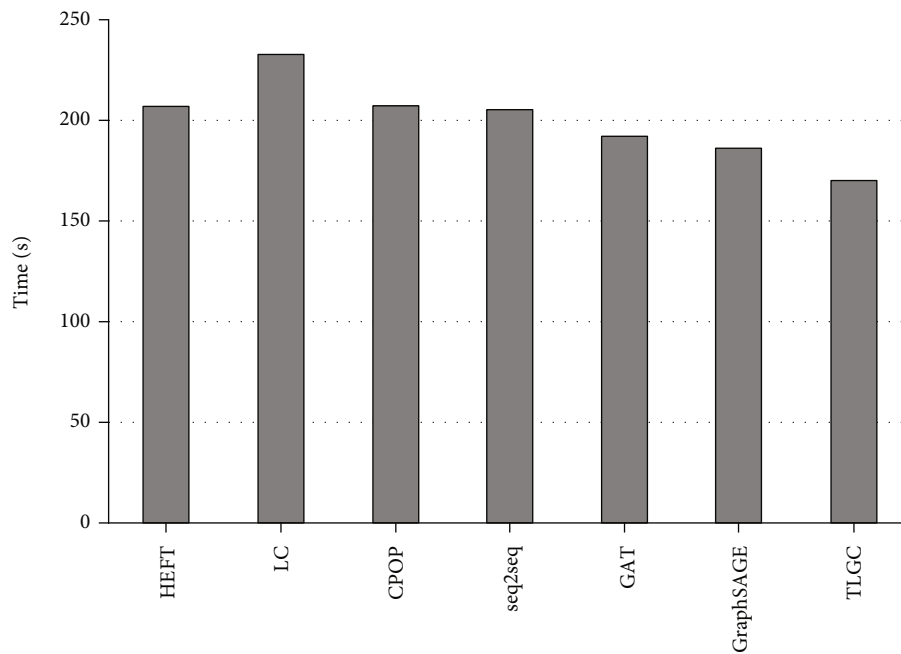
the tasks based on an increasing sum of TL and BL values, CPOP algorithm [9] which calculates node scheduling priority through a descending sum of TL and BL values, and MCP (Modified Critical Path) algorithm [35] which prioritize tasks with their descending BL values. Although these algorithms intuitively shorten the length of the critical path, the scheduling effects are often not satisfactory.

Considering the DAG in Figure 2, the number of available processors is 3, and the length of the critical path of the DAG before the scheduling is 21. Figure 4(a) shows the scheduling result of the LC algorithm, which maps the DAG into 3 clusters where each processor hosts one cluster of tasks. Note that there is a total order of the tasks in each linear cluster. The scheduling length of the LC algorithm is 17. Figures 4(c) and 4(d) show the scheduling results by employing the DCP and MCP algorithms, respectively. Their scheduling lengths are both 14 which is smaller than the one by the LC algorithm. After performing the CPOP scheduling algorithm, the scheduling length becomes 13, which is shown in Figure 4(b).

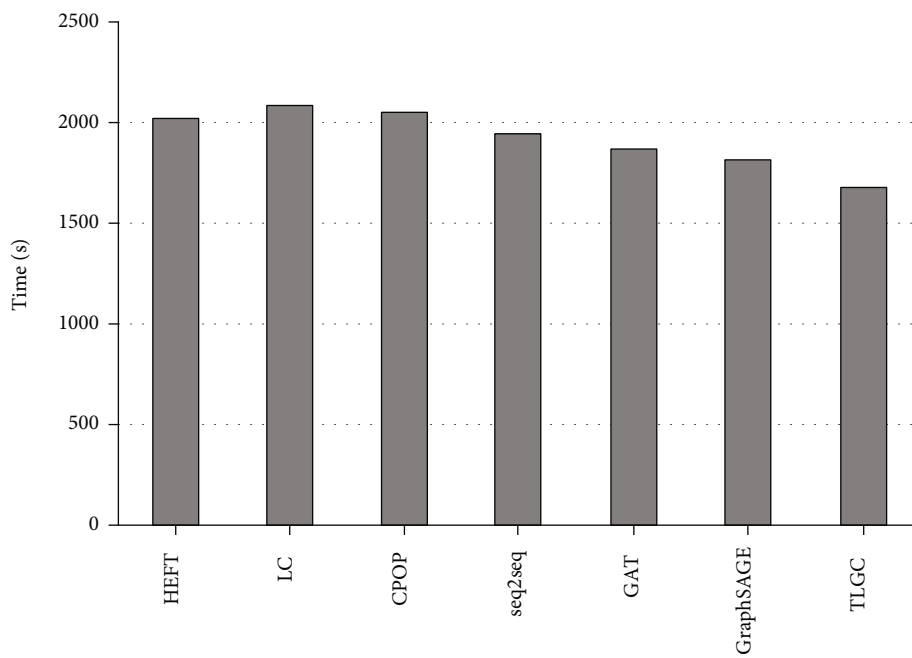
As shown in the scheduling results of utilizing the four representative scheduling algorithms, we can see the key to these conventional scheduling algorithms is to reduce the critical path's length. However, this process ignores the communication time of noncritical path nodes, which also plays an important factor in affecting the DAG's scheduling length.

*5.2. Scheduling Results with Neural Networks.* In order to cope with the above issue and to get a smaller scheduling length, more and more researchers often use the model of recurrent neural network (RNN) [26] to read the input information about operations and their dependencies to





(a) Node number = 100



(b) Node number = 1000

FIGURE 7: Continued.

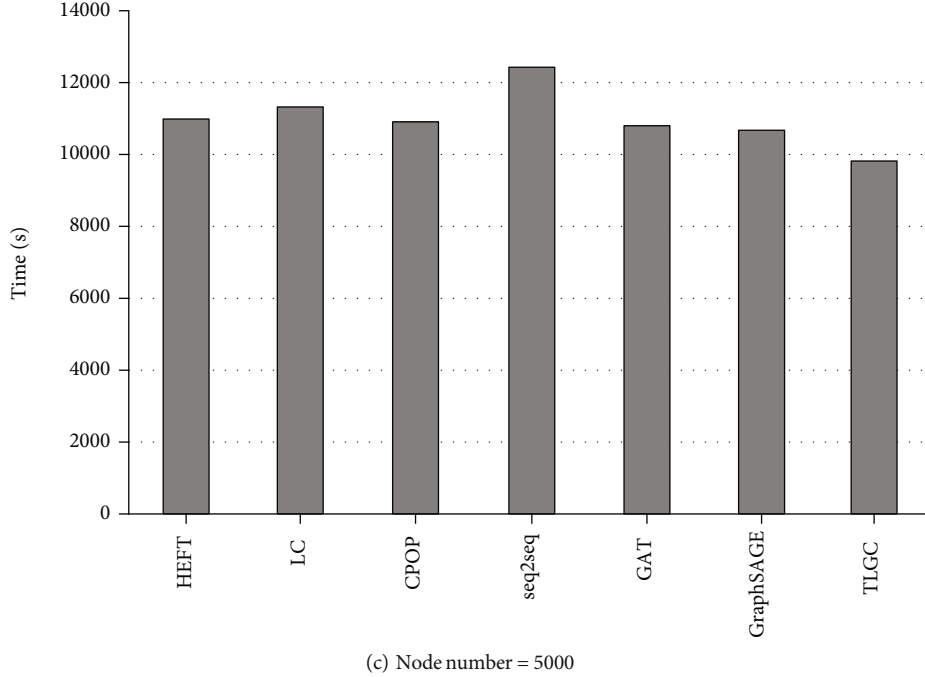


FIGURE 7: When CCR = 1.0, the scheduling lengths of the randomly generated DAGs with 100, 1000, and 5000 nodes under different scheduling methods.

generate scheduling strategies. Moreover, they use reinforcement learning (RL) to continuously optimize training to generate better strategies [10, 27, 28]. For example, under the scheduling model of seq2seq [28], the scheduling length of DAG in Figure 2 is 13, which is shown in Figure 5(a).

As mentioned before, although the sequence to sequence recurrent neural network model benefits natural language processing, it only serializes the input information. Therefore, in order to extract the information from the graph, researchers proposed graph neural networks [29]. Common graph neural networks include graph attention network (GAT) [11] and GraphSAGE [12]. Figure 5(b) shows the scheduling result based on the GAT model. Its scheduling length is 13 since it does not thoroughly learn the impact of communication time on the scheduling length. Figure 5(c) shows the scheduling result based on the GraphSAGE model, and its scheduling length is 12 because it does not thoroughly learn the impact of previously scheduled tasks on subsequent tasks. Figure 5(d) shows the scheduling result of our TLGC scheme, and the length is 11, which is the smallest among all the scheduling methods.

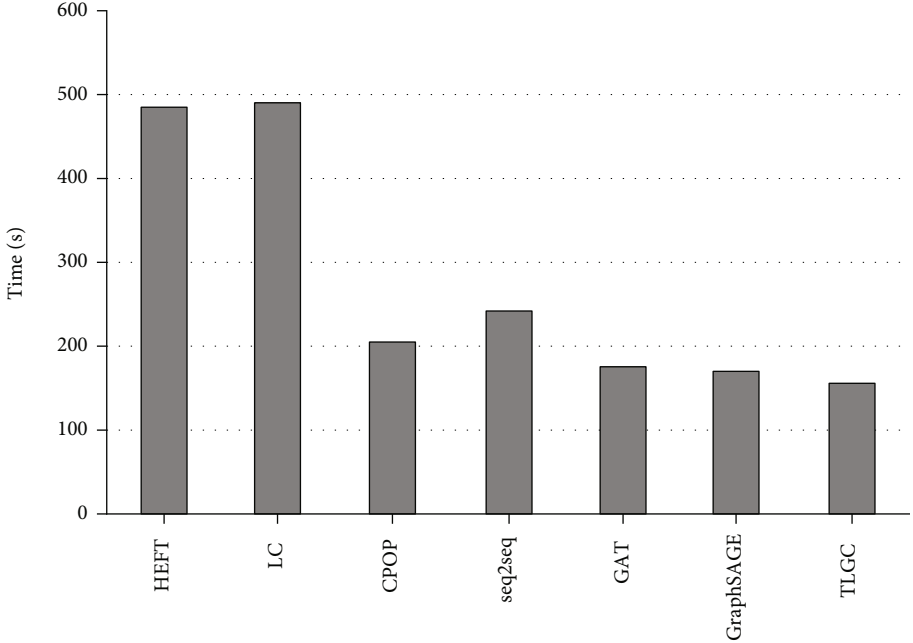
The output of our TLGC scheme gives the scheduling order of tasks in Figure 2 as  $v_1$ ,  $v_3$ ,  $v_2$ ,  $v_4$ ,  $v_5$ ,  $v_6$ , and  $v_7$ . The processors that can handle tasks are  $p_0$ ,  $p_1$ , and  $p_2$ . For a task  $v$ , its earliest start time determines the processor where it will be executed. For  $v_1$  in Figure 2, its earliest start time  $EST(v_1)$  is 0 for each processor. Without loss of generality, we schedule task  $v_1$  to processor  $p_0$ .  $v_3$  is also scheduled on processor  $p_0$  ( $EST_0(v_3) = 1$ ); otherwise, the communication with  $v_1$  will increase the earliest start time of  $v_3$  ( $EST_1(v_3) = EST_2(v_3) = 1 + 7 = 8$ ). Thus, the earliest start time of  $v_3$  is 1.

If  $v_2$  is scheduled on processor  $p_1$  or  $p_2$ , its earliest start time  $EST_1(v_2) = EST_2(v_2) = 1 + 8 = 9$ . If  $v_2$  is scheduled on processor  $p_0$ , it can start after  $v_3$  is completed and  $EST_0(v_2) = EST(v_3) + 1 = 2$ . Thus, its earliest start time  $EST(v_2)$  is 2 on processor  $p_0$ . The case for  $v_4$  is similar with  $v_2$ . Its earliest start time on processors  $p_0$ ,  $p_1$ , and  $p_2$  are 4, 8, and 8, respectively. Thus,  $v_4$  is scheduled on processor  $p_0$  and  $EST(v_4)$  is 4.

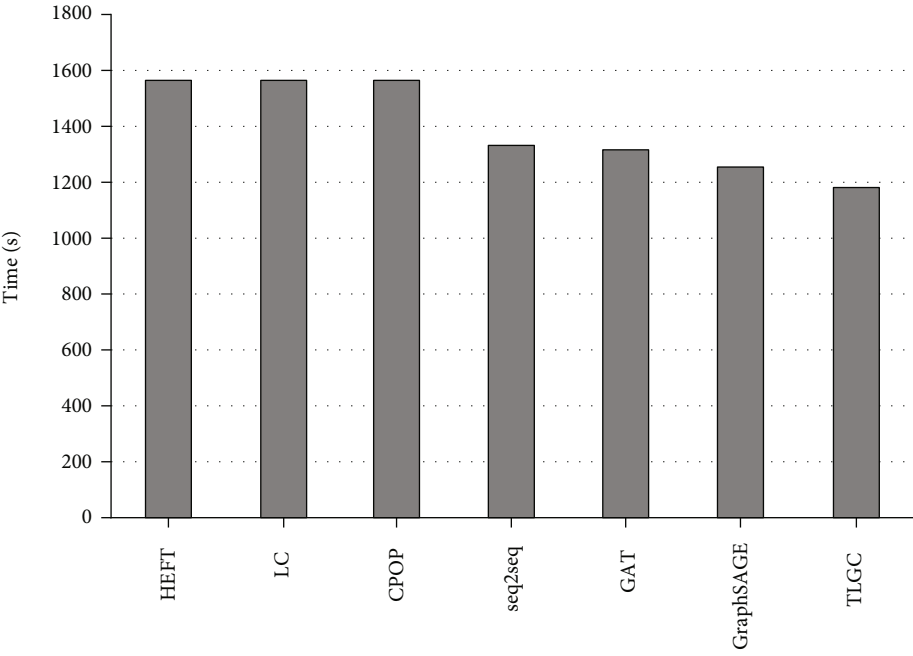
For  $v_5$ , it depends on  $v_2$  and  $v_3$ . If  $v_5$  is scheduled on processor  $p_0$ , the communication time can be saved and  $EST_0(v_5) = 7(EST(v_4)) + 3 = 7$  since it needs to wait for  $v_4$  to be completed. If  $v_5$  is scheduled on process  $p_1$ ,  $EST_1(v_5)$  is the maximum between  $(EST(v_2) + 2 + 1 = 5)$  and  $(EST(v_3) + 1 + 2 = 4)$ . Thus,  $v_5$  is scheduled on processor  $p_1$  and  $EST(v_5)$  is 5. For  $v_6$ , its earliest start time on  $p_0$ ,  $p_1$ , and  $p_2$  is 7, 9, and 9, respectively. Thus,  $v_6$  is scheduled on processor  $p_0$ .

The parent tasks of  $v_7$  are  $v_2$  and  $v_5$ . If  $v_7$  is scheduled on processor  $p_0$ , the communication time between  $v_7$  and  $v_2$  can be avoided.  $v_7$  may start after  $v_6$  is completed which is time 9. However, since  $v_7$  also depends on  $v_5$ , task  $v_7$  can only be executed after  $v_5$  finishes its computation which is time  $5 + 3 + 3 = 11$ . Thus,  $EST_0(v_7)$  is 11. If  $v_7$  is scheduled on processor  $p_1$ , considering its parent node  $v_2$ , the time it may be executed is  $2 + 2 + 2 = 6$ . Considering its parent node  $v_5$ , the time it may be executed is  $5 + 3 = 8$ . So  $EST_1(v_7)$  is 8. If  $v_7$  is scheduled on processor  $p_2$ , its earliest start time  $EST_2(v_7)$  is 11. Thus,  $EST(v_7)$  is 8 on processor  $p_1$  and  $v_7$  is scheduled on this processor.

Table 2 shows the earliest start time for each task on different processors. The last column in Table 2 means the selected processor where the corresponding task is placed



(a) Node numbers = 100



(b) Node numbers = 1000

FIGURE 8: Continued.

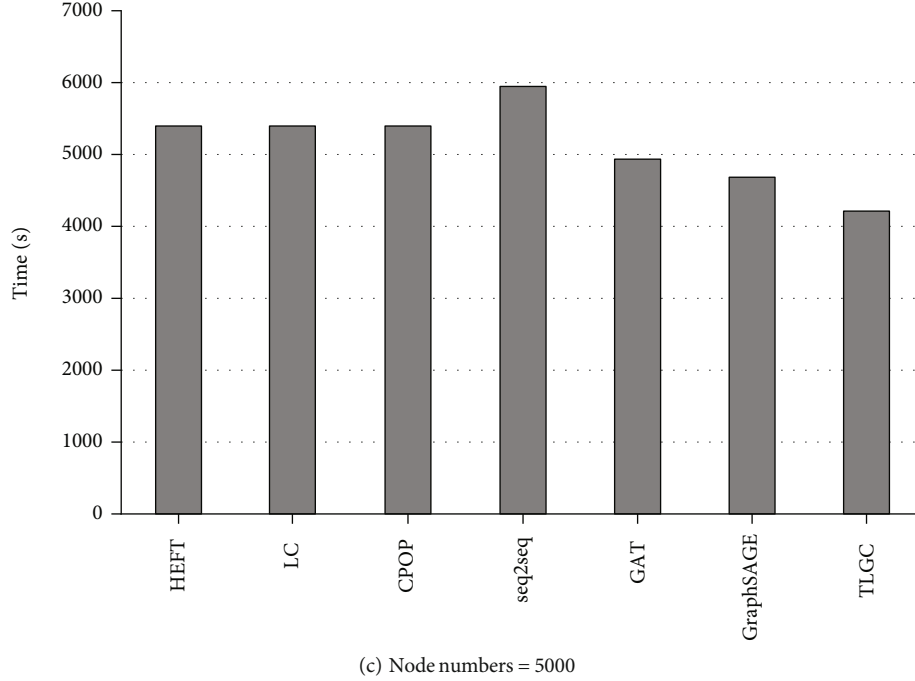


FIGURE 8: When CCR = 10.0, the scheduling lengths of the randomly generated DAGs with 100, 1000, and 5000 nodes under different scheduling methods.

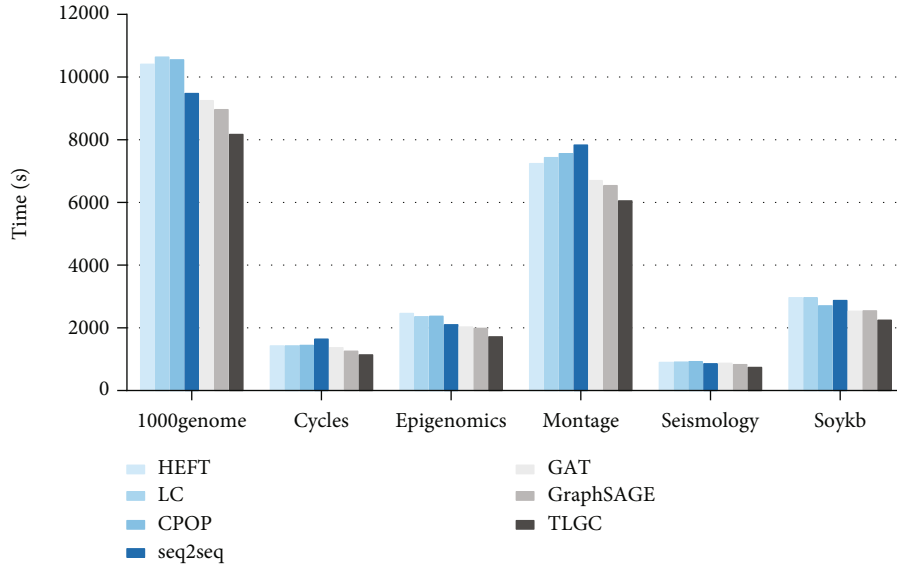


FIGURE 9: The scheduling lengths of real workflow DAGs under the strategies generated by different scheduling methods.

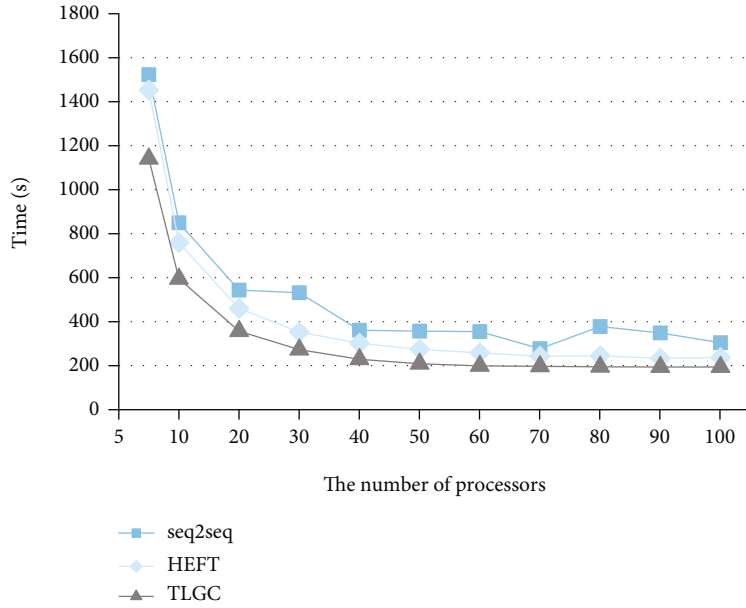
onto. The tasks are executed with the order from the top to the bottom.

## 6. Experiments

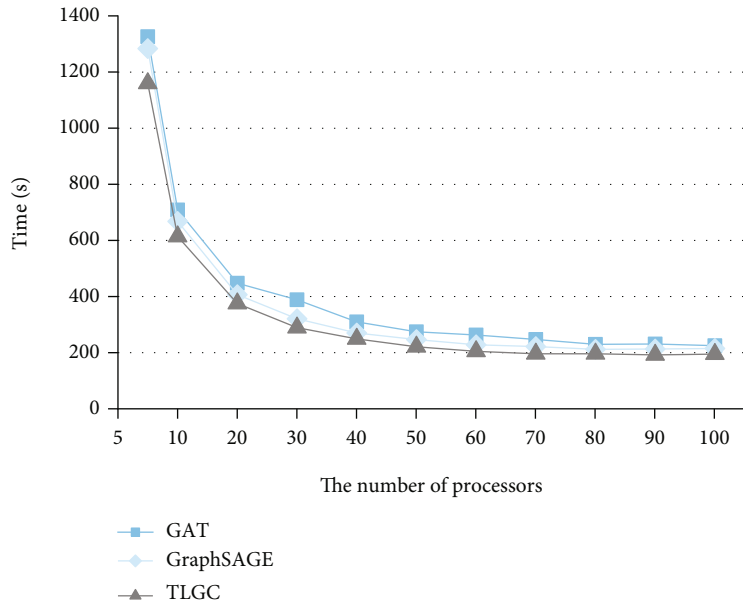
In this section, based on both randomly generated and real-world data sets, we will compare the proposed TLGC scheduling scheme with the conventional scheduling algorithms and neural network based methods to show the superiority of the TLGC scheme proposed in this paper.

**6.1. Experimental Environment.** The experiment was conducted on a Linux server with 56 Intel Xeon E5-2680v4@2.40 GHz CPUs, 256 GB memory, and 3.0 TB hard disk. The operating system of the server is Ubuntu 16.04.7 LTS. The code was implemented by Python 3.7.9 and TensorFlow 1.14.0. The server was equipped NVIDIA Tesla P100 GPU with 16 GB video memory. The versions of NVIDIA driver and CUDA are 440.118.02 and 10.2, respectively.

**6.2. Data Sets.** This subsection describes the DAG data sets used in this paper. This paper adopts two data sets: random



(a) Comparison of TLGC with conventional algorithm (HEFT) and seq2seq model



(b) Comparison of TLGC with graph neural network models (GAT, GraphSAGE)

FIGURE 10: With the increase of the number of processors, the scheduling lengths of the same workflow (cycles) DAG under different scheduling methods.

structure [36] and tasks generated from parallelized applications.

Reference [36] generated DAGs with a random structure. However, the data set does not consider the communication cost. We add the communication overhead between tasks in the DAG without changing its topology. The communication overhead is proportional to the amount of data transmitted. Reference [37] investigates the weight of the edge is affected by the computation time of its two end nodes. As a result, the communication overhead is generated as follows:

- (1) A random value  $R$  is generated by uniform distribution, normal distribution, or gamma distribution as the randomization parameter of communication overhead
- (2) The weights  $(w_s, w_t)$  of the nodes connected with the corresponding edge (the source node  $w_s$  and the destination node  $w_t$ ) are added, and the sum is square-rooted to weaken the influence of node weights
- (3) Multiply the random value  $R$  by the result obtained in step (2), i.e.,  $R\sqrt{w_s + w_t}$

TABLE 4: Scheduling generation time vs. scheduling length.

Model/ algorithm	Scheduling generation (s)	Scheduling length (s)	Proportion (%)
HEFT	0.614	2119.050	0.0287
LC	0.592	2192.470	0.0268
CPOP	0.620	2126.570	0.0292
seq2seq	15.419	2256.650	0.683
GAT	19.812	1866.450	1.062
GraphSAGE	20.089	1817.940	1.105
TLGC	20.303	1676.870	1.211

- (4) According to the requirement of CCR (Communication to Computation Ratios), the weights of edges in the DAG can be scaled

CCR represents the proportion of communication time and computation time in the DAG. In this paper, the values of CCR are set as 0.1, 1.0, and 10.0 to generate three randomized data sets.

For the real data traces, this paper uses the following six DAGs (<https://github.com/workflowhub/pegasus-traces>) to train and evaluate the graph neural network. The number of edges is usually far less than the square of the number of nodes in practical DAGs, so it is often a sparse graph. The attribute information of the six DAGs is shown in Table 3.

**6.3. Results and Analysis.** The scheduling network is trained by using the above data sets. Three graph neural networks models (GAT, GraphSAGE, and TLGC) and reinforcement learning method for proximal policy optimization [33] are implemented with the data sets to obtain their scheduling lengths. The results are compared with the commonly used conventional algorithms and sequence to sequence model [10]. The conventional algorithms include the Heterogeneous Earliest Finish Time algorithm (HEFT [9]), Linear Clustering algorithm (LC [8]), and Critical-Path-on-a-Processor algorithm (CPop [9]). Note that similar to the MCP algorithm [35] mentioned before, HEFT prioritizes the tasks based on their descending BL values, but HEFT breaks ties randomly while MCP algorithm breaks ties with BL values of descendants.

**6.3.1. Results on Randomly Generated DAGs.** For the randomly generated DAGs, we evaluate the data sets with 100, 1000, and 5000 nodes when CCR = 0.1 (as shown in Figure 6), 1.0 (as shown in Figure 7), and 10.0 (as shown in Figure 8). The experimental results show that the scheduling strategy generated by the TLGC scheduling scheme has stable performance and sufficient superiority compared with conventional scheduling algorithms, the sequence to sequence scheduling model, and other graph neural network models (GAT, GraphSAGE).

For data sets with different CCR and node number values, the TLGC scheduling scheme can always find a good scheduling strategy, while the performance of conventional scheduling algorithms is unstable in the sense that the performance of the

same algorithm may differ greatly for DAGs with different attributes. As shown in Figure 8(a), HEFT and LC have poor performance in the data set. The scheduling length of our TLGC scheme is almost one-third of the ones by HEFT and LC. In addition, the scheduling length of TLGC is still shorter than that of CPOP, which is the best conventional scheduling algorithm on this data set. Although HEFT did not perform well on this data set, it achieves the shortest scheduling lengths among the conventional scheduling algorithms on the data sets in Figures 7(a) and 7(b). However, even for these two data sets, the scheduling strategy generated by the TLGC scheme is still significantly improved compared with HEFT on the same data sets. The reason is that the conventional algorithms are based on greedy or heuristic ideas and only consider the characteristics of one aspect of the DAG (such as the critical path), so they cannot devise an efficient scheduling based on the global information. In contrast, the TLGC scheme can learn the information of graph topology so it can schedule tasks well.

We note that for some DAGs, the scheduling strategy generated by the TLGC scheme has no remarkable improvement compared with the conventional algorithms (cf. Figure 7(c)). The result can be expected. For the DAG of a specific structure, a strategy may “happen” to find the optimal or suboptimal scheduling, so there is not much room for further reducing the scheduling length.

The scheduling based on sequence to sequence (seq2seq) neural network is also unstable. For example, the scheduling lengths of the seq2seq method are even higher than the ones by conventional scheduling algorithms (cf. Figures 7(c) and 8(c)). However, for the data sets in Figures 8(a) and 8(b), the scheduling lengths of the seq2seq method are much shorter than the ones by conventional scheduling algorithms. The reason is that the sequence to sequence neural network serializes the DAG and ignores graph structure information. Therefore, the performance fluctuates wildly, and the scheduling strategy has a certain contingency.

At the same time, it can be seen that the scheduling strategy based on TLGC reduces the scheduling lengths of the DAGs by 8%-15% compared with the strategies generated by other graph neural network models. In addition, it reduces the scheduling lengths of DAGs by 15%-25% compared with the conventional scheduling algorithms and the scheduling strategy based on the seq2seq method.

**6.3.2. Results on the Real Workflow DAGs.** The scheduling results for the six real DAG workflows by various scheduling methods are shown in Figure 9. It can be found that the TLGC scheduling scheme still outperforms the conventional scheduling algorithms and all the other neural network-based methods. Specifically, compared with the conventional scheduling algorithms and the scheduling strategy based on the sequence to sequence model, the TLGC scheme can reduce the scheduling lengths by around 20%. Compared with the scheduling strategies based on GAT or GraphSAGE models, the TLGC scheme can reduce the scheduling lengths by around 10%.

We also evaluate the scheduling lengths of the same workflow (cycles) DAG under different scheduling methods when increasing the number of processors. Figure 10(a) compares HEFT (the frequently used conventional

scheduling algorithm), sequence to sequence scheduling, and the TLGC scheduling model. Figure 10(b) compares the TLGC scheduling model with the other two neural network models (GAT and GraphSAGE). It can be seen from the figure that, when the number of available processors is not large, doubling the number of processors (from 5 to 10 and 20), the scheduling lengths decrease sharply. Then, the curve tends to be flat, and the scheduling lengths decrease slowly. When the number of processors reaches 80, the scheduling lengths tend not to change with the number of processors. This means that the number of processors is no longer the bottleneck of the task scheduling strategy, and increasing the number of processors cannot reduce the scheduling lengths.

For the scheduling performance of the sequence to sequence model, we can see it does not always decrease steadily with increasing the number of processors. For example, when the number of processors increases from 70 to 80, the scheduling length does not decrease but increases, indicating that the sequence to sequence model does not thoroughly learn the structured information of the DAG and its scheduling performance is unstable.

In addition, as shown in Figure 10, with the increase of processor numbers, the TLGC scheduling scheme always achieves the shortest scheduling lengths. Compared with HEFT and the sequence to sequence scheduling model, the scheduling lengths are reduced by around 20% and this proportion tends to be stable with the increased processor numbers. Compared with the other two graph neural network (GAT, GraphSAGE) scheduling models, the scheduling lengths are reduced around 10% and are stable at 10% when increasing the number of available processors.

**6.4. Overhead Analysis.** In this subsection, we will analyze how much the scheduling generation time will account for in the scheduling length of the DAG under various scheduling methods. For the data sets generated with  $CCR = 1.0$  and node numbers = 1000, we randomly select ten samples and calculate the average of both the running time of generating the scheduling strategy and the scheduling lengths of different scheduling methods. We then calculate the proportion of the scheduling strategy generation time in the scheduling length. The results are shown in Table 4.

The TLGC scheme involves the cost of neural network parameter training and inference. The training requires many data sets and takes much time. However, the trained model can be used for scheduling strategy generation. Thus, we only consider the inference time for generating the scheduling strategy. According to this table, we can see that the scheduling generation time of TLGC is much higher than that of the conventional scheduling algorithms. However, for a relatively large DAG, the scheduling strategy generation overhead accounts for about 1% of the scheduling time, which can be ignored.

## 7. Conclusion

This paper studies the DAG scheduling problem which is aimed at scheduling all the tasks offloaded from vehicles to

the servers on the roadside units. The objective is to minimize the scheduling length, i.e., the maximum finish time of all tasks. We propose the TLGC scheduling scheme which adopts the reinforcement learning scheduling based on graph convolutional neural network. Different from previous works [3, 5], the TLGC considers the communication delay between tasks which makes minimizing the scheduling length more challenging. Compared with the representative conventional scheduling methods (HEFT [9], LC [8], and CPOP [9]) and the scheduling model based on seq2seq [10], the scheduling length of TLGC is reduced by 15% to 25%, and the scheduling performance remains stable with the increase of the number of processors. Compared with the other graph neural network models (GAT [11], GraphSAGE [12]), the scheduling length of TLGC is reduced by 8% to 15%.

## Data Availability

The experiment data sets are from previously reported studies cited in the paper.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant numbers 61832006 and 61972447).

## References

- [1] A. Boukerche and R. E. De Grande, "Vehicular cloud computing: architectures, applications, and mobility," *Computer Networks*, vol. 135, pp. 171–189, 2018.
- [2] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2019.
- [3] Y. Liu, S. Wang, Q. Zhao et al., "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.
- [4] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeeski, "Fog following me: latency and quality balanced task allocation in vehicular fog computing," in *The 15th Annual IEEE international conference on sensing, communication, and networking*, pp. 1–9, Hong Kong, China, 2018.
- [5] H. Mao, M. Schwarzkopf, S. B. Venkatakrisnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 270–288, Beijing, China, 2019.
- [6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, 2018.
- [7] R. Giroudeau, J. C. Konig, F. K. Moulai, and J. Palaysi, "Complexity and approximation for precedence constrained scheduling problems with large communication delays," *Theoretical Computer Science*, vol. 401, no. 1-3, pp. 107–119, 2008.

- [8] S. J. Kim, *A General Approach to Multiprocessor Scheduling*, The University of Texas at Austin, 1988.
- [9] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.
- [10] Y. Gao, L. Chen, and B. Li, "Spotlight: optimizing device placement for training deep neural networks," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 1662–1670, Stockholm, Sweden, 2018.
- [11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *The 6th International Conference on Learning Representations*, pp. 1–12, Vancouver, BC, Canada, 2018.
- [12] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1024–1034, 2017.
- [13] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: a programming model for large-scale applications on the Internet of things," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pp. 15–20, Hong Kong, China, 2013.
- [14] S. Wan, S. Ding, and C. Chen, "Edge computing enabled video segmentation for real-time traffic monitoring in Internet of vehicles," *Pattern Recognition*, vol. 121, article 108146, 2022.
- [15] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: an in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pp. 1–14, San Jose / Silicon Valley, CA, USA, 2017.
- [16] F. Guo, H. Zhang, H. Ji, X. Li, and V. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.
- [17] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4854–4866, 2019.
- [18] X. Fang, Z. Cai, W. Tang et al., "Job scheduling to minimize total completion time on multiple edge servers," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2245–2255, 2020.
- [19] Z. Cai, Z. Xiong, H. Xu, P. Wang, W. Li, and Y. Pan, "Generative adversarial networks," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–38, 2021.
- [20] K. Li, G. Luo, Y. Ye, W. Li, S. Ji, and Z. Cai, "Adversarial privacy preserving graph embedding against inference attack," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6904–6915, 2021.
- [21] Y. Liang, Z. Cai, J. Yu, Q. Han, and Y. Li, "Deep learning based inference of private information using embedded sensors in smart devices," *IEEE Network Magazine*, vol. 32, no. 4, pp. 8–14, 2018.
- [22] D. Yu, Z. Zou, S. Chen et al., "Decentralized parallel SGD with privacy preservation in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 5211–5220, 2021.
- [23] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE transactions on parallel and distributed systems*, vol. 7, no. 5, pp. 506–521, 1996.
- [24] A. M. Rahmani and M. A. Vahedi, "A novel task scheduling in multiprocessor systems with genetic algorithm by using elitism stepping method," *INFOCOMP Journal of Computer Science*, vol. 7, no. 2, pp. 58–64, 2008.
- [25] F. Lonsing and A. Biere, "DepQBF: a dependency-aware QBF solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 71–76, 2010.
- [26] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 2014, <https://arxiv.org/abs/1409.2329>.
- [27] Y. Gao, L. Chen, and B. Li, "Post: device placement with cross-entropy minimization and proximal policy optimization," *Advances in Neural Information Processing Systems*, vol. 31, pp. 9993–10002, 2018.
- [28] A. Mirhoseini, H. Pham, S. B. Le QV et al., "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 2430–2439, Sydney, NSW, Australia, 2017.
- [29] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.
- [30] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, <https://arxiv.org/abs/1609.02907>.
- [31] P. W. Battaglia, J. B. Hamrick, V. Bapst et al., "Relational inductive biases, deep learning, and graph networks," 2018, <https://arxiv.org/abs/1806.01261>.
- [32] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.
- [33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, <https://arxiv.org/abs/1707.06347>.
- [34] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.
- [35] M.-Y. Wu and D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, 1990.
- [36] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.
- [37] H. Wang and O. Sinnen, "List-scheduling versus cluster-scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1736–1749, 2018.