Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Reinforcement Learning for Resource Constrained Project Scheduling Problem with Activity Iterations and Crashing

Inkyung Sung [*] **Bongjun Choi** [*] **Peter Nielsen** [*]

[*] *Operations Research Group, Department of Materials and Production, Aalborg University, Denmark (inkyung_sung@mp.aau.dk, boc@m-tech.aau.dk, peter@mp.aau.dk).*

Abstract: Resource allocation is a key decision-making process in project management that assigns resources to activities of a project and determines the timing of the allocation in a cost and time effective manner. In this research, we address the resource allocation for a project, where iterations between activities of the project exist and the crashing, a method to shorten the duration of an activity by incorporating additional resources, is available. Considering the stochastic nature of project execution, we formulate the resource allocation as a Markov decision process and seek the best resource allocation policy using a deep reinforcement learning algorithm. The feasibility and performance of applying the algorithm to the resource allocation is then investigated by comparison with heuristic rules. © 2020 IFAC

Keywords: Job and activity scheduling, Intelligent decision support systems in manufacturing, Resource Allocation, Reinforcement learning control, Deep Q-learning, Activity Iteration, Crashing

1. INTRODUCTION

Choosing the right resources for activities of a project and allocating the resources to the activities at the right time are key factors to effective project management because available resources are typically sparse. Following the fact, the Resource-Constrained Project Scheduling Problem (RCPSP) and its variants have been actively studied since the last couple of decades (Hartmann and Briskorn, 2010; Lombardi and Milano, 2012; Sitek and Wikarek, 2016).

In the RCPSP, a project consists of a set of activities, where an activity consumes a specific set of resources during its processing time. An activity cannot be started until all of its preceding activities are completed. Then given the limited availability of resources, the activities are scheduled in general to minimize the completion time of the project.

In this research, we address the RCPSP taking into account more realistic dynamics of project execution. First, we consider iterations between activities possibly due to poor quality of output from an activity or the coupled nature of the activities. Such iterations are fundamentals in project execution especially for design related tasks. An illustrative automative stamping die design project with such iterations is presented in Fig. 1, reproduced from Eppinger et al. (1997).

Next, the crashing, a common method in project management, which shortens duration of a project by incorporating additional resources, is considered for the RCPSP (Eisner, 2008). Importantly, linked with the iteration between activities, allocating more resources to an activity can also reduce the probability of an unpleasant iteration occurring, introducing a more complicated trade-off between resources and time.

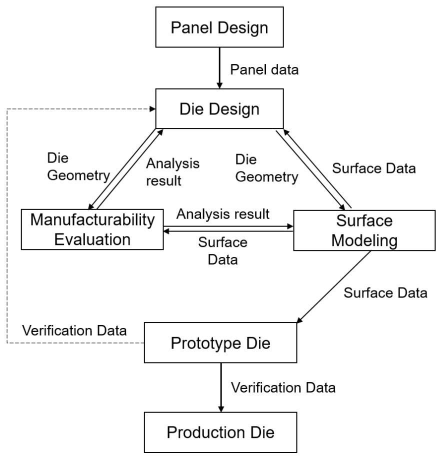


Fig. 1. A design project with activity iterations

Last, we consider the stochastic nature of a project execution. Given a precedence relationship between activities of a project, a sequence of activities necessary to complete a project and activities' durations are all random following known distributions. Note that most studies of the RCPSP have focused on project scheduling under deterministic environments, where all parameters for the problem are known and fixed (Hartmann and Briskorn, 2010). While the RCPSP under stochastic environments has also been addressed in literature, they have rather focused on stochastic duration of activities (Brčić et al., 2012; Herroelen and Leus, 2005) which is often encoutered in a variety of environements (Bocewicz et al., 2016; Michna et al., 2018; Nielsen et al., 2016).

Considering the dynamics of projects as described above and the sequentiality of the resource allocation during project execution, the Stochastic RCPSP with Activity Iterations and Crashing (SRCPSPAIC for short) can be formulated as a Markov Decision Process (MDP). In an MDP, an agent performs an action and a state of the system of interest is updated by the underlying dynamics of the system, partially being controlled by the action. The agent then observes the updated state, receiving rewards (or penalties). The main task in an MDP is to find a policy, which specifies an action given a state of a system, such that the expected total rewards by the policy are maximized.

Recently, reinforcement learning has been well-recognized as an outperforming solution approach for solving MDPs, especially in the domains of robotics and control. Reinforcement learning has also been tested for resource allocation decision-makings in project management demonstrating its performance (Gai and Qiu, 2018; Huang et al., 2011; Mao et al., 2016).

In reinforcement learning, rather than finding an optimal policy in an exact manner (this can be done using dynamic programming), a policy is continuously updated to become optimal based on the trajectories of a target system, i.e. a sequence of the tuples ⟨state, action, reward⟩ sampled from an environment simulating the system.

The applicability of reinforcement learning has been improved with recent advances in Deep Q-Networks (DQN), which helps to apply reinforcement learning even to the domains where modeling features for reinforcement learning are not clear or high-dimensional state spaces are needed. Following the successes of the DQN, in this paper, we solve an MDP designed for the SRCPSPAIC using the deep Q-learning algorithm proposed by Mnih et al. (2015).

The remainder of this paper is constructed as follows. Section 2 describes the formulation of the MDP for the target resource allocation decision-making problem. The deep Q-learning algorithm implemented for the MDP is described in Section 3. The performance of a policy obtained by the algorithm is then evaluated by comparison with heuristic resource allocation rules in Section 4. This paper is finally concluded in Section 5 with remarks on the reinforcement learning application to project management.

2. MDP FOR THE SRCPSPAIC

An MDP is a 4-tuple $\langle S, A, P, R\rangle$, where:

- S is a finite set of states;
- A is a finite set of available actions;
- P is a probability of being in state s' given that the system was in state s and took action a; and,
- R is a reward function, returning the expected immediate reward for taking action a in state s.

In the MDP for the SRCPSPAIC, the state s is a 2-tuple $\langle \tau, \mathbf{o}\rangle$, where τ is the elapsed time from the start of a project and $\mathbf{o}$ is a bit string specifying an activity to be started. Specifically, given N activities in a project indexed from 1 to N, $\mathbf{o}$ is a N-sized bit string, where the ith position of the string corresponds to activity i. Then, activity i can be identified by a bit string having "1" only at the ith position of the string. Note that this kind of data structure is called one-hot encoding and is often used to represent categorical data in machine learning.

Given state s, a project manager determines an action, i.e. to determine the number of resources to be assigned to the activity denoted in the state. We assume that resources are identical and renewable. The action is then represented as one-hot bit string $\mathbf{r}$, where the rth position of the string corresponds to the allocation of r resource units. The cost of an action consists of two parts; one is the resource usage cost which immediately occurrs when we assign resources and the other is associated with the completion time of the project which occurrs when a project is finished.

Suppose that at state $s = \langle \tau, \mathbf{o}_i\rangle$, where activity i is associated with the state element $\mathbf{o}_i$, n number of resources are assigned to activity i, by an action denoted as $\mathbf{r}_n$. The state transition probability is then computed as:

$$P(\langle \tau+\delta, \mathbf{o}_j\rangle | \langle \tau, \mathbf{o}_i\rangle, \mathbf{r}_n) = P(\delta|\mathbf{o}_i, \mathbf{r}_n) \times P(\mathbf{o}_j|\mathbf{o}_i, \mathbf{r}_n),$$

where $P(\delta|\mathbf{o}_i, \mathbf{r}_n)$ is the probability of completing activity i in δ time units with n resources and $P(\mathbf{o}_j|\mathbf{o}_i, \mathbf{r}_n)$ is the probability of getting activity j as a successor of activity i. Unlike a deterministic precedence relationship, where successors of an activity are all fixed, we consider a project, where the successor of the current activity is random.

Fig. 2 describes an example of such a project with five activities. Nodes and arcs in Fig. 2 represent the activities and their precedence relationships, respectively. The number on the arc indicates the likelihood that after completing the activity at the start node of an arc, the project continues with starting the activity at the end node of the arc.

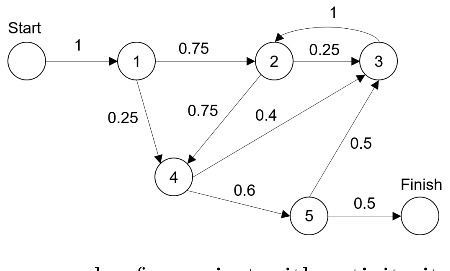


Fig. 2. An example of a project with activity iterations

Suppose that activity 1 is just completed at time 10. Then, with a 0.75 chance, activity 2 will be started. The corresponding state to the situation is represented as $s = \langle 10, (0, 1, 0, 0, 0)\rangle$. When there are three resources available for the project, an action, which assigns two resources to activity 2, is denoted as $a = (0, 1, 0)$. Refer to Eppinger et al. (1994, 1997) for the activity iteration and randomness on it.

3. DEEP REINFORCEMENT LEARNING

A policy $\pi = P(a|s)$ is a function that specifies an action given a state. The expected return of following policy π referred to as the action-value function is then written as:

$$Q_\pi(s, a) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi],$$

where $\gamma \in [0, 1]$ is a discount factor and r_t is an immediate reward received by taking action a_t at state s_t.

The main goal of the MDP framework is to find a policy that maximizes the action-value function. Let us define the optimal action-value function $Q^*(s,a)$ as the maximum action-value function over all policies, i.e. $Q^*(s,a) = \max_\pi \mathbb{E}[Q_\pi(s,a)]$. Given that the optimal action-value function are known, an optimal policy at state s can be immediately determined, that is to select action $a' = \operatorname{argmax}_a Q^*(s,a)$. The optimal action-value function and in turn the optimal policy can be found by solving the following Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a].$$

However, the Bellman optimality equation is nonlinear and there is no closed form of solution for the equation in general. Therefore, the optimal value function is often estimated by iterative updates. It is proven that the estimated action-value function at the ith iteration, $Q_i(s,a)$, set as $\mathbb{E}[r + \gamma \max_{a'} [Q_{i-1}(s',a')|s,a]$, converges to the optimal action-value function as i goes to infinity (Sutton et al., 1998).

Note that such iterative logic is impractical in reality as it requires too many iterations to fully update action-value functions for all states and consumes huge amounts of memory to store the function values. Instead, it is now common to estimate the action-value function using a parameterized approximator. This approach generalizes the experience (training states used to update the action-value function) to new and similar situations, and thus allows to update the action-value function efficiently using relatively small training state data.

In this research, we use a neural network with weight θ for the approximator, referred to as a Q-network. A Q-network can be trained by updating its parameters so that it approximates the optimal action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$. Specifically, by substituting the optimal action-value function with the approximate target value $y = r + \gamma \max_{a'} Q(s',a';\theta)$, the parameter θ can be updated by the following equation:

$$\theta' \leftarrow \theta - \alpha \mathbb{E}_{s,a,r,s'}[(y - Q(s,a;\theta)) \nabla_\theta Q(s,a;\theta)],$$

where α is a learning rate.

For the action-value function update using the Q-network, we implement *deep Q-learning* algorithm proposed by Mnih et al. (2015). The deep Q-learning algorithm in Mnih et al. (2015) has two key features to manage the correlation between samples and the non-stationarity in target values, which are all obstacles to make the Q-network converge to optimal action-value function. In classic Q-learning, once a transition (s,a,s') is made given an action, the action-value function is updated immediately and the next action is determined by the updated action-value function. In doing so, the transitions are all correlated and the target values to update the Q-network is moving at every iteration.

First, the algorithm uses a replay buffer, i.e a dataset of state transitions. The algorithm randomly samples a batch of transactions from the buffer and update a Q-network based on the samples. Using this technique, samples are no longer correlated and the variance of the gradient for the parameter θ update becomes low. Next, the algorithm uses an additional Q-network for target values. The algorithm uses 1) action-value function Q to select actions and 2) target action-value function $\hat{Q}$ to generate the targets for Q updates. This modification increases the stability of the algorithm over that of the classic Q-learning algorithms (refer to Mnih et al. (2013, 2015) for the details of the algorithm). The algorithm is described in Algorithm 1, modified from the algorithm in Mnih et al. (2015) for the SRCPSPAIC.

Algorithm 1 Deep Q-learning with experience replay and target network

```
1: Initialize replay memory $D$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value $\hat{Q}$ with weights $\theta^- = \theta$
4: for episode = 1, ... $M$ do
5:     Initialize sequence $s_1 = \{s_1\}$
6:     for $t = 1, \dots, T$ do
7:         With probability $\epsilon$, select a random action $a_t$
8:         Otherwise, select $a_t = \operatorname{argmax}_a Q(s_t, a; \theta)$
9:         Execute action $a_t$ and observe reward $r_t$ and the next state $s_{t+1}$
10:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
11:        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
12:        if episode terminates at step $j+1$ then
13:            $y_j = r_j$
14:        else
15:            $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$.
16:        end if
17:        Update $\theta$ to minimize $(y_j - Q(s_j, a_j; \theta))^2$
18:        Every C steps, reset $\hat{Q} = Q$
19:    end for
20: end for
```

4. COMPUTATIONAL RESULTS

We test the performance of the reinforcement learning algorithm in the SRCPSPAIC on a set of hypothetical projects. For the test, we vary the number of activities in a project from 10 to 90 by an increment of 20, resulting in five projects. For each project, we generate a sequence of activities as a baseline of a precedence relationship and randomly add links between non-adjacency activities including the *backward* iterations, which is the arc from an activity to its one of predecessors. The number of available resources are set as 20 for all projects. The cost of project execution is composed of two parts. One is a cost for resource usage (\$/resource · time) and the other is a cost associated with the tardiness of a project completion. The due date of a project is set arbitrarily.

As benchmark solutions, we implement four heuristic rules, namely *min, mid, max, random.* The first three rules are to allocate the minimum, middle, and maximum number of available resources, respectively, to an activity and the last one is to allocate a randomly determined number of resources to an activity. Due to the randomness in project execution (e.g. stochastic activity duration and iterations between activities), given a project, we sample 100 scenarios (i.e. a possible situation for the project execution) and evaluate the performance of the five resource allocation algorithms (reinforcement learning

+ four heuristic rules) over the scenarios. The experiments were performed using a machine with 32 Intel Xeon cores (2.10GHz), an NVIDIA Quadro P600 processor graphics processor and 2 GB memory.

The test results are summarized in Table 1. The cost improvement by the reinforcement learning algorithm over the heuristics (*Gap*) and the time taken for the reinforcement algorithm to provide better solutions than the heuristics (*Time*) are presented. Note that during the algorithm run, at every iteration where an Q-network is updated, we check if the policy from the current Q-network can provide a better performance than the heuristics and record the runtime when the policy outperforms the heuristics. This measurement is not the total runtime of the reinforcement learning algorithm, which is the time to train the Q-network using the given number of episodes.

Table 1. Performance Comparison

Activity	Gap †				Time (sec)
number	Min	Mid	Max	Random	
10	4	295	508	284	96
30	506	154	286	137	94
50	856	84	200	89	2,129
70	855	59	126	158	6,666
90	491	50	38	101	17,341

†:100× $(\text{Cost}_{heuristic} - \text{Cost}_{proposed})/\text{Cost}_{proposed}$

First, it is observed that the reinforcement learning algorithm outperforms the simple heuristic rules, showing dramatic cost savings. Unlike the heuristic rules, the resource allocation by reinforcement learning determines the number of resources to allocate, taking into account situations of a project execution (e.g. how long a project is executing and how far a project completion is from its due date), which is critical to minimize the cost of a project's execution.

We also observe that for small projects (with 10 and 30 activities), the reinforcement learning algorithm could train an outperforming policy in a short time, whereas the algorithm takes a relatively long time to train the policy to an adequate level for large projects (with 50, 70, and 90 activities). For a large project, considering the increased number of states necessary to visit during the reinforcement learning algorithm to train an Q-network, this result is straightforward.

The training quality of the Q-network over iterations of the algorithm is shown in Fig. 3. In the figure, the expected cost of a project execution following the policy obtained from the Q-network at each iteration is plotted. The expected costs by the heuristic rules are also presented as reference points.

From Fig. 3, it is clear that the expected cost by a policy from the Q-network is decreased over the iterations, implying that the Q-network is appropriately trained by the algorithm.

Based on the observations, it is concluded that the policy trained by the reinforcement learning algorithm can outperform in the SRCPSPAIC compared with the four heuristic rules. That said, more advanced heuristics should be addressed in future research to clearly demonstrate the performance of the proposed approach in the SRCP-SPAIC.

5. DISCUSSION AND CONCLUDING REMARKS

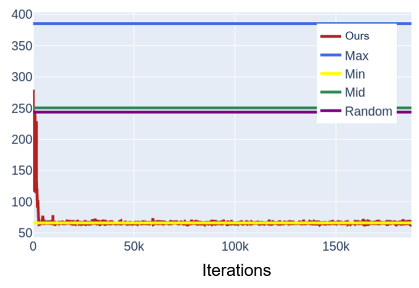
In this study, we investigate the feasibility of applying reinforcement learning for the resource allocation decisions in a project execution. Specifically, we set the resource constrained project scheduling problem with iterations between activities of a project and the crashing of the activity duration. For resource allocation to such projects, the deep reinforcement learning algorithm designed in this study outperforms the heuristic rules with respect to minimizing the expected cost for project execution.

Based on the demonstrated performance of the algorithm, we believe that the characteristics of a project and the reinforcement learning approach addressed in this study can even contribute to the industrial 4.0 scenario, where technologies that can handle uncertainties involved in manufacturing and production processes are essential. Imagine a system where multiple on-line robots are conducting tasks with stochastic durations (Gola and Kłosowski, 2019) and their reliabilities are stochastically decreasing over time (Gola, 2019), making it difficult to predict the reliability level of the system. Rule-based or human operator heavily involved control/management systems cannot properly address the complicated and stochastic dynamics of such systems.

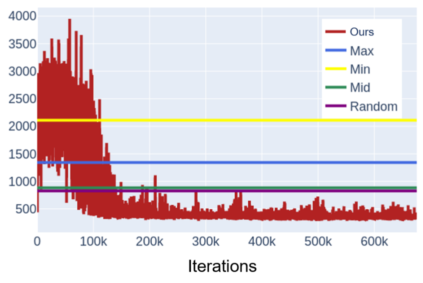
Let us close this paper by discussing issues in the reinforcement learning implementation for the SRCPSPAIC and room to further improve the applicability of reinforcement learning to the problem. First, there are many parameters to be tuned for the reinforcement learning implementation (see many parameters in Algorithm 1 including the hidden parameters of the Q-network construction/initialization). To set the parameters at appropriate values, and accordingly maximize the performance of the algorithm, running the algorithm multiple times while adjusting the involved parameters are inevitable, which is undesirable especially for on-line resource allocation. Poorly, the parameters tuned would/should be updated for a different project setting.

Furthermore, unlike the applications where the dynamics of a target system are known or available with approximations (e.g. robot kinematics), it is difficult to predict the behavior of a project's execution. Obviously, the performance of reinforcement learning is limited by the representativeness of samples used for the algorithm training. Therefore, the historical data and experience on a target project are critical to build a simulation environment for reinforcement learning and to improve the performance of the approach in the project management domain.

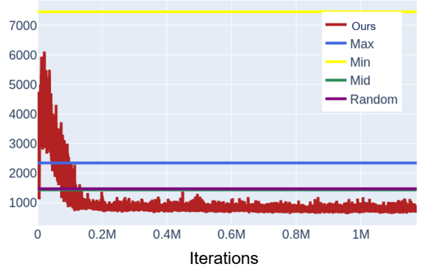
Last, it is difficult for a decision-maker or a stakeholder of a project to understand and interpret the decision-making logic of the policy obtained from reinforcement learning and this often decreases their willingness to follow the policy. To improve the applicability of reinforcement learning, extra steps such as extracting underlying decision-making rules or managerial insights from the policy can be conducted.



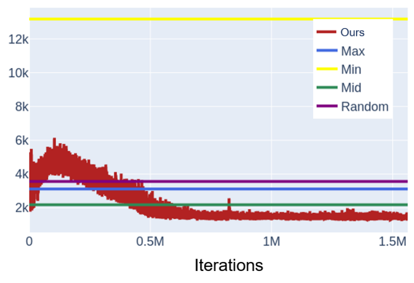
(a) N=10



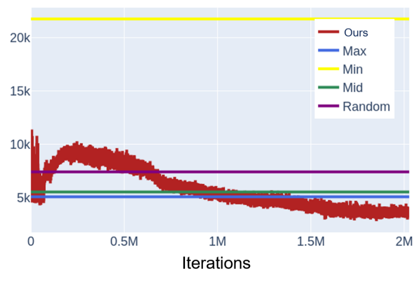
(b) N=30



(c) N=50



(d) N=70



(e) N=90

Fig. 3. The performance of the reinforcement learning algorithm over the training iterations for projects with different numbers of activities (N)

REFERENCES

Bocewicz, G., Nielsen, I.E., and Banaszak, Z.A. (2016). Production flows scheduling subject to fuzzy processing time constraints. *International Journal of Computer Integrated Manufacturing*, 29(10), 1105–1127.

Brčić, M., Kalpić, D., and Fertalj, K. (2012). Resource constrained project scheduling under uncertainty: a survey. In *23rd Central European Conference on Information and Intelligent Systems.*

Eisner, H. (2008). *Essentials of project and systems engineering management.* John Wiley & Sons.

Eppinger, S.D., Nukala, M.V., and Whitney, D.E. (1997). Generalised models of design interation using signal flow graphs. *Research in Engineering Design*, 9(2), 112–123.

Eppinger, S.D., Whitney, D.E., Smith, R.P., and Gebala, D.A. (1994). A model-based method for organizing tasks in product development. *Research in engineering design*, 6(1), 1–13.

Gai, K. and Qiu, M. (2018). Optimal resource allocation using reinforcement learning for IoT content-centric services. *Applied Soft Computing*, 70, 12–21.

Gola, A. (2019). Reliability analysis of reconfigurable manufacturing system structures using computer simulation methods. *Eksploatacja i Niezawodność*, 21(1).

Gola, A. and Kłosowski, G. (2019). Development of computer-controlled material handling model by means of fuzzy logic and genetic algorithms. *Neurocomputing*, 338, 381–392.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1), 1–14.

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research*, 165(2), 289–306.

Huang, Z., van der Aalst, W.M., Lu, X., and Duan, H. (2011). Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering*, 70(1), 127–145.

Lombardi, M. and Milano, M. (2012). Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17(1), 51–85.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 50–56. ACM.

Michna, Z., Nielsen, P., and Nielsen, I.E. (2018). The impact of stochastic lead times on the bullwhip effect–a theoretical insight. *Production & Manufacturing Research*, 6(1), 190–200.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.

Nielsen, I., Wójcik, R., Bocewicz, G., and Banaszak, Z. (2016). Multimodal processes optimization subject to fuzzy operation time constraints: declarative modeling approach. *Frontiers of Information Technology & Electronic Engineering*, 17(4), 338–347.

Sitek, P. and Wikarek, J. (2016). A constraint-based approach to modeling and solving resource-constrained scheduling problems. In *International Conference on Computational Collective Intelligence*, 423–433. Springer.

Sutton, R.S., Barto, A.G., et al. (1998). *Introduction to reinforcement learning*, volume 2. MIT press Cambridge.