Made available through Montana State University's ScholarWorks

A Reinforcement Learning Approach to the Dynamic Job Scheduling Problem

Farshina Nazrul Shimim and Bradley M. Whitaker

Accessibility Disclaimer:

For a more accessible version of this document, please submit an accessibility request form through the Montana State University Library website.

A Reinforcement Learning Approach to the Dynamic Job Scheduling Problem

Farshina Nazrul Shimim
Electrical and Computer Engineering
Montana State University
Bozeman, USA
farshin.nazrulshimim@student.montana.edu

Bradley M. Whitaker
Electrical and Computer Engineering
Montana State University
Bozeman, USA
bradley.whitaker1@montana.edu

2022 IEEE Green Energy and Smart Systems Conference (IGESSC) | 978-1-6654-9866-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IGESSC55810.2022.9955328

***Abstract*—Scheduling or day-ahead planning improves the efficiency of a process and often leads to other advantages such as energy savings and increased revenue. However, most real-world scheduling problems are very complicated and are usually affected by several external parameters. Hence, finding the best schedule given a set of jobs requires extensive calculations that increase exponentially with the number of jobs. Traditional schedulers are, at times, unable to address uncertainties in the system. This paper proposes a Reinforcement Learning approach for solving the Job Scheduling Problem in a dynamic environment with an aim to minimize the peak instantaneous electricity consumption. The training instance is randomly reset after a certain period and the solver uses online training to adapt to the new environment. Simulation results show that both the proposed approach and a Genetic Algorithm-based approach achieve the minimum peak power consumption possible, which is 58% less than on-demand dispatch. Also, for 82.2% of the simulations, our method finds a better schedule than its initialization.**

***Index Terms*—Deep reinforcement learning, peak power minimization, Actor-critic method, job scheduling, optimization, neural networks, sequential decision making.**

I. Introduction

For decades, researchers have focused on solving the Job scheduling Problem (JSP), which is a classic problem in the computer science and operational management domains [1], [2], [3]. In this problem, given a set of jobs with a flexible set of parameters (i.e., arrival time, deadline, duration) the goal is to find the best combination of these jobs that either maximizes or minimizes the given objective function(s) (e.g., minimizing the total processing time, number of machines used). Based on the schedule (possible combinations of job start and end times) it is possible to obtain a wide range of values for each of the objective functions [4], [5]. Generally, problems like JSP are known as combinatorial optimization problems (COP) where each combination of the variables is a potential solution [6].

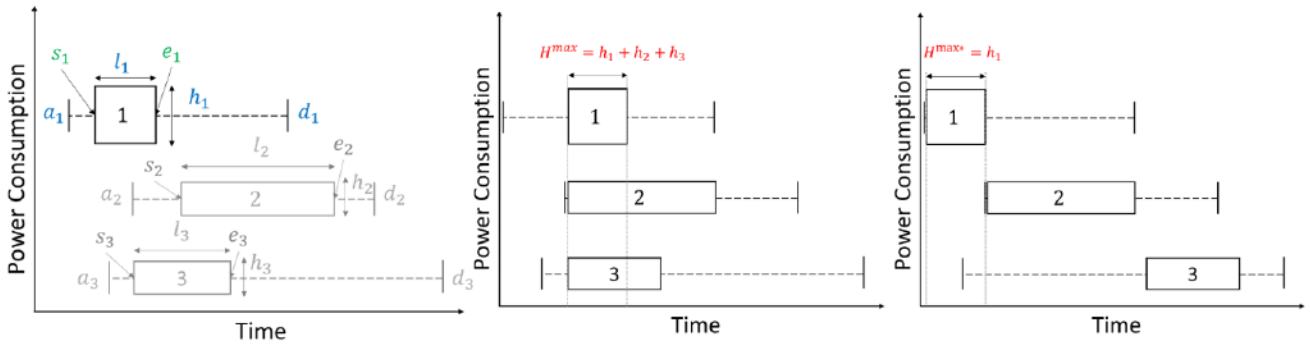
Several real-life problems under the COP cannot be solved optimally within a feasible time period using an exact algorithm. This is because there can be trillions of possible combinations even for a small dataset, and the objective function of the problem might be too complex to visualize [7], [8]. These problems are termed as NP-hard problems [9] and the optimal solution cannot be guaranteed without trying all feasible solutions by brute force in exponential time. An interesting fact about this class of problems is that even though obtaining an optimal solution takes non-polynomial time, it is possible to verify if a given solution is optimal in polynomial time.

To find a 'good enough' solution for the COPs, state-of-the-art solvers often use approximations. This way, a COP can be solved in feasible time, but without any guarantee that the obtained solution is optimal. In most cases, these state-of-the-art solvers are *metaheuristic* [10]. While they yield quality solutions, they are static in nature: for a new problem instance, the solver must solve them from scratch, with no 'learning' from the previous solving process. These solvers are also unable to address uncertainties in the system (e.g., machine failure, change in arrival time) [11]. In such cases, we need solvers that can adapt to these uncertainties. Machine learning is an important concept in this regard [12].

Machine learning (ML) is a domain of data science that allows a machine to learn from its previous experience of performing one or more tasks. With appropriate training, a ML model identifies implicit patterns from the data it receives and can perform certain tasks without human intervention and explicit coding [13], [14]. Reinforcement learning (RL) is a special field of ML techniques that are suitable to solve NP-hard problems if the problems are modelled correctly [15], [16]. In RL, an agent takes sequential decisions under certain conditions by performing trial-and-error experiments. In recent years, RL algorithms (RLAs) have significantly evolved by combining with deep learning techniques, emerging to a new field called Deep Reinforcement Learning (DRL) [17].

Various research has been conducted over the years to solve the JSP using RLAs/DRLAs. Wang *et al.* [18] provide a detailed review of several works in this regard. There are significant variations in the environment design, application sectors and solution approaches for this problem. Considering the uncertainties associated with the actual production environment, [19] solves the JSP for multiple machines by selecting the best dispatch rule from a set of ten dispatch rules using a Deep Q-Network. In [20], a tabular Q-learning method wasn implemented for real-time scheduling using multiple dispatch rules. A cooperative multi-agent deep reinforcement

Authorized licensed use limited to: Montana State University Library. Downloaded on June 23,2026 at 23:48:47 UTC from IEEE Xplore. Restrictions apply.



(a) Job parameters (b) A 'bad' solution (c) An optimal solution

Fig. 1: Solving JSP for 3 jobs: (a) the given parameters (arrival time, duration, deadline, height) and the variables (start time, end time) for the jobs, (b) a feasible solution where all jobs start at the same time. Here, in the interval $[st_1, e_1]$, $H^{max} = h_1 + h_2 + h_3$ which results in maximum possible peak power consumption. (c) another possible solution where the jobs are scheduled avoiding overlaps. Here, in the interval $[st_1, e_1]$, $H^{max} = h_1$ considering $h_1 > h_2, h_3$. As H^{max} cannot be reduced further, it is the minimum peak achieved and the schedule shown in this figure is an optimal solution.

learning based method is proposed in [21] to obtain multiple objectives in the Flexible JSP environment.

A variation of JSP has been investigated in our work, where each job represents a responsive smart appliance in a home. Some of these appliances are dishwasher, electric heater, lawnmower etc. Day-ahead scheduling of these appliances can be both cost effective and energy efficient [22]. In our work, we focus on designing an Artificial Intelligence agent that can schedule a wide variety of jobs (appliances) under dynamic environments in order to minimize the peak electricity consumption. The scheduling scheme uses actor-critic, a Deep Reinforcement Learning Algorithm to create an optimized schedule given the JSP instance.

Unlike most of the research done in this domain, our work is not limited to selecting from a set of preset dispatch rules. Rather, through a trial-and-error sequence, the proposed method creates its own dispatch rule by approximating a series of actions as outputs of the actor-critic algorithm. For the performance evaluation, we compare the solution provided by our proposed RL-based AI solver with 3 baseline approaches and 1 other approach using Genetic Algorithm (GA) [23]. The baseline approaches are: on-arrival scheduling, modified first-in-first-out, and random scheduling.

II. Methodology

A. Problem Description

1) Job Scheduling Problem (JSP): In JSP, given a set of jobs $\mathcal{J}$, we need to find the best schedule for the jobs in order to obtain a specific objective. Each job $j \in \mathcal{J}$ is represented as a 4-tuple: (a_j, l_j, d_j, h_j), where the parameters denote the arrival time, length/duration, deadline and height (resource requirement), respectively [22], [24], [18]. As we are considering each smart appliance in a home as a job, the height in this case refers to the power consumption for that appliance in the duration of l_j within the time interval $[a_j, d_j]$. The objective is to schedule all $j \in J$ jobs with a view to minimizing the peak power consumption for a specific interval $[0, T]$. Fig. 1 illustrates a 3-job example of JSP showing its parameters. It also compares 2 potential solutions and argues why one is preferred over the other.

If job j is assigned a start time $st_j : st_j \in [a_j, d_j - l_j]$ and an end time $e_j = st_j + l_j : e_j \in [a_j + l_j, d_j]$, then at each time-step $t : t \in [0, T]$, the instantaneous power consumption is,

$$H(t) = \begin{cases} \sum_{j=1}^{n} h_j, & \text{if } t \in [st_j, e_j] \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The peak power consumption for the whole interval is,

$$H^{max} = \max_{t \in [0,T]} H(t) \quad (2)$$

Hence, the overall goal for the scheduling would be finding the schedule of jobs that results in the minimum H^{max} from all possible combinations of jobs,

$$S_{sol} = \arg\min H^{max} \quad (3)$$

where, S_{sol} is the schedule, i.e., set of all start times, $< st_j \forall j \in J >$.

B. Proposed RL-based JSP solver

1) Markov decision Process (MDP): A MDP is a mathematical model for problems that requires sequential decision making and performing certain optimizations [15], [25], [26]. It is defined as a 5- tuple, $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma >$, where, the first 4 symbols represent finite sets of states, actions, immediate rewards obtained from state $s_i \in \mathcal{S}$ to another state $s_j \in \mathcal{S}$ using an action $a_{ij} \in \mathcal{A}$, and transition probabilities associated with each transition between any two states, respectively. The last parameter, γ, is the discount factor that decides how much weight should be placed on the rewards associated with the more recent transitions.

In our work, the states are the execution times of all jobs. The actions are the shifts on these execution times (whether

Authorized licensed use limited to: Montana State University Library. Downloaded on June 23,2026 at 23:48:47 UTC from IEEE Xplore. Restrictions apply.

a job should start earlier or later). Both states and actions are discrete. At each move, the peak power is calculated for the whole instance. The immediate reward for each transition is given by comparing the updated peak power to the previous peak power and the global best peak obtained so far. For more details, refer to the 'Description of Environment' section. The transition probability is unknown in this case, but that does not create any inconvenience for the method that we used (see more details in 'Actor-critic Method' section). As the rewards for all moves are important for training the model, γ is set very close to 1 (0.99 in this paper).

2) Description of Environment: The environment is a discrete 2-d field (F) in which the x-axis represents the time-steps and the y-axis represents the power consumption. The design of the field was inspired from [27]. The time-steps are arbitrary in our case, they can be seconds, minutes or hours based on user preference. F contains n players, each player denoting a job. The base of the jobs are located on the x-axis ($y = 0$). The jobs overlap if they are executed at the same time. Therefore, $size(F) = [T, h_j^{max}]$.

The game starts with a set of random start time for each job. There are 2 possible actions for each job: move 1 step left (start 1 time-step earlier) or move 1 step right (start 1 time-step later). At each move, the agent selects one job and performs a move associated with that job. After that move, the field is updated and the total consumed peak power is calculated, H_m^{max}, where $m \in \mathcal{M}$ is a valid move. The current peak H_m^{max} is then compared to the peak in the previous move H_{m-1}^{max} (except for the first move). At the same time, H_m^{max} is compared to the so far global best peak value H_{global}^{max}. The immediate reward r_{ij}^m is defined as,

$$r_{ij}^m = \begin{cases} 1, & \text{if } H_m^{max} < H_{m-1}^{max} \\ 100, & \text{if } H_m^{max} \leq H_{global}^{max} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

At the beginning of each episode, the whole environment is reset, i.e., the jobs start randomly and the AI agent faces a new instance. It is responsible for moving the jobs left or right to find a set of start times that results in minimum peak electricity consumption.

C. Solution Approaches

1) Reinforcement Learning: Reinforcement Learning (RL) is a sequential decision-making learning technique for an Artificially Intelligent agent in a completely new environment [15], [26]. Here, an agent learns to select the best options under certain conditions by performing operations in a trial-and-error manner. The agent explores different states $s \in \mathcal{S}$ of the environment, and takes actions $a \in \mathcal{A}$. Once any action is taken, the state changes accordingly and the environment provides a feedback to the agent in the form of a reward or a penalty, denoted by $r \in \mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. Thus, each decision of the agent results in some reward, and at the end of successive decisions, the agent receives a chain of rewards (also known as long-term reward). The overall goal for the agent is to take the actions that maximize the long time reward. Eventually, with more exploration, the agent gains knowledge of the environment and learns which sequence of actions result in the highest reward.

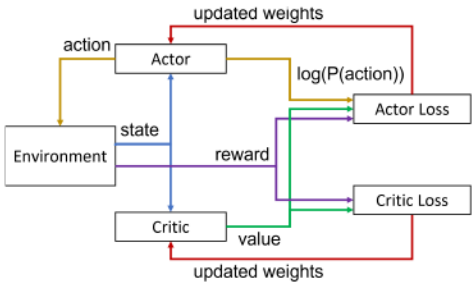


Fig. 2: Actor-critic method.

The specialty of RL is, instead of learning a task from labelled data, the agent learns the best actions based on the rewards it gets. The rewards come only after an action is taken, and they are highly situation-dependent. The agent has to learn the best action based on the instantaneous state/situation. RL algorithms that emply deep neural networks are known as Deep Reinforcement Learning (DRL) algorithms. These learning techniques are more data-driven and can be used to solve high-dimensional, complex tasks.

2) Actor-critic Method: The actor-critic method is a type of model-free DRL method [26]. The model-free methods are used to solve a problem without using the transition probability for the state-action pairs, $\mathcal{T} : \mathcal{S} \times \mathcal{A}$. In the actor-critic method, the agent samples actions from the environment with replacement (known as bootstrapping) and updates the model based on its current estimates of the values of the selected state-action pair.

Fig. 2 shows the architecture of this method [28]. Two approximators (neural networks) are used to independently estimate the policy function and the value function, respectively. The policy function (π) is a softmax function over all possible actions given a state. For each action, this function represents the probability of that action being selected by the agent in that state. The model or the network that computes the policy function is known as *actor*. The value function is obtained by the *critic* model and it estimates the reward for the selected action given the state. After N moves, the losses are calculated for both the actor and the critic networks and the weights are updated accordingly. To calculate the losses, the expected returns corresponding to the policy function (π), $\rho^{\pi}(s_t, a_t)$, are calculated combining the rewards obtained for all moves using the discount factor, γ, as follows:

$$\rho^{\pi}(s_t, a_t) = \sum_{m=1}^{N} \gamma^{m-1} r_{ij}^{N-m+1} \tag{5}$$

The actor loss is calculated from the following equation:

$$\phi_{actor} = \sum_{m=1}^{N} \log \pi_{\theta}(a_t|s_t)\ \alpha^{\pi,\gamma}(s_t, a_t), \tag{6}$$

Authorized licensed use limited to: Montana State University Library. Downloaded on June 23,2026 at 23:48:47 UTC from IEEE Xplore. Restrictions apply.

where $\pi_\theta(a_t|s_t)$ is the approximated policy function, the output of the actor network. θ is the policy parameter.

The advantage function is a measure of how good or advantageous the specific action $a_t \in \pi^\theta$ given s_t is over any randomly chosen action from $\mathcal{A}$. It is defined as follows:

$$\alpha^{\pi,\gamma}(s_t, a_t) = \rho^\pi(s_t, a_t) - V^\pi(s_t), \tag{7}$$

where $V^\pi(s_t)$ is the approximated value function, the output of the critic network.

The critic loss is the Huber loss [29] between the actual reward and the approximated reward, defined as follows:

$$\phi_{critic} = \begin{cases} \frac{1}{2}\left(\alpha^{\pi,\gamma}(s_t, a_t)\right)^2, & \text{for } |\alpha^{\pi,\gamma}(s_t, a_t)| \leq \delta \\ \delta(|\alpha^{\pi,\gamma}(s_t, a_t)| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}, \tag{8}$$

where δ is a threshold parameter. The total loss is obtained by adding Eqns. 6 and 8, respectively.

In our work, the input for both the actor and the critic network is the location of the jobs along the x-axis, that is, $s_t[0] \in \mathcal{S}$. The actor network output has all the possible actions $a_t \in \mathcal{A}$. Recall that for each job, there are 2 actions (go 1 left or right). The actor network has, therefore, $2n$ output nodes for n jobs. The critic network has only 1 node in its output layer, approximating the value function.

III. Numerical Experimentation

A. Dataset

The proposed RL-based job scheduler was tested on a modified version of benchmark industrial dataset [30]. From 'instance_0' of the dataset, we filtered the jobs starting and ending in between time-step 800 and 899. Therefore, the modified dataset had 100 time-steps and 17 non-preemptive jobs with different arrivals, deadlines, durations, and heights. We discretized the job-heights to the next largest integers (using the *ceiling* operator) to make the field size finite. Thus, the field size, $size(F)$ becomes $\{100 \times 889\}$, 889 being the height of the tallest job. The actual height is 888.89, so there was a difference of 0.11 because of the discretization. It is considered that the heights corresponds to kilowatts (kW) and, for our application, the discretization difference is negligible.

For the each job $j \in \mathcal{J}$ in the dataset, there were several possible start times within the interval $[a_j, d_j - l_j]$. The total possible combinations for the test instance was calculated as $\approx 2.44 \times 10^{25}$. For reference, it took about 50 minutes to to determine the solution for 10^5 combinations. Extrapolating this linearly to 10^{25} combinations, it would take over 10^{14} years to solve the problem using a brute-force approach.

B. Experimental Setup

We modeled our proposed solver using the *Tensorflow* [31] and *Keras* [32] APIs in Python. We did a grid-search for the maximum steps per episode and the hidden layer configuration for the networks. The max steps were set as 1500, and a good network hidden layer configuration was a 2-layered network with 4000 and 1000 hidden nodes, respectively. We run the algorithm for a total of 1000 episodes. The discount factor, γ, was set as 0.99.

The networks were trained using back-propagation method, and we used the *Adam* optimizer from *Keras* [33]. The parameters of *Adam* were set as follows: *learning_rate*= $0.0001, \beta_1 = 0.999, \beta_2 = 0.999, \epsilon = 0.001$. The training was done on an online basis, where the agent is expected to learn forever, facing new pattern in each episode in order to dynamically adapt to any changes. After several trials, it was observed that a large learning rate causes instability. On the other hand, for the online training, the learning rate is expected not to decay substantially with time. Therefore, for this work, the initial learning rate was chosen to be small with fairly high exponential decay rates, β_1 and β_2.

C. Baseline Approaches

To evaluate the performance with the proposed RL_based approach, it was compared with 4 baseline approaches. They are briefly discussed below.

1) On-demand Scheduling: This approach schedules the jobs exactly at their arrival time with no waiting [22]. Therefore, $st_j = a_j \ \forall j \in \mathcal{J}$.

2) Random Scheduling: Here, the start time of the jobs are assigned randomly within the feasible window of execution [24]. For this approach, $st_j = rand\,(a_j, (d_j - l_j)) \ \ \forall j \in \mathcal{J}$.

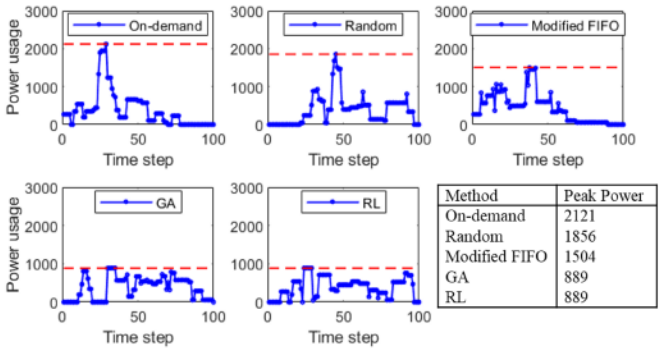
3) Modified FIFO: FIFO is the acronym for first-in, first-out method in scheduling [34]. We modified this method to compare to our algorithm in the following way: For each job $j \in \mathcal{J}$, assign the start time, st_j using the following equation,

$$st_j = \begin{cases} e_{j-1} + \delta & \text{if } j > 1 \ \& \ e_{j-1} \in [a_j, d_j - l_j] \\ a_j & \text{otherwise} \end{cases} \tag{9}$$

where, δ is a very small number, defined as the time taken for communication between the jobs and the agent. Eqn. 9 refers that a job should start after the previous job ends if that is feasible. Otherwise, it should start at the earliest possible time-step, i.e., On-demand.

4) Genetic Algorithm: GA is one type of evolutionary algorithm which is inspired by the 'survival of the fittest' criteria [23], [10]. Evolutionary algorithms are mostly used in cases where the optimal solution or fitness is not known prior to solving, and the objective function and the constraints are non-linear. With each 'generation', the algorithm identifies a better solution set and the process continues until the solutions converge [35]. In this work, each 'chromosome' is an array of the start times of all jobs, $st_j \ \forall j \in \mathcal{J}$. Their corresponding fitness is the resultant peak power from executing the jobs. This approach was implemented using the *geneticalgorithm* library from *Py-PI*. The GA parameters were selected as follows: variable type = *int*, max. iteration = 100, population size = 200, mutation probability = 0.1, elite ratio = 0.01, crossover probability = 0.7, parents portion = 0.3, crossover type = *uniform*, max. iteration without improvement = 50.

Authorized licensed use limited to: Montana State University Library. Downloaded on June 23,2026 at 23:48:47 UTC from IEEE Xplore. Restrictions apply.



Method	Peak Power
On-demand	2121
Random	1856
Modified FIFO	1504
GA	889
RL	889

Fig. 3: Scheduling solutions and their corresponding peak power consumptions for the test case using 5 different approaches: On-demand, Random, Modified FIFO, GA, and finally the proposed RL-based method.

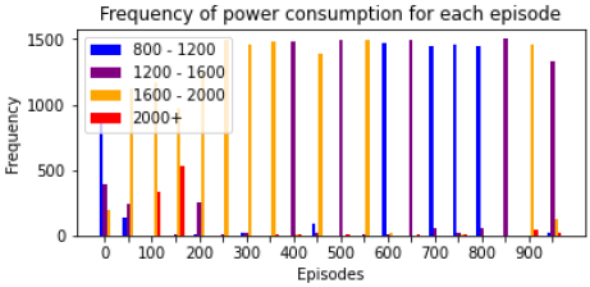


Fig. 4: Frequency of the peak power consumption for all steps in an episode. The blue bars are associated with the good solutions whereas the red bars represent the bad ones.

D. Results and Discussion

Fig. 3 shows the solutions obtained using the 4 baseline approaches and the proposed RL-based approach for the test case. The corresponding peaks are shown with the red dashed lines for each subplot. The table in 3 shows the peak power obtained using each method. From the figures, it is seen that if the jobs are scheduled at their arrival time (On-demand), the peak power occurs between time-step 20 and 30 and it is more than 2000 kW (2121 kW). The peak powers for the random scheduling and the Modified FIFO scheduling are less than On-demand, yet somewhere in between 1500 and 2000 kW. Both the ML approaches (GA and RL) provide a peak power of 889 kW, which is the height of the tallest job. As the peak cannot be lower than this height, 889 is indeed the optimal value for this test instance. The results show that the optimal solution reduces the peak power consumption by 1232 kW (58%) compared to the On-demand scheduling.

An interesting observation is that even though both GA and RL provide the optimal peak, their corresponding scheduling solutions (start times) significantly differ from each other. Therefore, it is clear that multiple optimal solutions exist for this particular problem instance.

Fig. 4 shows the frequency of the peak power obtained at each step for each episode within specific ranges using the proposed approach. The bars are shown for every 50 episodes.

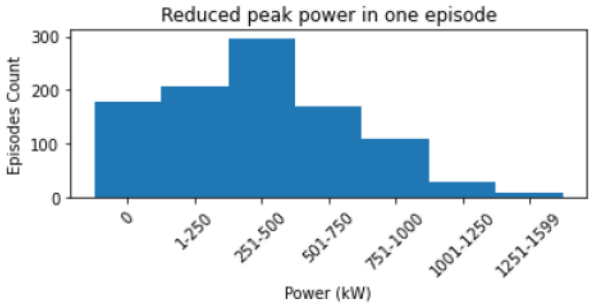


Fig. 5: A histogram of the reduced peak power consumption (initial peak power - minimum peak power) by the RL agent in one episode.

The first episode was pure exploration for the AI agent in the environment. The actions in the subsequent episodes were influenced by the actions taken in the previous episodes.

Recall that the actor and the critic models are updated only once at the end of each episode and then the jobs are randomly initialized, so the agent faces an entirely new environment. Through the course of episodes, the agent develops generalization ability and learns to schedule randomly located jobs. From the graph in Fig. 4, in the later episodes, the agent tends to schedule more effectively, resulting in peaks in a lower-range (mostly in between $800-1600$). However, at each episode, the moves taken by the agent is limited (max 1500). Therefore, for a bad initialization, the agent might not have sufficient moves to re-allocate the jobs to achieve a minimum peak.

Fig. 5 shows a histogram of the amount of peak power that is reduced in an episode (the difference between the initial peak and the minimum peak obtained in that episode) for all 1000 episodes. From the plot, it is seen that for 822 episodes (82.2%) the algorithm was able to find a better solution than the initial one. The remaining episodes could not reduce the power because of an unfavorable initialization and the limitation in the maximum number of steps in one episode. The average peak reduced excluding and including the artifacts are 468.36 kW and 384.99 kW, respectively. The standard deviation including the artifacts is 322.753 kW and decreases to 296.11 kW when the artifacts are excluded. Here to be noted, for some episodes, the initial peak power was more than 2700 kW (the max initial peak was 2711 kW). Note that the peak for On-demand scheduling was 2121 kW, so there exists worse combinations than On-demand. Compared to the worst-case scenario that the agent faced, the power saving for the optimal scheduling is 1822 kW (67.2%). In a single episode, the maximum peak power that the agent reduced is 1575 kW. Nevertheless, the agent found the optimal solution in 11 different episodes.

IV. Conclusion and Future Works

This paper presents a data-driven and model-free RL-based approach to solve the JSP under a dynamic environment involving smart appliances in a home. Using this approach,

Authorized licensed use limited to: Montana State University Library. Downloaded on June 23,2026 at 23:48:47 UTC from IEEE Xplore. Restrictions apply.

the execution time of the appliances were selected with a view to minimizing the peak electricity consumption within a certain time period. The AI agent was trained in an online fashion using the actor-critic algorithm. On the other hand, the JSP instance was designed as a game environment. The agent learns to efficiently schedule the jobs by playing the game and receiving scores (rewards) for good schedules.

As the best schedule is not known beforehand, the effectiveness of any schedule is measured comparing its peak power to the previously obtained ones. The proposed approach is compared to 4 approaches (3 baseline and 1 ML-based): On-demand, random, Modified FIFO, and GA. Results show that both the proposed approach and GA achieve the minimum possible peak power consumption (889 kW), which represents 58.1% improvement over On-demand scheduling, 52.1% over random scheduling and 40.9% over scheduling using Modified FIFO, respectively. However, unlike GA, RL is dynamic in nature and is adaptive to any sudden changes in the system.

The proposed work has several scopes for future expansion. For now, the actions of the agent are heavily influenced by its previous experiences from the encounter with the environment. The action selection paradigm can be further investigated to parametrize it. In future work, the agent will face more uncertainties in the system, e.g., sudden termination of one or more jobs. Those jobs will be replaced by pseudo-jobs to keep the network outputs the same size. This step will allow to transfer the experience of an existing model to solve new instances with different JSP parameters. Currently, we are using a small dataset. The approach itself can be modified to implement on a larger instance by introducing a moving-window-based method for scheduling. In brief, the adaptive capability of RL-based algorithms will be further investigated and improved for a wide variety of JSP instances.

REFERENCES

[1] C. Zhang, W. Song, Z. Cao, J. Z. 0002, P. S. Tan, and C. Xu, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Advances in Neural Information Processing Systems*, 2020.

[2] B.-A. Han and J.-J. Yang, "Research on adaptive job shop scheduling problems based on dueling double dqn," *IEEE Access*, vol. 8, pp. 186 474–186 495, 2020.

[3] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Applied Soft Computing*, vol. 91, p. 106208, 2020.

[4] Q. Yan, W. Wu, and H. Wang, "Deep reinforcement learning for distributed flow shop scheduling with flexible maintenance," *Machines*, vol. 10, no. 3, 2022.

[5] Y. C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, 2005.

[6] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[7] P. Bonami, A. Lodi, and G. Zarpellon, "Learning a classification of mixed-integer quadratic programming problems," *CPAIOR*, pp. 595–604, 2018.

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *International conference on learning representations*, 2017.

[9] J. Hoogeveen, J. Lenstra, and B. Veltman, "Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard," *European Journal of Operational Research*, vol. 89, no. 1, pp. 172–175, 1996.

[10] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[11] L. Wang, X. Hu, Y. Wang, S. Xu, S. Ma, K. Yang, Z. Liu, and W. Wang, "Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning," *Computer Networks*, vol. 190, p. 107969, 2021.

[12] T. Onomi, Y. Maenami, and K. Nakajima, "Superconducting neural network for solving a combinatorial optimization problem," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 701–704, 2011.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[14] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[15] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2014.

[16] E. Özcan, M. Misir, G. Ochoa, and E. Burke, "A reinforcement learning - great-deluge hyper-heuristic for examination timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, pp. 39–59, 01 2010.

[17] J. X. Chen, "The evolution of computing: Alphago," *Computing in Science Engineering*, vol. 18, no. 4, pp. 4–7, 2016.

[18] L. Wang, Z. Pan, and J. Wang, "A review of reinforcement learning based intelligent optimization for manufacturing scheduling," *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.

[19] Y. Zhao, Y. Wang, Y. Tan, J. Zhang, and H. Yu, "Dynamic jobshop scheduling algorithm based on deep q network," *IEEE Access*, vol. 9, pp. 122 995–123 011, 2021.

[20] M. Ghaleb, H. A. Namoura, and S. Taghipour, "Reinforcement learning-based real-time scheduling under random machine breakdowns and other disturbances: A case study," *2021 Annual Reliability and Maintainability Symposium (RAMS)*, pp. 1–8, 2021.

[21] J. Popper, W. Motsch, A. David, T. Petzsche, and M. Ruskowski, "Utilizing multi-agent deep reinforcement learning for flexible job shop scheduling under sustainable viewpoints," *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pp. 1–6, 2021.

[22] S. Yaw and B. Mumey, "Scheduling non-preemptible jobs to minimize peak demand," *Algorithms*, vol. 10, p. 122, 10 2017.

[23] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[24] E. Pryor, S. Yaw, and B. Mumey, "Scheduling jobs with precedence constraints to minimize peak demand," *Combinatorial Optimization and Applications*, pp. 140–150, 12 2020.

[25] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[27] M. AL-Ma'amari. (2020) Deep reinforcement learning with python. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-with-python-part-1-creating-the-environment-dad6e0237d2d

[28] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[29] P. J. Huber, "Robust estimation of a location parameter," *Annals of Mathematical Statistics*, vol. 35, pp. 73–101, Mar. 1964.

[30] N. Ludwig, L. Barth, D. Wagner, and V. Hagenmeyer, "Industrial demand-side flexibility: A benchmark data set," *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, p. 460–473, 2019.

[31] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[32] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[33] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[34] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Pearson, 2014.

[35] H. Ospina-Mateus, S. Sana, J. Chedid, and F. Gazabon Arrieta, "Application of genetic algorithm to job scheduling under ergonomic constraints in manufacturing industry," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, p. 2063–2090, 05 2019.

Authorized licensed use limited to: Montana State University Library. Downloaded on June 23,2026 at 23:48:47 UTC from IEEE Xplore. Restrictions apply.