Deep Reinforcement Learning-Based Stochastic Resource-Constrained Multi-Project Dynamic Scheduling Strategy Postprint

Authors: Guo Xiaojian, Hu Fangyong

Date: 2022-05-10T11:22:57+00:00

Abstract

Currently, research on the Stochastic Duration Distributed Resource-Constrained Multi-Project Scheduling Problem (SDRCMPSP) is scarce, with most studies focusing on static scheduling schemes that cannot adjust and optimize strategies in real-time in response to environmental changes or promptly address frequently occurring dynamic factors. To this end, a Deep Reinforcement Learning (DRL) model for stochastic resource-constrained multi-project dynamic scheduling is established with the objective of minimizing total tardiness cost, a corresponding agent interaction environment is designed, and the DDDQN algorithm from reinforcement learning is adopted to solve the model. In the experiments, sensitivity analysis is first performed on the algorithm' s hyperparameters, followed by training and testing the model under two distinct conditions—variable activity durations and uncertain arrival times—using the optimal hyperparameter combination. The results indicate that the deep reinforcement learning algorithm can achieve scheduling outcomes superior to any single rule, effectively reduce the expected total tardiness cost for stochastic resource-constrained multi-projects, and provide a solid foundation for multi-project scheduling decision optimization.

Full Text

Preamble

Stochastic Resource-Constrained Multi-Project Dynamic Scheduling Strategy Based on Deep Reinforcement Learning

Guo Xiaojian, Hu Fangyong
(School of Economics & Management, Jiangxi University of Science & Technology, Ganzhou, Jiangxi 341000, China)

Abstract: Current research on the Stochastic Distributed Resource-Constrained Multi-Project Scheduling Problem (SDRCMPSP) remains limited, with most existing approaches relying on static scheduling schemes that cannot adjust strategies in real time in response to environmental changes or promptly address frequently occurring dynamic factors. To address this gap, this paper establishes a Deep Reinforcement Learning (DRL) model for stochastic resource-constrained multi-project dynamic scheduling with the objective of minimizing total tardiness cost. We design a corresponding agent interaction environment and employ the DDDQN algorithm from reinforcement learning to solve the model. The experiment first conducts sensitivity analysis on the algorithm' s hyperparameters, then trains and tests the model under two different conditions: variable activity durations and uncertain arrival times. Results demonstrate that the deep reinforcement learning algorithm yields scheduling outcomes superior to any single rule, effectively reducing the expected total tardiness cost for stochastic resource-constrained multi-project scheduling and providing a sound basis for multi-project scheduling decision optimization.

Keywords: distributed multi-project; stochastic scheduling; deep reinforcement learning; resource-constrained

0 Introduction

As society develops, project management technology has become increasingly critical, with rational resource allocation and scheduling playing a decisive role in project execution. This class of problems is commonly known as the Resource-Constrained Project Scheduling Problem (RCPSP) [1]. In practice, multiple resource-constrained projects often need to be scheduled simultaneously [2], with constraints from both local and global resources, referred to as the Distributed Resource-Constrained Multi-Project Scheduling Problem (DRCMPSP). However, in real engineering contexts, project implementation may be affected by various uncertain factors such as lack of relevant project experience, production equipment failures, resource unavailability, and weather conditions, causing deviations between actual activity durations or project arrival times and their estimates [3]. These uncertainties render pre-established multi-project plans infeasible, necessitating effective methods to reduce the Expected Total Tardiness Cost (ETTC). This class of problems is termed the Stochastic Distributed Resource-Constrained Multi-Project Scheduling Problem (SDRCMPSP).

Current literature on SDRCMPSP remains relatively scarce, with most scheduling strategies being static [4], such as priority rule algorithms [5][6]. Song et al. employed priority rule heuristics to generate baseline schedules and postponed affected activities to the earliest feasible execution times [7]. Tosselli et al. adopted a repeated negotiation game approach [8]. Liu Dongning et al. utilized a Multi-Priority Rule Heuristic (MPRH) method [9], which, while capable

of reducing multi-project delay costs in dynamic environments with variable activity durations, required multiple simulation experiments at each decision point and failed to effectively leverage past experience for timely, goal-oriented responses to dynamic environments.

Reinforcement learning algorithms, particularly those that are goal-oriented, have been widely applied to dynamic scheduling problems due to their advantage of enabling offline learning and online application [10]. Previous literature demonstrates that reinforcement learning algorithms have been successfully applied to dynamic scheduling problems in job shop operations with favorable results [11][12][13][14]. However, no existing studies have investigated the application of deep reinforcement learning algorithms to multi-project stochastic scheduling problems. Therefore, this paper applies such algorithms to the dynamic scheduling of SDRCMPSP.

This paper addresses the impact of variable activity durations and project arrival date deviations on increased Total Tardiness Cost (TTC) in multi-project operations. Through continuous simulation-based interaction between the agent and environment in deep reinforcement learning, we optimize scheduling strategies using the DDDQN algorithm from literature [14]. We first propose a mathematical model for multi-project scheduling in static environments and transform it into a dynamic scheduling process using a parallel scheduling scheme. Subsequently, we construct an interactive environment for the agent based on the multi-project dynamic scheduling process and total tardiness cost. The DDDQN algorithm enables the agent to continuously explore and exploit existing knowledge to optimize strategies for different states based on current environmental conditions, thereby reducing ETTC for stochastic resource-constrained multi-project scheduling. We conduct hyperparameter strategy combination analysis on the established model and algorithm to determine the optimal hyperparameter combination, which is then applied to simulation studies under dynamic environments with either variable activity durations or uncertain project arrival times. Simulation results demonstrate that the deep reinforcement learning method enables the agent to learn scheduling strategies superior to any single rule, providing sound decision-making support for multi-project scheduling in dynamic environments.

---

1 Problem Formulation

1.1 Multi-Project Static Problem

Consider a multi-project composed of m single projects i ($i = 1, \ldots, m$). In project i, there exist a_{ij} ($j = 1, \ldots, J_i$) real activities with duration d_{ij} and two dummy activities a_{i0} and $a_{i(J_i+1)}$ with zero duration and resource requirements. Let A_{ij} denote the set of immediate predecessors of activity a_{ij}; the start time of activity a_{ij} must be greater than the maximum completion time of activities in A_{ij}. Let L_i represent the set of local resources for project i, with R_{il} denoting the

available amount of local resource l; let G represent the set of global resources with R_g denoting its available amount. When activity a_{ij} is executed at any time, it requires r_{lij} units of local renewable resources and rg_{ij} units of global renewable resources. The arrival time of project i is ST_i, and its completion time is represented by the completion time of activity $a_{i(J_i+1)}$, denoted as $FT_{i(J_i+1)}$. When $FT_{i(J_i+1)}$ exceeds the due date, tardiness cost TC (tardiness cost) is incurred. For project i, its tardiness cost TC_i is:

$$TC_i = c_i \times \max\{FT_{i(J_i+1)} - \omega_i, 0\}$$

where c_i represents the unit tardiness cost of project i and ω_i represents the due date of project i. For multi-project scheduling problems considering global resource allocation, the objective is to minimize the Total Tardiness Cost (TTC) across all projects:

$$\min TTC = \sum_{i=1}^{m} TC_i$$

1.2 Multi-Project Dynamic Scheduling Transformation

In practice, activity durations vary due to environmental uncertainties and follow certain probability distributions, transforming the static multi-project scheduling problem into a dynamic one. Since serial scheduling' s principle of scheduling activities as early as possible is unsuitable for multi-project scheduling in dynamic environments, this paper adopts a parallel scheduling approach to achieve dynamic multi-project scheduling. In the parallel scheduling process for multi-projects, t represents the current time (time elapsed since multi-project commencement, initially 0). The relevant sets include: F (set of completed activities), D (set of activities currently being executed), P (set of candidate activities whose predecessors are all completed), and U (set of unselected activities for each project). The multi-project dynamic scheduling process proceeds as follows:

a) Input multi-project scheduling information including project scales, activity durations, precedence relationships, and local and global resource requirements; clear all activity sets.

b) Determine whether the start time ST_i of any project not yet added to set U satisfies $ST_i \leq t$; if so, add all activities of the corresponding project to U.

c) Determine whether any activity in U has all its predecessors completed; if such activities exist, add them to P and remove them from U; otherwise, proceed to step c).

d) If P is non-empty: select the highest-priority activity from P according to the scheduling rule and check for resource conflicts between the selected

activity and activities in D; if a conflict exists, return to step d); otherwise, remove the activity from P, add it to D, and continue with step c). If P is empty: proceed to step d).

e) Identify the earliest completing activity or activities in D, set t equal to their completion time, remove them from D, and add them to F.

f) Determine whether all activities across all projects are completed; if so, output the completion times $FT_{i(J_i+1)}$ $(i = 1, \ldots, m)$ for each project and calculate TTC; otherwise, return to step b).

---

2 DDDQN-Based Distributed Multi-Project Dynamic Scheduling

The multi-project dynamic scheduling problem requires selecting priority activities for execution from the current executable activity sets of each project while satisfying local and global resource constraints until all project activities are scheduled. This constitutes a sequential decision-making process that can be formulated as a Markov or semi-Markov decision problem through proper definition of states, actions, and immediate rewards.

2.1 State Description

State features should comprehensively reflect both local and global characteristics of the agent' s environment at the current decision moment. When making decisions, the agent must select appropriate actions based on current environmental features. For problems with finite state sets, representation through arrays or tables is feasible, known as Reinforcement Learning (RL). However, real-world problems often involve large or continuous state spaces, requiring deep neural networks'function approximation capabilities to eliminate the "curse of dimensionality"faced by RL algorithms. In this paper, state features for multi-project scheduling comprise three $m \times \max(J_i + 1)$ matrices, where $\max(J_i + 1)$ represents the maximum activity scale across all projects. The three matrices are: scheduling result matrix FN, activity execution matrix DN, and executable activity matrix PN.

Matrix FN consists of the completion times of activities in each project. Before network input, it undergoes max-normalization, with initial values set to zero.

Matrix DN indicates the current status of each activity in each project: 1 if being executed, 0 otherwise. Since values in this matrix are binary, normalization is unnecessary.

Matrix PN indicates whether each activity in each project is executable at the current time (i.e., all its predecessors are completed): 1 if executable, 0 otherwise. Similarly, normalization is not required.

Considering feature extraction from raw deep learning inputs, the three feature matrices are treated as three different channels of an image with height equal to the number of matrix rows and width equal to the number of matrix columns, using convolutional input for training.

2.2 Action Definition

In the multi-project dynamic scheduling DRL model, the action space consists of numerous single-rule scheduling algorithms. The reinforcement learning algorithm selects appropriate scheduling rules for different states to overcome the limitations of single rules. This paper employs 15 single scheduling rules, with the first 7 being centralized rules and the latter 8 being composite rules. Here, OFT_i represents the optimal scheduling solution for project i considering only local resource constraints, obtained using an improved grey wolf algorithm based on activity list encoding. The 15 scheduling rules are listed in Table 1, where project i denotes a project that has arrived with a non-empty candidate activity set, and j denotes a candidate activity within that project. Rules 1-6 treat candidate activity sets across projects as a unified set, simultaneously determining the activity and its project during selection. Rules 7-14 prioritize project selection first, then select activities from the chosen project' s candidate activity set.

2.3 Reward Function

Since the objective of this paper' s distributed multi-project dynamic scheduling is to minimize total tardiness cost, the immediate reward is defined as follows to accurately evaluate actions:

$$r_t = -\sum_{i=1}^{m} \left(\max\{FT_{i(J_i+1)} - \omega_i, 0\} \times c_i \right)$$

where $\max\{FT_{i(J_i+1)} - \omega_i, 0\} \times c_i$ represents the tardiness cost of project i. Let $u_t = \max\{FT_{ij} | a_{ij} \in F\}$ denote the maximum completion time of completed activities in project i at time t, with $u_0 = 0$. The algorithm' s cumulative reward is then calculated as:

$$R = \sum_{t=1}^{T} r_t = -\sum_{i=1}^{m} \left(\max\{FT_{i(J_i+1)} - ST_i - \omega_i, 0\} \times c_i \right)$$

Therefore, maximizing cumulative reward R is equivalent to minimizing u_T. Since all activity expected durations d_{ij} are constants, minimizing u_T is equivalent to minimizing the sum of products of project completion times and tardiness costs, i.e., minimizing TTC.

2.4 Exploration-Exploitation Strategy

A reasonable exploration-exploitation strategy enables the agent to fully utilize learned experience while ensuring exploration of new policy behaviors. This paper employs a linearly decreasing greedy strategy, with the agent' s action selection policy defined as:

$$a = \begin{cases} \arg\max_{a'} Q(s, a'; \theta) & \text{if rand}() < \varepsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

where rand() is a random number in $[0, 1]$ and ε follows the distribution:

$$\varepsilon = \max(\varepsilon_{\min}, 1 - \varepsilon_{\text{rate}} \times t)$$

where $\varepsilon_{\min}$ is the minimum value of ε and $\varepsilon_{\text{rate}}$ is the decay rate.

2.5 DDDQN Algorithm Flow

a) Define discount factor γ, learning rate α, experience replay buffer capacity M, network training period L, target network update period N, minimum training batch size mini_batch, and maximum training episodes $T_{\max}$. Initialize Q-network and target Q-network parameters θ and θ', input multi-project scheduling information, and set step = 0.

b) Reset all project scheduling information, clear scheduling result sets, and initialize multi-project scheduling state s_0.

c) Based on the current state, select the agent' s action (priority rule scheduling algorithm) at the current decision point according to the exploration-exploitation strategy, and execute steps b)-e) of the multi-project scheduling process.

d) Calculate the immediate reward value using Equation (6), determine the next state state_ and training termination flag done (True if terminated, False otherwise), and store the tuple $\{\text{state}, \text{action}, \text{reward}, \text{state\_}, \text{done}\}$ in the experience replay buffer.

e) Check whether network parameter update conditions are met; if so, update network parameters based on TD error; otherwise, proceed to step f).

f) Determine whether all projects are completed; if so, increment step by 1 and proceed to step g); otherwise, return to step c).

g) Check whether the maximum training episodes is reached, i.e., step = $T_{\max}$; if so, stop training and save the Q-network parameters locally; otherwise, return to step b).

Algorithm 1: Pseudocode for DDDQN-Based Multi-Project Dynamic Scheduling

1. Initialize minimum training batch mini_batch, step-size η, experience replay buffer capacity M, network training period L, target network update period N, maximum training episodes $T_{\max}$, and action selection counter num $= 0$.
2. Randomly initialize Q-network parameters θ and copy them to target Q-network θ'.
3. For step $= 1$ to M do:
4. Reset multi-project scheduling information, clear scheduling results, and initialize scheduling state s_1.
5. While not done do:
6. Based on current state s, select action a according to the exploration-exploitation policy.
7. Execute steps 2, 3, and 4 of the multi-project scheduling process until resource conflicts occur.
8. Store $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in experience replay buffer M.
9. If num$\%L == 0$ and num $>$ mini_batch then:
10. Randomly sample mini_batch tuples $(s, a, r, s', \text{done})$ from experience replay buffer.
11. Calculate TD error and update Q-network parameters.
12. If num$\%N == 0$ then update target network parameters $\theta' \leftarrow \theta$.
13. End if.
14. End while.
15. End for.

---

3 Simulation Experiments

To verify the effectiveness of the constructed simulation environment and the DDDQN algorithm for solving distributed multi-project dynamic scheduling problems, we select problem sets MP30_5 and MP90_2 from the MPSPLIB standard library for testing. Each problem set contains 5 instances with specific information detailed in Table 2. Experiments were conducted on a laptop with Windows 10 64-bit system, 24GB RAM, and AMD R7 4800H processor, running in a TensorFlow 2.0 environment.

3.1 Parameter Analysis

In reinforcement learning, hyperparameters critically impact network learning performance. Currently, hyperparameter determination generally relies on manual experience or random search. To identify the optimal hyperparameter combination, this paper uses instance MP30_{5_5} with U1 duration distribution to conduct sensitivity analysis on network structure, learning rate, target network update period N, minimum training batch size mini_batch, discount factor γ, and other hyperparameters. During sensitivity analysis of each hyperparameter, other hyperparameters remain fixed. Figure 1 shows the cumulative reward iteration curves under different hyperparameter values, where training effectiveness is judged by cumulative reward changes during training. Taking Figure 1(a) as an example, the x-axis represents training episodes and the y-axis shows cumulative reward curves. The figure demonstrates that different network structures affect algorithm performance, with the structure shown by the red line delivering optimal performance, which is thus selected as the network structure for this model while other parameters remain unchanged. Similarly, optimal values for other hyperparameters are determined, with final hyperparameter strategies summarized in Table 4.

Figure 1(a) illustrates the impact of five different convolutional layers, where each row shows filter numbers and kernel sizes with stride $(1, 1)$. The fourth network structure provides more effective performance improvement compared to the other four. Figure 1(b) shows the impact of different learning rates, revealing that low rates yield the worst training performance while high rates also degrade performance; therefore, a learning rate of 0.0001 is selected. Figure 1(c) demonstrates the effect of different target network update periods, showing that $N = 100$ delivers the best training performance. Figure 1(d) indicates that minimum training batch size has minimal impact, though performance degrades when mini_batch = 64. While mini_batch = 256 or 128 both show stable convergence, 128 is selected to reduce training time. Figure 1(e) shows the impact of different discount factors, where low values degrade performance and high values slow convergence, leading to selection of $\gamma = 0.99$.

3.2 DRL Model Solving Problem Sets

This section applies the proposed DRL model to 10 instances across two problem sets. Experiments consist of training and testing phases. In the training phase, the DRL model undergoes 5,000 simulation training episodes for each of the 10 instances under different duration distributions, using hyperparameter values from Table 4, with trained models saved locally. In the testing phase, each trained model performs 50 simulation runs under 5 duration distributions on its corresponding instance to obtain average TTC.

Table 4 summarizes the hyperparameter combinations. Figure 2 shows the training process for instance MP30_{5_5} under five duration distributions, with x-axis representing total tardiness cost and y-axis representing training episodes.

For distributions with relatively small variance (U1, B1), TTC iteration curves remain at the bottom with minimal fluctuation. For medium variance distributions (U2, B2), curves occupy the middle position with moderate fluctuation. For the high-variance Exp distribution, curves appear at the top with significant fluctuation, indicating that expected total tardiness cost increases with activity duration uncertainty.

In the testing phase for the trained MP30_{5_5} model, Figure 3 compares its performance against the 15 single scheduling rules in the action space, with each approach performing 50 simulation runs under corresponding duration distributions to obtain average TTC. The results demonstrate that the DRL algorithm overcomes the myopia of single rules and achieves better scheduling outcomes in dynamic environments. Further comparison selects the best-performing rule among the 15 across 5 duration distributions and compares it with the DRL model results using improvement rate as the evaluation metric, calculated by Equation (11). Table 5 shows the reinforcement learning improvement rates.

Table 6 compares the scheduling runtime between priority rule algorithms and the DRL model across 50 multi-project scheduling processes. The trained model can rapidly make optimal scheduling decisions based on current environmental states, with speed comparable to priority rule algorithms.

Table 7 presents average TTC values for MP30_5 and MP90_2 problem sets under different duration distributions across 50 scheduling runs. Compared with literature [9], our results are inferior for MP30_5 under U1 distribution but superior for MP90_2.

3.3 Uncertain Arrival Time

In practice, project arrival times often deviate from estimates due to local resource shortages. This section investigates the DRL model' s performance under uncertain project arrival times, where activity durations follow constant distribution while project arrival times follow the distribution types characterized in Equation (12), with features identical to the duration distributions in Section 3.2. Here, a is a random integer in $[0, 4]$; for example, $a = 0$ indicates that project i' s arrival time follows a U1 distribution.

The state space generated by uncertain multi-project arrival times is smaller than that from uncertain durations, so DRL model training is set to 5,000 episodes with hyperparameters identical to Section 3.2. Figure 4 shows the total tardiness cost during model training, where TTC continuously decreases and stabilizes as training episodes increase, indicating effective training. Since no existing literature addresses DRCMPSP with uncertain project arrival times, the trained model' s average TTC of 3022.01 from 100 simulation experiments is compared with the best single rule' s TTC of 4973.75, yielding an improvement rate of 64.6%.

4 Conclusion

This paper pioneers the application of deep reinforcement learning to multi-project scheduling problems, proposing a DRL-based distributed multi-project dynamic scheduling model to minimize total tardiness cost under stochastic durations. We constructed an interactive simulation environment for the agent and conducted simulation experiments using instances from MP30_5 and MP90_2 problem sets. The study performed sensitivity analysis on model hyperparameters and trained/tested the model on various instances. Results demonstrate that the proposed DRL model offers advantages for distributed multi-project dynamic scheduling in stochastic environments. The trained model makes decisions at speeds comparable to priority rules while effectively reducing total tardiness cost for stochastic distributed multi-project scheduling, thereby expanding the application of deep reinforcement learning to stochastic project scheduling problems.

References

[1] Lova A, Tormos P. Analysis of Scheduling Schemes and Heuristic Rules Performance in Resource-Constrained Multiproject Scheduling [J]. Annals of Operations Research, 2001, 102 (1-4): 263-286.

[2] Lee Y H, Kumara S, Chatterjee K. Multi-agent based dynamic resource scheduling for distributed multiple projects using a market mechanism [J]. Journal of Intelligent Manufacturing, 2003, 14 (5): 471-484.

[3] Davari M, Demeulemeester E. Important classes of reactions for the proactive and reactive resource-constrained project scheduling problem [J]. Annals of Operations Research, 2019, 274 (1-2): 187-210.

[4] Satic U, Jacko P, Kirkbride C. Performance evaluation of scheduling policies for the dynamic and stochastic resource-constrained multi-project scheduling problem [J]. International Journal of Production Research, 2020, 60 (4): 1411-1423.

[5] Wang Yanting, He Zhengwen, Kerkhove L P, et al. On the performance of priority rules for the stochastic resource constrained multi-project scheduling problem [J]. Computers & Industrial Engineering, 2017, 114 (DEC.): 223-234.

[6] Chen Haojie, Ding Guofu, Zhang Jian, et al. Research on priority rules for the stochastic resource constrained multi-project scheduling problem with new project arrival [J]. Computers & Industrial Engineering, 2019, 137 (2): 106060-.

[7] Song Wen, Xi Hui, Kang Donghun, et al. An Agent-based Simulation System for Multi-Project Scheduling under Uncertainty [J]. Simulation Modelling Practice and Theory, 2018, 86 (11): 187-203.

[8] Tosselli L, Bogado V, E Martínez. A repeated-negotiation game approach to distributed (re)scheduling of multiple projects using decoupled learning [J]. Simulation Modelling Practice and Theory, 2019, 98 (4): 101-112.

[9] Liu Dongning, Xu Zhe. A stochastic scheduling for distributed multi-project with multi-PR heuristic [J/OL]. Systems Engineering-theory & Practice, 2021, 41 (12): 3294-3303.

[10] Han Xincheng, Yu Shengping, Yuan Zhiming, et al. High-speed railway dynamic scheduling based on Q-learning method [J]. Control Theory & Applications, 2021, 38 (10): 1511-1521.

[11] Waschneck B, Reichstaller A, Belzner L, et al. Deep reinforcement learning for semiconductor production scheduling [C]// SEMI Advanced Semiconductor Manufacturing Conference (ASMC). 2018.

[12] Lin Chuncheng, Deng Derjiunn, Chih yenling, et al. Smart Manufacturing Scheduling with Edge Computing Using Multi-class Deep Q Network [J]. IEEE Trans on Industrial Informatics, 2019, 15 (7): 129-138.

[13] Liu Chienliang, Chang Chuanchin, Tseng C J. Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems [J]. IEEE Access, 2020, PP (99): 1-1.

Note: Figure translations are in progress. See original paper for figures.