Proceedings of the 7th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/2025.28610

Stochastic Resource Constrained Project Scheduling Problem with Uncertain Resource Availability Based on Deep Reinforcement Learning

Zhiming Yang

School of Economics and Management, Beihang University, Beijing, China
vrdssotyzm@163.com

Abstract. In complex engineering projects such as aerospace, construction, and pharmaceuticals, project scheduling needs to optimize activity allocation under resource constraints, dynamic uncertainties, and activity interruptions. This paper focuses on the core challenge of project management in the context of intelligent manufacturing - the Stochastic Resource-Constrained Project Scheduling Problem with Uncertain Resource Availability (SRCPSP-URA). An end-to-end framework based on deep reinforcement learning (DRL) and dynamic graph attention network variant (GATv2) is proposed to solve SRCPSP-URA. A Markov decision process (MDP) model is developed for DRL to determine which activity to start at a stage. GATv2 is applied to extract complex and dynamic features from scheduling problems and provide them for policy network to create scheduling policy. DRL agent is trained and updated through the proximal policy optimization (PPO) algorithm to improve its decision-making performance. Computational results on benchmark datasets show that our DRL-GATv2 algorithm outperforms the widely used priority-rule heuristics in performance.

Keywords: Stochastic resource-constrained project scheduling, Uncertain resource availability, Deep reinforcement learning, Dynamic graph attention variant

1. Introduction

The resource-constrained project scheduling problem (RCPSP) [1] belongs to one of the most widely studied combinatorial optimization problems of NP-Hard. The basic type of RCPSP aims to minimize the duration of a project while satisfying precedence and resource availability constraints. RCPSP and its variants have multiple application scenarios in real world problems, such as software development, construction engineering and manufacturing. However, many studies on RCPSP have hypothesized that resources and activity duration are deterministic, and that activities cannot be interrupted. With the increasing complexity of industrial scenarios in reality, this assumption is gradually becoming less applicable. The existing papers on the stochastic resource-constrained project scheduling problem (SRCPSP) mainly focus on the uncertainty of the activity duration, but pay less attention to uncertain resource quantities and activity interruptions. Compared to SRCPSP with uncertain activity duration, in the SRCPSP-URA scenario, project managers not only need to

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2025.28610

decide which activities to start, but also which activities to interrupt or delay when resources are insufficient.

Deep Reinforcement Learning (DRL) belongs to the category of unsupervised machine learning algorithms. Due to the advantage of combining the perceptual abilities of deep learning with the decision-making abilities of reinforcement learning, DRL can decide which action to choose in a deterministic or stochastic environment to achieve more reward(s) if the model is well-trained. DRL has been used to solve various combinatorial optimization problems such as job shop scheduling problem (JSSP), vehicle routing problem (VRP), travelling salesman problem (TSP) in recent years and has obtained better performance compared to the traditional heuristic algorithms especially in the dynamic environment.

Some research has incorporated graph neural network (GNN) into RL to address graph-structure problems, leveraging their capacity for feature extraction from non-Euclidean data. Unlike conventional architecture such as CNN and MLP, GNN has the advantage of applying to graphs of different sizes without changing the network structure. As an evolution of standard GNN, graph attention network (GAT) [2] enhances this paradigm by integrating self-attention mechanisms, assigning importance weights to neighboring nodes and thereby capturing intricate task dependencies. While GAT variants like GATv2 [3] solves the problem that GAT cannot calculate dynamic attention, their integration with DRL frameworks remains underexplored. No existing literature has investigated the potential of DRL-GATv2 framework for solving SRCPSP-URA. That's the inspiration behind this research.

The rest of this paper is constructed as follows: Section II reviews the related works; Section III presents the basic definitions of SRCPSP-URA; Section IV describes the methodology including the MDP model and DRL-GATv2 algorithm; Section V presents the experiment results and analysis; Conclusions are given in Section VI.

2. Literature review

In SRCPSP, it is assumed that project activities or resources are uncertain and may follow specific distributions in practice. In computation, most of these uncertainties are quantified by using selected probability distribution functions, which help to convert stochastic values to deterministic ones. In practice, the most researched approaches to dealing with projects with stochastic activity duration or resource availability are robust scheduling (proactive & reactive scheduling) and stochastic scheduling. For proactive scheduling, the intent is to take uncertainty into account in advance by imposing time buffers. Reactive scheduling is to reschedule uncertainty-affected activities, when the initial schedule becomes infeasible due to any disruptions.

Different from robust scheduling, the stochastic scheduling approach sequentially produces a schedule using a specific scheduling policy. If a policy attempts to obtain a sequence of all activities before project execution, it is called an open-loop policy. The most widely-adopted approach within open-loop policy consists of various priority rules. Chen et al. [4] examine the performance of 17 priority rule heuristics and the justification technique on the SRCPSP. Recent researches on this method have focused on open-loop policy enhanced by various metaheuristic algorithms such genetic algorithm (GA), tabu search, GRASP and a combination of multiple heuristics. Golab et al. [5] provide a comprehensive review of conventional metaheuristic techniques employed to solve the RCPSP, focusing on optimizing project duration under resource constraints, and detailing the popularity and comparative effectiveness of genetic algorithms, particle swarm optimization, ant colony optimization, and other metaheuristic approaches.

DOI: 10.54254/2755-2721/2025.28610

Instead of optimizing the entire task sequence prior to project execution, a closed-loop policy seeks to find an optimal decision policy for selecting the tasks to start at each decision-point, given the information a decision-maker knows about the current system. However, the performance of the policy is constrained by the curse of dimensionality inherent in the classical exact Bellman recursive solution approach, especially when dealing with large-scale instances. Consequently, there are merely a limited number of research papers on SRCPSP that employ closed-loop policies. Li and Womer [6] develop effective and efficient approximate dynamic programming (ADP) algorithms based on the rollout policy for stochastic scheduling problems with uncertain activity durations. Creemers [7] proposed an exact procedure that uses a backward stochastic dynamic-programming (SDP) recursion to solve SRCPSP, which perform well when solving small- to medium-sized problem instances where activity durations have a moderate-to-high level of variability. Xie et al. [8] implement a rollout-based ADP algorithm with priority-rule heuristic as the base policy to solve SRCPSP-URA, and the algorithm significantly outperforms the priority-rule heuristics for test instances up to 120 activities. Solving scheduling problems via reinforcement learning.

The success of RL as an emerging technique in Atari, AlphaGo has given great encouragement to researchers who hope to use this technique to solve some scheduling problems. Piotr Jędrzejowicz et al. [9] proposed a JABAT multiagent system based on RL to solve RCPSP and parallel processing results in more effective use of the available resources. Wang et al. [10] proposed a RL based approach for adaptive selection of priority rules in dynamic environments. They used transfer learning to reduce the time consumption of off-line training and reuse scheduling models between different cases. Sallam et al. [11] propose a reinforcement learning based meta-heuristic switching approach that integrates multi-operator differential evolution and discrete cuckoo search algorithms within a single framework to address stochastic resource-constrained project scheduling problems characterized by uncertain activity durations. Golab et al. [12] developed a multi-layer feed-forward neural network to solve the standard single-mode RCPSP. This network can automatically select an appropriate priority rule to schedule all activities of the project after the learning process.

Given that scheduling problems can be effectively represented using graphs, several researchers have expanded the application of GNN to address these challenges. Lei et al. [14] proposed an DRL framework to solve TSP and VRP and can be used to solve a multi depot CVRP case without any structural modification. Park et al. [15] introduced GNN to learn node features that embed the spatial structure of JSSP, and used six different types of inputs for accounting various relationships between nodes. Meanwhile, the PPO algorithm was employed for learning a transferable scheduling policy. Zhang et al. [16] develop an innovative architecture based on heterogeneous graph neural network (GNN) and deep reinforcement learning (DRL) to address the dynamic flexible job shop scheduling problem with insufficient transportation resources, aiming to learn high-quality priority dispatching rules end-to-end for minimizing makespan in smart workshops with uncertain and complex logistics conditions. Cai et al. [17] proposed a framework based on RL and GNN to solve RCPSP with resource disruptions and achieves competitive performance compared with some heuristic methods.

From the above literature review, despite the widespread application of DRL and GNN in combinatorial optimization problems, no existing literature has investigated the potential of DRL-GATv2 framework for solving SRCPSP-URA. This gap motivates our exploration of whether DRL-GATv2 framework can enhance decision-making policies in such dynamic, resource-constrained environments.

DOI: 10.54254/2755-2721/2025.28610

3. Problem description

3.1. Problem description and assumptions

Consider a project described by an AON network (V, E), where $V = \{0,1,\ldots,n+1\}$ is a set of activities and the activities 0 and n+1 are two dummy activities. The duration of activity i is d_i. Let $D = \{d_0, d_1, \ldots, d_{n+1}\}$ represents the vector of durations of all activities. Notably, durations for dummy activities d_0 and d_{n+1} are considered 0. E denotes the set of precedence relationships among pairs of activities. An activity can be started only when all of its predecessors have been finished. There are K renewable resources required for project execution with uncertain capacities. The available amount of each renewable resource k at different time periods is a random variable $\widetilde{R_k}$ with a known probability distribution. Let R_{kt} denotes the capacity of the resource k at period t. The requirement of activity i for resource k is r_{ik}. To ensure that $\widetilde{R_k}$ does not lead to an unfeasible problem, we set that the minimum capacity of the resource k is $LR_k = \max_{i \in V}\{r_{ik}\}$. In SRCPSP-URA, fluctuations in resources may cause certain ongoing activities to be forced to interrupt due to insufficient resources. In this article, it is assumed that activities are only allowed to be interrupted due to insufficient resource capacity. We also assume preempt-resume whereby an activity that has been interrupted can be picked up again seamlessly, without requiring any extra time; only the remaining portion of its duration needs to be carried out. The goal of SRCPSP-URA is to find a time- and resource-feasible schedule that minimizes the expected complete time of the whole project.

3.2. MDP formulation

We formulated the scheduling process of SRCPSP-URA as MDP with the following five components.

Stages: A decision stage is defined as a time point when one of the following conditions is met: (a) Due to the reduction of resources, the current execution activity has become unfeasible and has been forced to be interrupted; (b) Due to the increase in resources, new activity have become executable. (c) One or more activities end, leading to the release of resources and making new activities executable. The time point associated with stage j is recorded by t_j.

State: The state of stage j is represented by S_j. The information required for agents to make action choices at each stage includes the completion status of each activity, the consumption and remaining quantity of each resource, and the current set of activities that can be processed. To provide a more specific description of the status information for each stage, we define that $S_j = \left\{R_j^f, R_j, D_j, B_{1j}, \ldots, B_{nj}, \mu_j\right\}$ where B_{ij} denote the status of activity i in stage j, and $B_{ij} = \left\{\bar{d}_{ij}, d_j, R_i, f_{ij}\right\}$. R_j^f and R_{jt} denote the current remaining resources and resource capacity respectively. D_j represents the set of executable activities for the current stage. $\bar{d}_{ij}$ and d_i is respectively defined as the remaining duration of activity i in stage j and the total duration of activity i. R_i denotes the number of all resources required for activity i to be executed. f_{ij} is the status of the activity i in stage j. Specifically, $f_{ij} = 0$ indicates that activity i has not started in stage j; $f_{ij} = 1$ indicates that activity i has been completed in stage j ,while $f_{ij} = 2$

DOI: 10.54254/2755-2721/2025.28610

indicates that activity i is in progress in stage j. μ_j is used to represent the three conditions mentioned above. $\mu_j = 0$ indicates activity interruption caused by insufficient resources, i.e. condition (a); otherwise $\mu_j = 1$ indicates either an increase of resource capacity or completion of some activity, i.e. condition (b) and condition (c).

Actions: D_j represents the set of currently executable activities for the current stage. Unlike classical resource constrained project scheduling problems, D_j is determined by μ_j. When $\mu_j = 0$, $D_j = \{i \in U_j | \forall x \in pred_i, x \in F_j\}$; When $\mu_j = 1$, $D_j = i \in \left\{U_j \backslash A_j \middle| \forall x \in pred_i, x \in F_j \ and \ \forall k \in \{1, \ldots, K\}, r_{ik} \le R_{jk}^f\right\}$, where $pred_i$ represents the set of all the predecessors of activity i, U_j represents all unfinished activities in stage j, F_j represents all completed activities in stage j and A_j represents all activities in progress in stage j.

In stage j, the activity chosen by the agent is defined as a_j, where $a_j \in D_j$. In this article, the deep reinforcement learning strategy $\pi(a_i|s_i)$ generates a probability distribution for over all possible actions (activities) within the set D_j, and an action is subsequently chosen through probabilistic sampling. It should be noted that even if there are multiple activities that can be started simultaneously at the current moment, we stipulate that only one activity should be selected in each stage. Therefore, in different stage, such as decision stage j and stage $j+1$, their corresponding times t_j and t_{j+1} may be equal. The activity of selecting actions will continue until the executable activity set corresponding to the current time is empty, which means we will make the most of resources as much as possible.

Transition process: Let $S_{j+1} = T(S_j, a_j)$ be the state transition function from the stage j to the stage $j+1$, which can be described in detail as follows:

$$\bar{d}_{i(j+1)} = \max\left\{0, \bar{d}_{ij} + t_{j+1} - t_j\right\}, \forall i \in A_j \cup a_j \tag{1}$$

$$F_{j+1} = F_j \cup \{i \in A_j \cup a_j\} | \bar{d}_{i(j+1)} = d_i\} \tag{2}$$

$$f_{i(j+1)} = 2 \ \ if \ f_{ij} = 2 \ or \ i = a_j \tag{3}$$

$$f_{i(j+1)} = 1, if \ i \in F_{j+1} \tag{4}$$

$$R_{(j+1)k}^f = R_{(j+1)k} - \sum_{i \in A_j \cup a_j \backslash F_{j+1}} r_{ik} \tag{5}$$

$$\mu_{j+1} = 0 \ , if \ \exists k \in \{1,2,\ldots,K\} : \sum_{i \in A_j \cup a_j \backslash F_{j+1}} r_{ik} > R_{(j+1)k} \tag{6}$$

$$\mu_{i+1} = 1 \ , if \ \forall k \in \{1,2,\ldots,K\}, \sum_{i \in A_i \cup a_i \backslash F_{i+1}} r_{ik} \le R_{(j+1)k} \ and \ F_{j+1} \ne \emptyset \ \ or \ \exists \ i \in D_j \backslash a_j \ : \ r_{ik} \ge R_{(j+1)k}^f \tag{7}$$

$$f_{i(j+1)} = 0 \ , \ if \ \mu_{j+1} = 0 \ and \ f_{ij} = 0 \ or \ f_{ij} = 2 \tag{8}$$

$$D_{j+1} = \{i \in U_{j+1} | \forall x \in pred_i, x \in F_{j+1}\} \backslash F_{j+1} \ , if \ \mu_{j+1} = 0 \tag{9}$$

DOI: 10.54254/2755-2721/2025.28610

$$D_{j+1} = \left\{ i \in U_{j+1} \backslash A_{j+1} \middle| \forall x \in pred_i, x \in F_{j+1} \; and \; \forall k \in \{1,2,\ldots,K\}, r_{ik} \le R^f_{(j+1)k} \right\} \backslash F_{j+1}, \; if \; \mu_{j+1} = 1 \tag{10}$$

(1) represents the update formula for the remaining time of each activity in the state set S_j. Since there may be more than one activity completed during the process from stage j to stage $j+1$, we use $\max\left\{0, \bar{d}_{ij} + t_{j+1} - t_j\right\}$ to represent the remaining time of activities in stage $j+1$ to prevent negative values. (2) represents the update formula for the completed activity set in stage $j+1$. The update logic is to add all activities that have been executed for a time equal to the total duration to the completed activity set. (3), (4) and (5) represent the update formulas for activity status. (6) provides the update formula for the remaining resources in stage $j+1$, which can be calculated by subtracting the total amount of resources provided by the current environment from the total amount of resources consumed by the ongoing activities in stage $j+1$. (7) and (8) are the update formulas for variable μ_{j+1}. (9) indicates that if there is a shortage of resources leading to an activity interruption, all currently ongoing activities will be removed and activity scheduling will be rescheduled. (10) is the update formula for executable activities in stage $j+1$ when condition (b) or condition (c) occurs.

The example of state transition of the MDP model for the SRCPSP-URA can be depicted in Figure 1.

Reward: Since the goal of using deep reinforcement learning to solve SRCPSP-URA is to learn a scheduling strategy that minimizes the expected total project duration, we use a simple method to define the reward function, as follows:

$$r\left(S_j, a_j, S_{j+1}\right) = -\left(t_{j+1} - t_j\right) \tag{11}$$

This reward function aligns the goal of maximizing the cumulative reward function with the goal of minimizing the total project duration.

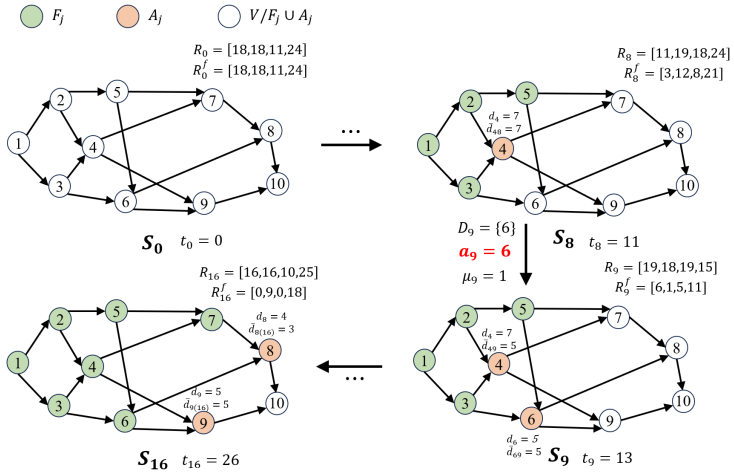

Figure 1. Example of state transition

DOI: 10.54254/2755-2721/2025.28610

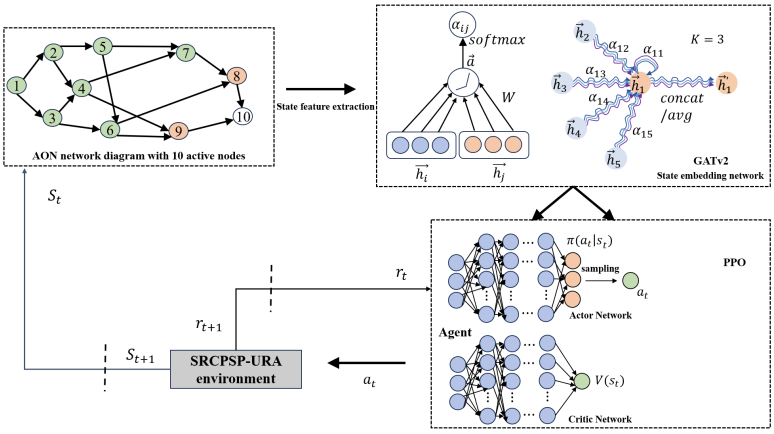

Figure 2. The framework of the proposed method

4. Methodology

The framework of the proposed method is shown in Figure 2, which mainly involves state feature extraction network and policy learning algorithm.

4.1. State feature extraction network

Graph-structured data representations are crucial in various applications, including scheduling problems, where activities and their constraints form a network of nodes and edges. Traditional methods often rely on handcrafted features or fixed rules to encode state and constraint information, which may not generalize well across different problem sizes. In this context, Graph Attention Network v2 (GATv2) offers an advanced and flexible approach for feature extraction by leveraging dynamic attention mechanisms to dynamically assign importance to node neighborhoods.

The GATv2 model begins by initializing each node i with its state information. The key innovation of GATv2 lies in its ability to compute dynamic attention coefficients α_{ij} that adaptively indicate the relative importance of node j's features with respect to node i. Unlike the static attention mechanism in GAT, which assigns a fixed ranking of attention coefficients across all nodes, GATv2 computes these coefficients conditionally based on the query node, thereby offering greater expressiveness. These coefficients are calculated using a modified attention function that applies a non-linear transformation after the concatenation of transformed features, as follows:

$$\alpha_{ij} = \frac{\exp\left(\overrightarrow{a}^T LeakyReLU\left(W\left[\overrightarrow{h_i}\middle\|\overrightarrow{h_j}\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\overrightarrow{a}^T LeakyReLU\left(W\left[\overrightarrow{h_i}\middle\|\overrightarrow{h_k}\right]\right)\right)} \tag{12}$$

Here, $(\bullet)^T$ represents transposition, $\|$ is the concatenation operation, W is a learnable weight matrix that transforms the node features into a latent space, and $\overrightarrow{a}$ is a learnable attention vector. $\mathcal{N}_i$ is the set of adjacent nodes of node i.The application of non-linear transformations after feature concatenation enables GATv2 to compute a universal approximator attention function, making it strictly more expressive than GAT. Using these attention coefficients, the GATv2 model aggregates information from the neighborhood of each node to compute its embedding vector:

DOI: 10.54254/2755-2721/2025.28610

$$\overrightarrow{h'}_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}{}^k W^k \overrightarrow{h}_j\right) \tag{13}$$

$$\overrightarrow{h'}_{i-con} = \Big\|_{k=0}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k W^k \overrightarrow{h}_j\right) \tag{14}$$

$$\overrightarrow{h'}_{i-avg} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k W^k \overrightarrow{h}_j\right) \tag{15}$$

where σ is a non-linear activation function, such as ReLU. (13) describes the aggregation mechanism for the feature vector $\overrightarrow{h'}_i$. (14) and (15) illustrate that the feature vector $\overrightarrow{h'}_i$ of node i from K different attention mechanisms is concatenated or averaged to obtain more detailed description of the node embedding. The above content only involves one GATv2 layer. After stacking multiple GATv2 layers, the final feature embedding can be output which serves as part of the input for the subsequent policy network. The aggregation method of the middle layer is generally concatenation, and if we perform multi-head attention on the final layer of the network, concatenation is no longer sensible—instead, we employ averaging.

After obtaining the final feature embeddings $\overrightarrow{h''}_i$ for each node through multiple GATv2 layers, we perform sum pooling on the N embeddings to derive $h_G = \frac{1}{N}\sum_{i=1}^{N} \overrightarrow{h''}_i$, which represents the embedding information of the entire graph. We use $H_i = \left[\overrightarrow{h''}_i || \mathrm{h_G} \middle\| R^f \middle\| R \middle\| \mu\right]$ as the input to the policy network to get the probability distribution as follows:

$$\pi_\theta(a|s) = Softmax(Mask(MLP_\theta(H_i))) \tag{16}$$

In the implementation, we apply the$MLP_\theta(H_i)$to all activities and utilize a masking mechanism to assign the logits of illegal actions (i.e., those not included in the decision set) a value of negative infinity. This ensures that the probability of these actions being selected is zero when the softmax function is applied. It is clear that our policy architecture is not restricted to a specific number of activities or resources, as the network parameter matrices are independent of the problem size.

4.2. Policy learning algorithm: PPO

In this study, the Proximal Policy Optimization (PPO) algorithm is chosen as the training algorithm to refine the model parameters. PPO stands out as an innovative policy gradient algorithm, offering a relatively straightforward implementation compared to its predecessors, which often involve complex mathematical formulations and computational overheads. It builds upon the traditional policy gradient algorithm and the Trust Region Policy Optimization (TRPO) algorithm.

The PPO algorithm is particularly adept at concurrently updating the model parameters of the Actor and Critic networks via gradient ascent. We adopt the same network architecture for both the Actor and Critic networks as delineated in [18], which has been empirically shown to strike a good balance between model complexity and performance.

The objective function $L^{CLIP}(\Theta)$ can be described as follows:

DOI: 10.54254/2755-2721/2025.28610

$$L^{CLIP}(\Theta)=\mathbb{E}\left[\min\left(r(\Theta)\widehat{A}_t, clip\left(r(\Theta),1-\varepsilon,1+\varepsilon\right)\widehat{A}_t\right)\right] \tag{17}$$

where, $r_t(\Theta)=\frac{\pi_\Theta(a_t|s_t)}{\pi_{old(a_t|s_t)}}$ represents the ratio of new and old probabilities; $\widehat{A}_t$ is the advantage function; ε is the clipping coefficient. In addition, we combine a value function error and entropy bonus term with the objective function to enhance the training performance. The advantage function can indicate the correct direction in which to update the parameters when the estimation of the value function is accurate. The entropy bonus encourages sufficient exploration during training. The complete loss function is written as:

$$Loss(\Theta)=L^{CLIP}(\Theta)+c_vL^V(\Theta)-c_{ent}L^{ENT}(\Theta) \tag{18}$$

where, $L^{ENT}(\Theta)$ is the entropy item, and $L^V(\Theta)$ is the mean square value function error item; c_{ent} and c_v are entropy loss coefficient and critic loss coefficient, respectively.

The PPO algorithm optimizes the aforementioned objective function by adjusting the parameter set in the direction indicated by $\nabla Loss(\Theta)$. Through iterative updates of the parameter set, the actor policy $\pi_\Theta(a_t s_t)$ is trained with the goal of maximizing the expected cumulative sum of rewards, thereby enabling the agent to make optimal decisions in complex environments.

5. Experiment

To evaluate the performance of the algorithm proposed in this article, we construct a resource variability dataset based on the PSPLIB benchmark set [19]. There are four different sizes of datasets – sm30 (single mode, 30 activities), sm60, sm90, sm120. Each of the first three sets includes 480 problem instances while set sm120 includes 600. To ensure the representation of uncertainty in resource capacity, the deterministic resource capacity parameters in the original dataset are replaced with stochastic parameters. We took the original resource availability R_k in the PSPLIB instances as the mean of uncertain resource capacity, then constructed the uniform, exponential and beta distributions in the following way:

Uni (Uniform Distribution): A uniform distribution defined over the interval $[LR_k, 2R_k-LR_k]$, with variance of $(R_k-LR_k)^2/3$

Exp (Exponential Distribution): An exponential distribution with an expected value of R_k and variance of ${R_k}^2$.

Beta (Beta Distribution): A beta distribution defined over the interval $[LR_k, 2R_k-LR_k]$, with variance of $(R_k-LR_k)^2/3$.

The expanded dataset is referred to as PSPLIB-URA. Additionally, due to the fact that the DRL-GATv2 framework is not affected by the scale of the activity, for each type of resource fluctuation situation, we randomly select 50 instances from each size to form a mixed dataset for training. Consequently, a total of 3 models are trained for the three different resource variability scenarios.

Furthermore, the setting of hyperparameters directly affects model performance. Table 1 lists the important hyperparameters involved in our deep reinforcement learning framework for SRCPSP-URA. Since the optimal hyperparameters may vary for models of different sizes, Table 1 exemplifies the optimal hyperparameters for the Uni model. The values of each parameter are determined through grid search.

DOI: 10.54254/2755-2721/2025.28610

Table 1. Setting of hyperparameters for the uni model

Hyperparameter	Value	Hyperparameter	Value
Optimizer	Adam	Learning rate	3×10^{-4}
Layer of policy network	4	Output dimensions of GATv2	64
Hidden dimensions of policy network	64	Total timestamps	3000
GAE parameter (λ)	0.95	the epochs to update the policy	4
Layer of value network	4	the number of mini-batches	8
Hidden dimensions of value network	64	entropy loss coefficient	0.015
Layer of GATv2	4	Clipping parameter ε	0.2
Heads of GATv2	4	Value function coefficient (c_v)	0.5
Hidden dimensions of GATv2	128	Discount factor(γ)	1

Numerous priority rules have been proposed for solving RCPSP, we selected four high-performing priority rule algorithms for comparison. Although our algorithm shares similarities with priority-based approaches, it is insufficient to evaluate it solely against priority algorithms. Therefore, we also included an approximate dynamic programming approach (ADP), as proposed in [8], for comparison purposes.

The algorithms used for comparison are as follows: Latest finishing time (LFT), Latest starting time (LST), Most total successors (MTS), Most direct successor (MDS), and Approximate dynamic programming (ADP).

The subsequent experiments were conducted using Python 3.8 to implement the algorithmic framework. The computations were performed on a computer equipped with an Intel(R) Core (TM) i7-14700HX processor and an Nvidia GeForce RTX 4060 GPU with 24GB of memory.

Tables 2, 3, and 4 respectively present the test results of all algorithms across uniform, beta, and exponential datasets. The average gap is computed as the percentage of deviation of the makespan of the solution from the critical path method (CPM) based makespan (lower bound), which was used in the existing literature to assess the quality of heuristic solutions for SRCPSP [20]. Our proposed algorithm stands out by demonstrating significant improvements over the heuristic solutions across all three test sets. Furthermore, it showcases exceptional generalization performance, consistently generating superior solutions when applied to problems of varying scales compared to all heuristic algorithms. Notably, on the three evaluated datasets, our algorithm not only outperforms the ADP algorithm on both the sm30 and sm120 datasets but also achieves this with considerably smaller standard deviations, thereby underscoring its enhanced stability and robustness in the face of diverse problem instances.

DOI: 10.54254/2755-2721/2025.28610

Table 2. Performance comparison of different algorithm with uniform distribution

Datasets	Index	LFT	LST	MTS	MDS	ADP	Ours
sm30-uni	Average makespan	77.39	79.96	86.44	77.19	66.87	65.14
	Average Gap(std)	48.0%	53.0%	65.4%	47.7%	27.9%(36.93)	24.6%(27.43)
sm60-uni	Average makespan	114.76	123.01	144.42	115.31	93.69	94.03
	Average Gap(std)	58.2%	69.6%	99.1%	59.0%	29.2%(38.77)	29.6%(32.56)
sm90-uni	Average makespan	142.01	156.55	195.66	144.00	·	115.16
	Average Gap(std)	63.5%	80.3%	125.3%	65.8%	·	32.6%(34.72)
sm120-uni	Average makespan	202.29	215.89	268.31	200.66	151.77	150.68
	Average Gap(std)	117.2%	131.8%	188.1%	115.5%	63.0%(55.66)	61.8%(52.91)

Table 3. Performance comparison of different algorithm with beta distribution

Datasets	Index	LFT	LST	MTS	MDS	ADP	Ours
sm30-uni	Average makespan	77.02	80.06	86.63	77.16	66.25	65.16
	Average Gap(std)	47.3%	53.2%	65.7%	47.6%	26.7%(36.46)	24.6%(27.31)
sm60-uni	Average makespan	114.38	123.00	144.45	115.39	93.34	93.95
	Average Gap(std)	57.7%	69.5%	99.1%	59.1%	28.7%(38.32)	29.5%(32.49)
sm90-uni	Average makespan	142.32	156.40	195.05	143.98	·	114.94
	Average Gap(std)	63.9%	80.1%	124.6%	65.8%	·	32.4%(34.32)
sm120-uni	Average makespan	201.99	215.66	268.06	199.80	150.95	150.95
	Average Gap(std)	116.9%	131.6%	187.8%	114.5%	62.1%(54.70)	62.1%(53.21)

Table 4. Performance comparison of different algorithm with exponential distribution

Datasets	Index	LFT	LST	MTS	MDS	ADP	Ours
sm30-uni	Average makespan	92.73	94.84	100.75	92.21	77.94	77.21
	Average Gap(std)	77.4%	81.4%	92.7%	76.4%	49.1%(47.04)	47.7%(39.21)
sm60-uni	Average makespan	150.60	156.29	175.78	147.90	115.19	116.75
	Average Gap(std)	107.6%	115.4%	142.3%	103.9%	58.8%(56.84)	60.9%(51.57)
sm90-uni	Average makespan	196.04	206.22	240.58	191.37	·	144.81
	Average Gap(std)	125.8%	137.3%	177.0%	120.4%	·	66.8%(56.91)
sm120-uni	Average makespan	270.98	280.59	328.96	263.62	190.31	188.48
	Average Gap(std)	191.0%	201.3%	253.2%	183.1%	104.4%(81.91)	102.4%(80.52)

6. Conclusion

This paper tackled SRCPSP-URA by proposing a framework integrating DRL with a GATv2. The goal was to optimize activity scheduling in complex projects under resource constraints and dynamic uncertainties. By developing a MDP model and leveraging GATv2 for feature extraction, our DRL-GATv2 algorithm effectively captured the complexities of scheduling problems, enabling

DOI: 10.54254/2755-2721/2025.28610

the creation of robust scheduling policies. The PPO algorithm enhanced the agent's decision-making in stochastic environments.

Experimental results on PSPLIB-URA datasets showed that our algorithm outperformed priority-rule heuristics and ADP approach, demonstrating superior scheduling performance and robustness across various resource variability scenarios. This study highlights the potential of DRL-GATv2 frameworks in addressing SRCPSP-URA, offering a scalable and adaptable solution for real-world applications. The integration of GATv2 with DRL provides rich feature representations, enhancing generalization across different problem sizes.

In conclusion, our work advances the state-of-the-art in project scheduling under uncertainty, demonstrating the efficacy of DRL-GATv2 frameworks and providing a promising direction for future research.

References

[1] E.L. Demeulemeester , W.S. Herroelen , Project Scheduling: A Research Handbook, Kluwer Academic Publishers, Norwell, MA, 2002.

[2] Veličković, Petar & Cucurull, Guillem & Casanova, Arantxa & Romero, Adriana & Lio, Pietro & Bengio, Y.. (2017). Graph Attention Networks. 10.48550/arXiv.1710.10903.

[3] Brody, Shaked & Alon, Uri & Yahav, Eran. (2021). How Attentive are Graph Attention Networks?. 10.48550/arXiv.2105.14491.

[4] Chen Zhi, Demeulemeester Erik, Bai Sijun, Guo Yuntao. Efficient priority rules for the stochastic resource-constrained project scheduling problem [J]. European Journal of Operational Research, 2018: S037722171830328X.

[5] Golab, A., Gooya, E., Falou, A & Cabon, M. (2022). Review of conventional metaheuristic techniques for resource-constrained project scheduling problem.Journal of Project Management, 7(2), 95-110.

[6] Li Haitao, Womer Norman K. Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming [J]. European Journal of Operational Research, 2015, 246(1): 20-33.

[7] Creemers S. Minimizing the expected makespan of a project with stochastic activity durations under resource constraints [J]. Journal of Scheduling, 2015, 18: 263-273.

[8] B Fang Xie A, C Haitao Li, D Zhe Xu. An Approximate Dynamic Programming Approach to Project Scheduling with Uncertain Resource Availabilities [J]. Applied Mathematical Modelling, 2021.

[9] P. Jędrzejowicz, E. Ratajczak-Ropel, Reinforcement Learning strategies for A-Team solving the Resource-Constrained Project Scheduling Problem, Neurocomputing, Volume 146, 2014, Pages 301-307.

[10] Teng Wang, Wei Cheng, Yahui Zhang, Xiaofeng Hu, Dynamic Selection of Priority Rules Based on Deep Reinforcement Learning for Rescheduling of RCPSP, IFAC-PapersOnLine, Volume 55, Issue 10, 2022, Pages 2144-2149.

[11] K.M. Sallam, R.K. Chakrabortty, M.J. Ryan, A reinforcement learning based multi- method approach for stochastic resource constrained project scheduling problems, Expert Syst. Appl. 169 (2021) 19.

[12] Golab, Amir & Sedgh Gooya, Ehsan & Falou, Ayman & Cabon, Mikael. (2022). A multilayer feed-forward neural network (MLFNN) for the resource-constrained project scheduling problem (RCPSP). Decision Science Letters. 11. 407-418.

[13] Golab, A., Sedgh Gooya, E., Al Falou, A., & Cabon, M. (2022). Investigating the performance of an artificial neural network for solving the resource constrained project scheduling problem (RCPSP). In Pattern Recognition and Tracking XXXIII. 12101, pp. 78-83. Florida: SPIE.

[14] Kun Lei, Peng Guo, Yi Wang, Xiao Wu, Wenchao Zhao, Solve routing problems with a residual edge-graph attention neural network, Neurocomputing, Volume 508, 2022, Pages 79-98.

[15] Park, Junyoung & Chun, Jaehyeong & Kim, Sang & Kim, Youngkook & Park, Jinkyoo. (2021). Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. International Journal of Production Research. 59. 1-18.

[16] Min Zhang, Liang Wang, Fusheng Qiu, Xiaorui Liu, Dynamic scheduling for flexible job shop with insufficient transportation resources via graph neural network and deep reinforcement learning, Computers & Industrial Engineering, Volume 186, 2023, 109718, ISSN 0360-8352.

DOI: 10.54254/2755-2721/2025.28610

[17] Hongxia Cai, Yunqi Bian, Lilan Liu, Deep reinforcement learning for solving resource constrained project scheduling problems with resource disruptions, Robotics and Computer-Integrated Manufacturing, Volume 85, 2024, 102628.
[18] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, Xu Chi, Learning to dispatch for job shop scheduling via deep reinforcement learning, Adv. Neural Inf. Process. Syst. 33 (2020) 1621–1632.
[19] R. Kolisch, A. Sprecher, PSPLIB – a project scheduling problem library, Eur. J. Oper. Res. 96 (1997) 205–216.
[20] F. Ballestín , R. Leus , Resource-constrained project scheduling for timely project completion with stochastic activity durations, Prod. Oper. Manag. 18 (2009) 459–474.