Optimizing Job Allocation using Reinforcement Learning with Graph Neural Networks

Lars C.P.M. Quaedvlieg
EPFL, Switzerland
`{[first name].[last name]}@epfl.ch`

Abstract

Efficient job allocation in complex scheduling problems poses significant challenges in real-world applications. In this report, we propose a novel approach that leverages the power of Reinforcement Learning (RL) and Graph Neural Networks (GNNs) to tackle the Job Allocation Problem (JAP). The JAP involves allocating a maximum set of jobs to available resources while considering several constraints. Our approach enables learning of adaptive policies through trial-and-error interactions with the environment while exploiting the graph-structured data of the problem. By leveraging RL, we eliminate the need for manual annotation, a major bottleneck in supervised learning approaches. Experimental evaluations on synthetic and real-world data demonstrate the effectiveness and generalizability of our proposed approach, outperforming baseline algorithms and showcasing its potential for optimizing job allocation in complex scheduling problems.

1 Introduction

Efficient job allocation plays a vital role in diverse fields, including healthcare, logistics, and manufacturing, enabling optimal utilization of resources and ensuring timely completion of tasks [Owliya et al., 2012, Bash and Forman, 2007, Kumar et al., 2018, Vijayakumar et al., 2022]. The Job Allocation Problem (JAP), which is introduced and investigated in this report, involves assigning a maximum number of jobs to available resources, taking into account various constraints. Traditional approaches to general scheduling [Crama, 1997, Pinedo, 2012], rooted in Combinatorial Optimization (CO), have provided valuable insights into the inherent challenges and trade-offs in solving allocation problems, but they often struggle to handle the increasing complexity and real-time dynamics of modern applications [Wolsey and Nemhauser, 1999, Chaudhry and Khan, 2016].

To address these limitations, recent research has turned to machine learning techniques to learn effective schedules [Zhang et al., 2020, 2022, Park et al., 2021, Chen et al., 2021]. However, the requirement of annotated data in supervised learning poses a problem for scheduling problems. Hence, it is currently infeasible to do supervised learning on NP-Hard problems [Yehuda et al., 2020].

Our approach showcases the use of reinforcement learning (RL) as a powerful technique to overcome the challenges posed by the requirement of annotated data. By learning directly from simulations and observing performance signals Kaelbling et al. [1996], our framework mitigates the need for extensive manual labeling. This enables us to leverage the benefits of RL for job allocation in a more practical and scalable manner.

Furthermore, we use graph neural networks to exploit the graph structure of the problem, as is commonly done in machine learning for CO problems [Wang and Gombolay, 2020, Gasse et al., 2019, Schuetz et al., 2022].

To evaluate the performance of our proposed approach, we conducted experiments on both real-world and synthetic datasets. We compared our model with baseline algorithms. We demonstrate that the

GNN consistently outperformed the baseline algorithms. Furthermore, out-of-distribution testing revealed the generalizability of the GNN.

The remaining structure of the report is as follows: in Sec. 2, we give a background and discuss the most promising research related to our work. Then, in Sec. 3, we formalize the JAP. In Sec. 4, we discuss the RL framework used to approach the problem. Then, we describe the model architecture and the training process in Sec. 5. Finally, we discuss experimental results in Sec. 6 and make conclusions in Sec. 7.

2 Related Works

The approach introduced in this report lies in the fields of combinatorial optimization, graph neural networks, and reinforcement learning. In this section, we discuss the most prominent research and explain its relation to our work.

Combinatorial Optimization (CO) has been a cornerstone in tackling complex allocation problems. Previous research has delved into various formulations and algorithms to optimize the allocation of limited resources efficiently. The exploration of heuristics [Colorni et al., 1996, Lorena and Narciso, 1996], approximation algorithms [Vazirani, 2001, Turek et al., 1992], and exact methods [Wolsey and Nemhauser, 1999] has paved the way for understanding the inherent challenges and trade-offs in solving CO problems. In particular, studies addressing job allocation [Penmatsa and Chronopoulos, 2006], nurse rostering [Burke et al., 2004], and other scheduling problems [Chaudhry and Khan, 2016] have contributed valuable insights into the allocation domain.

In recent years, **Graph Neural Networks (GNNs)** have proven to be powerful tools for analyzing and learning on graph-structured data [Zhou et al., 2020, Wu et al., 2020]. Advanced architectures such as the Graph Attention Network (GAT) [Veličković et al., 2018] and Graph Isomorphism Network (GIN) [Xu et al., 2019] have demonstrated remarkable capabilities in tackling node/edge-level and graph-level machine learning tasks, respectively. Importantly, the application of GNNs to combinatorial optimization has shown promise in effectively leveraging the inherent graph structure found in various problem domains [Gasse et al., 2019, Cappart et al., 2021, Schuetz et al., 2022], while also providing the benefit of invariance to the size of the graph. In this work, we utilize graph attention networks in this work since we are learning representations of edges.

Reinforcement Learning (RL) provides a principled framework for learning optimal policies through trial-and-error interactions with an environment. RL has demonstrated remarkable successes in various domains, from game-playing agents to robotic control [Mahmud et al., 2018, Polydoros and Nalpantidis, 2017]. When applied to scheduling problems, RL offers the potential to learn adaptive policies that optimize assignments and resource utilization [Zhang et al., 2020, Park et al., 2021].

As the JAP is a novel problem introduced in this report, there are no existing works directly addressing this problem. Nevertheless, previous studies have examined supervised learning approaches in other similar scheduling problems, by for example learning priority dispatch rules Chen et al. [2021], Ingimundardottir and Runarsson [2011] to compute allocation policies, but the requirement of manual labels presents a significant challenge. To overcome this limitation, recent research has turned its attention to RL [Park et al., 2021, Zhang et al., 2020, Kayhan and Yildiz, 2021], which can learn directly from simulations and observe performance without relying on manual labels.

In this report, we investigate the viability of RL for the JAP. We achieve this by formulating the problem within the RL framework and leveraging Deep Q-learning [Mnih et al., 2015] in conjunction with graph neural networks.

3 The Job Allocation Problem

In this section, we formally introduce the Job Allocation Problem (JAP), and its classical optimization formulation.

Notation: A graph $G(V, E)$ denotes a directed graph where V represents the vertex set and E represents the edge set. We note that $\{i, j\} \in E$ represents an undirected edge, while $(i, j) \in E$ represents a directed edge. We define $\mathcal{N}_{\text{in}}(v)$ and $\mathcal{N}_{\text{out}}(v)$ as the in- and out-neighborhoods of vertex v. The in- and out-degree of a vertex v are then defined as $d_{\text{in}}(v) = |\mathcal{N}_{\text{in}}(v)|$ and $d_{\text{out}}(v) = |\mathcal{N}_{\text{out}}(v)|$.

Definition 1 (Job Allocation Graph)**.** *An arbitrary graph $G(A, B)$ is called a job allocation graph if we can partition G as $G(P \cup J, S \cup C)$ where $P \cap J = \emptyset$ and $S \cap C = \emptyset$, such that $\forall \{v, w\} \in S$: $[v \in P \wedge w \in J] \vee [v \in J \wedge w \in P]$ and $\forall (v, w) \in C : v, w \in J$.*

Whilst the remainder of this report will talk about people and jobs, any graph that satisfies definition 1 can be described under the framework introduced in this report. Fig. 1 depicts an example of such a graph, where P represent the people vertices and J represents the job vertex set. The structure of the graph is bipartite between the people and job vertices, with additional edges from jobs to other jobs. We call the undirected edge set between jobs and people the "selection set" S. For directed edges from jobs to other jobs, we define the "conflict set" C. Formally, a directed edge $(j_i, j_k) \in C$ semantically means that if the job j_i is selected by a person p, then the job j_k cannot be done by the same person. An edge $(p, j) \in S$ means that person p can do the job j.

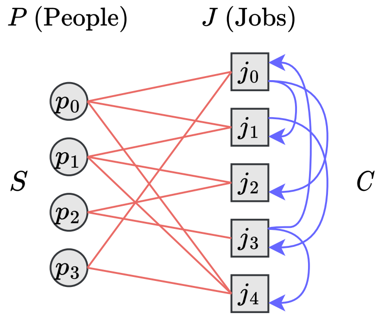


Figure 1: Example of the problem instance. We have individuals represented by $p_0, \cdots, p_3 \in P$, and jobs denoted by $j_0, \cdots, j_4 \in J$. The red selection edges from set S connect people to jobs, signifying that a person is qualified to do a job. Additionally, there are directed blue conflict edges in set C. These connect jobs, indicating that if a person p is assigned to a job vertex j, then that person cannot also be assigned to any $j' \in \mathcal{N}_{\text{out}}(j)$.

Definition 2 (Maximum Job Allocation)**.** *Given a graph $G(P \cup J, S \cup C)$ that adheres to definition 1, a maximum job allocation is a solution to the following constrained maximization problem:*

$$\begin{aligned} &\textit{maximize } |A| \ , \\ &\quad \textit{s.t. } A \subseteq S \ , \\ &\textit{and } \forall \{p_i, j_i\}, \{p_i, j_k\} \in A : (j_i, j_k) \in C \vee (j_k, j_i) \in C \Rightarrow i = k \ . \end{aligned}$$

We call a job allocation any subset $A \subseteq S$ that satisfies the constraints in definition 2. The goal of the JAP is to find the maximum number of job assignments such that there are no conflicts between the jobs assigned to the same person.

4 Optimal Job Allocations through Reinforcement Learning

In Sec. 3, we discuss how to formalize the JAP and how an optimum solution to the problem is defined. In this section, we discuss how we formulate the JAP as a sequential decision-making problem using the Reinforcement Learning framework.

Definition 3 (Markov Decision Process (MDP))**.** *A Markov Decision Process (MDP) is defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is a set of states and $\mathcal{A}$ is a set of actions. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function that captures the probability of transitioning to a state $s' \in \mathcal{S}$ given a current state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$ taken in s. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a function that defines the reward given when transitioning to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$. Finally, $\gamma \in [0, 1]$ is the discount factor, which emphasizes the importance of future rewards.*

When a problem can be formalized as an MDP, as seen in definition 3, it can be solved using RL. We will begin by describing the state space $\mathcal{S}$, which will be the set of all job allocation graphs. This means that any job allocation graph is equivalent to a state in the MDP.

Given a job allocation graph, the actions should allow for a job assignment to be selected. The idea behind this approach is the fact that assignments can be selected sequentially while eliminating constraint violations with the transition function. Hence, the action space $\mathcal{A}$ for a job allocation graph $G(P \cup J, S \cup C)$ is equivalent to the selection set S, as these are all possible job assignments.

In order to ensure a valid job allocation graph is maintained after choosing an action, the transition function behaves as follows: given a graph $G(P \cup J, S \cup C)$ and action $\{p, j\}$, it deterministically maps to a new graph $G'(P \cup J, S' \cup C')$, where $S' = S \setminus \{\{p, j\}\} \setminus \{\{p, j_i\} | (j, j_i) \in C\}$ [1]. As can be seen, the transition function eliminates any assignments that person p cannot do anymore after choosing to do job j. This ensures that any transition to a new graph maintains feasibility.

An episode with T steps is completed when a terminal state is reached. A terminal state is a graph $G(P \cup J, S \cup C)$ where no further assignments are possible, that is, $S = \emptyset$. The set of all the job allocations in the episode $A = \{a_1, a_2, \cdots, a_T\}$ is a feasible solution to the problem. It is important to note that, starting from any job allocation graph, the optimum solution A^* can be reached by picking the assignments in A^* in any order as actions. One consideration is that this implies that there are a lot of symmetries, which can make it more difficult to solve the problem. This has been addressed in Kwon et al. [2020], but this work has not been adopted in this report.

The reward function gives a reward of 1 at each step of the environment, which means that the maximum cumulative reward attainable can be bounded from above by $|S|$. The discount factor γ is set to 1 since the total cumulative reward from the beginning of the episode is equal to the cardinality of the set of solutions.

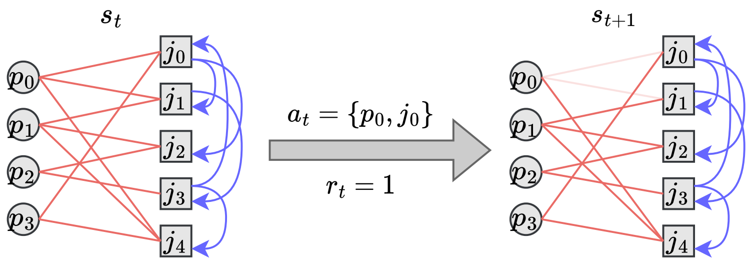


Figure 2: Example of a transition (s_t, a_t, r_t, s_{t+1}) in the MDP. When action $\{p_0, j_0\}$ is picked, its edge and the edge between p_0 and j_1, which conflicts with j_0, are removed from the graph in s_{t+1} (depicted as transparent edges).

An example of a transition of the model can be seen in Fig. 2. Since the state space $\mathcal{S}$ is too large for tabular reinforcement learning methods, we utilize techniques for reinforcement learning with function approximation. In Sec. 5, we describe how we do this using graph neural networks.

5 Graph Neural Network-Based Algorithm for MIS

As discussed in Sec. 4, the state space of the MDP is too large to be approached with tabular reinforcement learning. By leveraging the graph structure of the states, we use graph neural networks to approximate Q-values. We first discuss the architecture of the model in Sec. 5.1, and then describe the training pipeline in Sec. 5.2.

5.1 Function Approximation with Graph Neural Networks

The goal is to design a model $Q_\theta(s, a) \approx Q^*(s, a)$, where $\theta \in \Theta$ are parameters and $Q^*(s, a)$ are the optimum Q-values. We develop a novel graph neural network module, the *Context-Aware Embedding (CAE)* module, in order to effectively utilize the specific graph structure of the problem. We then use this module in the complete model architecture.

Given a graph $G(J \cup P, S \cup C)$ and initial vertex embeddings $\mu^0 \in \mathbb{R}^{|P| \times 2}, \nu^0 \in \mathbb{R}^{|J| \times 2}$, which are just the out-degrees of the vertices for the two edge types, the high-level model architecture can be seen in Fig. 3. Initially, the graph and the embeddings are sequentially put through K CAE modules, parameterized by $\theta_i \in \Theta$, where Θ for all $i = 1, \cdots, K$. We note that Θ is the parameter

[1]It is important to realize that, since job allocation graphs are equivalent to states, this represents a deterministic transition with $\mathbb{P}[G'|G, \{p, j\}] = 1$.

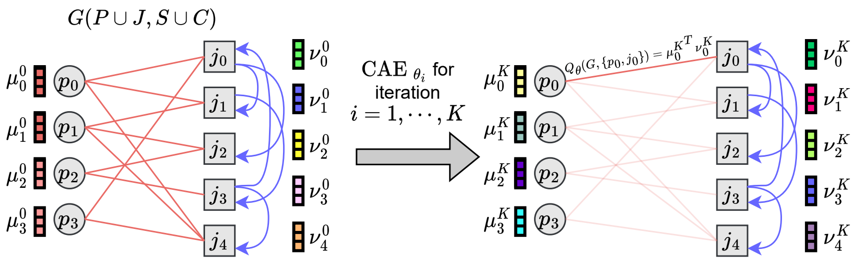

$G(P \cup J, S \cup C)$
CAE $_{\theta_i}$ for iteration $i = 1, \cdots, K$
$Q_\theta(G, \{p_0, j_0\}) = \mu_0^{K^T} \nu_0^K$

Figure 3: Overview of the model architecture. First, a job allocation graph $G(J \cup P, S \cup C)$ with initial node embeddings $\mu^0 \in \mathbb{R}^{|P| \times 2}, \nu^0 \in \mathbb{R}^{|J| \times 2}$ is put through K Context-Aware Embedding modules with parameters $\theta_i \in \Theta$ for $i = 1, \cdots, K$. Afterward, Q-values can be predicted by doing an inner product of the corresponding vertex embeddings. For example, for the highlighted edge $\{p_0, j_0\}$, $Q_\theta(G, \{p_0, j_0\}) = \mu_0^{K^T} \nu_0^K$.

space. These modules compute node embeddings μ^i for person vertices and ν^i for job vertices for $i = 1, \cdots, K$. Finally, Q-values can easily be computed by taking a dot product between the final embeddings for two vertices of a given selection edge. For example, the estimated Q-value of choosing the person p to do the job j can be calculated as $Q_\theta(G, \{p, j\}) = \mu_p^{K^T} \nu_j^K$.

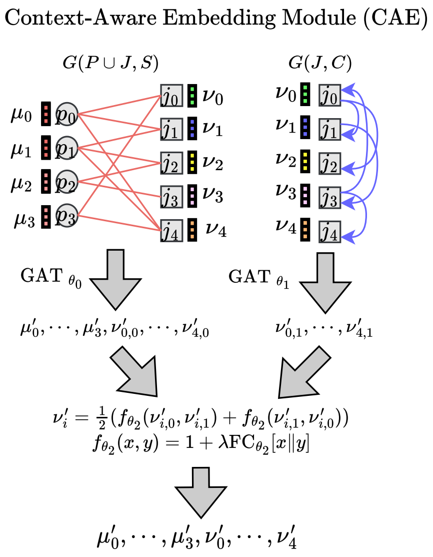

Context-Aware Embedding Module (CAE)
$G(P \cup J, S)$
$G(J, C)$
GAT $_{\theta_0}$
GAT $_{\theta_1}$
$\mu'_0, \cdots, \mu'_3, \nu'_{0,0}, \cdots, \nu'_{4,0}$
$\nu'_{0,1}, \cdots, \nu'_{4,1}$
$\nu'_i = \frac{1}{2}(f_{\theta_2}(\nu'_{i,0}, \nu'_{i,1}) + f_{\theta_2}(\nu'_{i,1}, \nu'_{i,0}))$
$f_{\theta_2}(x, y) = 1 + \lambda \mathrm{FC}_{\theta_2}[x \| y]$
$\mu'_0, \cdots, \mu'_3, \nu'_0, \cdots, \nu'_4$

Figure 4: Overview of the Context-Aware Embedding Module. Given a graph $G(P \cup J, S \cup C)$ and vertex embeddings $\mu \in \mathbb{R}^{|P| \times d_{in}}, \nu \in \mathbb{R}^{|J| \times d_{in}}$, the module first splits it into two subgraphs $G(P \cup J, S)$ and $G(J, C)$. Then, these subgraphs are put through their own GAT layers, parameterized by $\theta_0, \theta_1 \in \Theta$. The final embeddings of the job vertices are then computed by combining them symmetrically using the learned linear function f_{θ_2} with parameters $\theta_2 \in \Theta$.

Context-Aware Embedding (CAE) Module:

The module is depicted in Fig. 4. From a graph $G(J \cup P, S \cup C)$ and its current vertex embeddings $\mu \in \mathbb{R}^{|P| \times d_{\text{in}}}$ and $\nu \in \mathbb{R}^{|J| \times d_{\text{in}}}$ where d_{in} is the dimensionality of a single vertex embedding. The CAE module operates by splitting G into two subgraphs $G(J \cup P, S)$ and $G(J, C)$. Then, these graphs are put through their own attention layers [Veličković et al., 2018] with parameters $\theta_0, \theta_1 \in \Theta$ to update the node embeddings of the subgraphs. This splitting into subgraphs is inspired by Zhang et al. [2022], which utilizes the same idea, since the edges have semantically different meanings. In the case of the JAP, selection edges are good to have, whilst conflict edges are bad to have. Since we have two sets of embeddings for the job vertictes after the attention layers, let's say $\nu'_0 \in \mathbb{R}^{|P| \times d_{\text{out}}}$ and $\nu'_1 \in \mathbb{R}^{|J| \times d_{\text{out}}}$ where d_{out} is the output dimensionality of the vertex embeddings, we combine them into one as follows:

$$\nu = \frac{1}{2}\left(f_{\theta_2}(\nu_0,\nu_1) + f_{\theta_2}(\nu_1,\nu_0)\right) \ , \\ f_{\theta_2}(x,y) = 1 + \lambda \mathrm{FC}_{\theta_2}([x\|y]) \ . \tag{1}$$

Here, $\lambda \in \mathbb{R}$ is a learned parameter that modulates how far to deviate from a baseline value of 1. Furthermore, $[x\|y] \in \mathbb{R}^{|x|+|y|}$ represents the vector concatenation of x and y, and FC_{θ_2} is a fully connected layer with parameters $\theta_2 \in \Theta$. The form of this equation has been proposed by Bachlechner et al. [2020] and is meant to allow faster convergence.

We note that, whilst it has been omitted from Fig. 3 for readability purposes, each CAE layer except for the last is followed by Layer Normalization [Ba et al., 2016] and the GELU [Hendrycks and Gimpel, 2016] activation function.

5.2 Optimizing the Model with Deep Reinforcement Learning

We optimize the model using Double Deep Q-Learning [Van Hasselt et al., 2016], which addresses stability and overestimation problems in Deep Q-Learning [Mnih et al., 2015]. In order to improve sample efficiency, we also utilize prioritized experience replay [Schaul et al., 2015], which samples from a replay buffer according to Eq. (2). Here, δ_i represents the Q-learning error of sample i, and ϵ is a small value to avoid division by zero. Importance weights $w_i = \frac{(N \cdot P(i))^{-\beta}}{\max_j w_j}$ are used to correct the non-uniform sampling procedure. Hyperparameters α and β regulate the sampling and correction strengths respectively.

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \ , \ \ p_i = |\delta_i| + \epsilon \ . \tag{2}$$

The full algorithmic pipeline of the training process is described in Algorithm 1.

Algorithm 1 Pipeline of the Training Approach

```
1:  **Input:** A distribution $\mathcal{D}$ over job allocation graphs, soft update rate parameter $\tau$, prioritized
    experience replay parameters $\alpha$ and $\beta$, batch size $b$, learning rate $\eta$.
2:  Initialize $Q_\theta$ and make copy $Q_{\text{target}} \leftarrow Q_\theta$.
3:  Initialize a *prioritized replay buffer* $\mathcal{B} \leftarrow \varnothing$.
4:  **for** each episode $t = 0, \ldots, T-1$ **do**
5:      Sample a graph $G_{\text{init}}(P \cup J, S \cup C) \sim \mathcal{D}$.
6:      Let $s \leftarrow G_{\text{init}}(P \cup J, S \cup C)$.
7:      **while** $S \neq \emptyset$ **do**
8:          Sample $a \leftarrow \{p \in P, j \in J\} \sim \text{softmax}_{\bar{a} \in S}(Q_\theta(s, \bar{a}))$.    ▷ Sample a job assignment.
9:          Receive $r \leftarrow 1$.
10:         Update $S' = S \setminus \{\{p, j\}\} \setminus \{\{p, j_i\} | (j, j_i) \in C\}$.    ▷ Remove conflicting edges.
11:         Update $s' \leftarrow s(P \cup J, S' \cup C)$.
12:         Store $(s, a, r, s', p)$ in $\mathcal{B}$.
13:         Set $\Delta = 0$.
14:         **for** $i = 1, \cdots, b$ **do**
15:             Sample $k \sim \frac{p_k^\alpha}{\sum_j p_j^\alpha}$.
16:             Compute $w_k = \frac{(N \cdot P(k))^{-\beta}}{\max_j w_j}$.
17:             Compute $\delta_k = r_k + \gamma Q_{\text{target}}(s'_k, \arg\max_{a'} Q_\theta(s'_k, a')) - Q_\theta(s_k, a_k)$.
18:             Update transition priority $p_k \leftarrow |\delta_i| + \epsilon$.
19:             Accumulate $\Delta \leftarrow \Delta + w_k \cdot \delta_k \cdot \nabla_\theta Q_\theta(s_k, a_k)$.
20:         **end for**
21:         Update $\theta \leftarrow \theta + \eta\Delta$.    ▷ Mini-batch Gradient Descent.
22:         Update $Q_{\text{target}} \leftarrow \tau Q_\theta + (1-\tau) Q_{\text{target}}$.
23:         $s \leftarrow s'$.
24:     **end while**
25: **end for**
```

6 Experiments

In this section, we evaluate the proposed approach of this report against baseline algorithms. First, we describe the training setup and datasets. Additional experiments on insights into the learned representations of the model can be found in Appendix B.

6.1 Training Setup

Our model: We use the model with 16 neurons in all hidden layers, and 8 in the output layer. Each hidden layer is equipped with Layer Normalization, followed by the GELU activation function. We stack 3 CAE modules, thus setting $K = 3$ in Fig. 3. We refer to this model as the *GNN*.

The specific hyperparameters during training can be found in Appendix A. The model is optimized for 200 episodes using the Adam optimizer with weight decay [Loshchilov and Hutter, 2017].

Baselines: We utilize three baseline algorithms to compare our performance. The first, which we refer to add *Greedy*, first selects the job with the minimum degree. Then, for that job, it chooses the person with the minimum degree connected to it. We also have a *Random* algorithm, which randomly selects assignments and an *Untrained GNN* with a random model initialization.

Datasets: We evaluate our model on a real-world hospital dataset (*Planny*) and two synthetically generated datasets. The latter two are random network datasets generated using the *Erdős–Rényi* and *Barabási–Albert* models. Some key statistics of these datasets can be found in Appendix A.

6.2 Results

As per the findings presented in Table 1, it becomes evident that the Greedy algorithm serves as a robust baseline across all datasets. However, the Graph Neural Network (GNN) demonstrates superior performance across each dataset, particularly for Erdős–Rényi instances. Nevertheless, for the Barabási–Albert dataset, it remains uncertain if the model has effectively acquired any meaningful knowledge, given that the minimum performance is already nearly optimal.

Table 1: Approximation ratios (higher is better; the best performance in bold) on different synthetic datasets and a real-world dataset (Planny). We report the average approximation ratios along with the standard deviation.

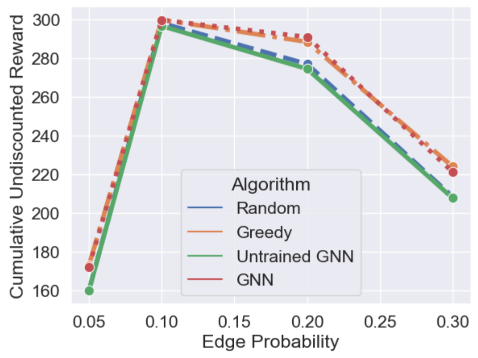
Method (↓) Dataset (→)	Planny	Erdős–Rényi	Barabási–Albert
GNN	$\mathbf{0.989 \pm 0.013}$	$\mathbf{0.981 \pm 0.010}$	0.999 ± 0.002
Greedy	0.987 ± 0.026	0.962 ± 0.014	$\mathbf{1.000}$
Random	0.969 ± 0.041	0.924 ± 0.021	0.996 ± 0.004
Untrained GNN	0.916 ± 0.088	0.915 ± 0.027	0.999 ± 0.005

To assess the generalizability of each model, out-of-distribution testing is conducted on the remaining datasets, and the corresponding results are provided in Table 2. The model trained on the Barabási–Albert dataset exhibits performance on par with a random model, suggesting that this dataset might not have provided the model with ample opportunities to learn a valuable representation, given that most assignments are already close to optimality. Interestingly, the other models achieve optimal scores on the Barabási–Albert dataset. Both the Planny GNN and the Erdős–Rényi GNN surpass the Greedy algorithm on the other datasets, despite not being trained on them.

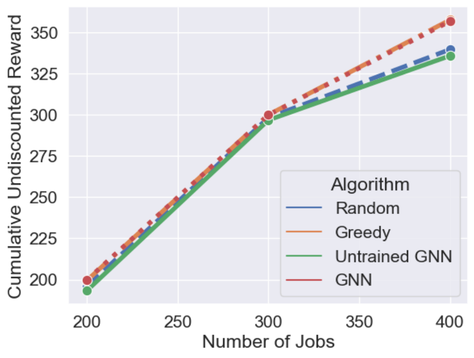
Table 2: Approximation ratios (higher is better; the best performance in bold) of out-of-distribution models. We report the average approximation ratios along with the standard deviation.

Method (↓) Dataset (→)	Planny	Erdős–Rényi	Barabási–Albert
Planny GNN	$\mathbf{0.989 \pm 0.013}$	0.970 ± 0.012	$\mathbf{1.000}$
Erdős–Rényi GNN	0.984 ± 0.023	$\mathbf{0.981 \pm 0.010}$	$\mathbf{1.000}$
Barabási–Albert GNN	0.968 ± 0.040	0.928 ± 0.024	0.999 ± 0.002

Finally, we conduct out-of-distribution experiments using datasets generated by Erdős–Rényi models, varying the numbers of jobs and densities of graphs. The experimental results are illustrated in Fig. 5. Our findings demonstrate that the performance of the GNN model is comparable to, and occasionally surpasses, that of the Greedy model. These outcomes suggest favorable scalability of the GNN model to larger and more diverse instances.



(a) Changing the probability of an edge with a fixed 300 jobs. It was trained on graphs with an edge probability of 0.10.



(b) Changing the number of jobs with a fixed edge probability of 0.10. It was trained on graphs with 300 jobs.

Figure 5: Out-of-distribution performance of the algorithms when tweaking the individual parameters of the Erdős–Rényi model.

7 Conclusion

In this report, we proposed a novel approach that combines Reinforcement Learning with Graph Neural Networks to tackle the Job Allocation Problem. We formulated the JAP as a Markov Decision Process and developed a model architecture that utilizes Graph Neural Networks to approximate Q-values. Our approach eliminates the need for manual annotation, which is often a major bottleneck in supervised learning approaches. Through experimental evaluations on real-world and synthetic datasets, we demonstrated that our proposed approach outperforms baseline algorithms, including a greedy algorithm and random models. The Graph Neural Network consistently achieves higher approximation ratios, showcasing its effectiveness in solving the JAP. Furthermore, the GNN model exhibited generalization capabilities, achieving competitive performance on out-of-distribution datasets. Overall, our findings highlight the potential of leveraging Reinforcement Learning and Graph Neural Networks for optimizing job allocation in complex scheduling problems. Future work can explore further improvements to the model architecture, as well as investigate its application in other real-world domains and more complex and constrained problem settings.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Thomas Bachlechner, Huanru Henry Majumder, Bodhisattwa Prasad Mao, Garrison W. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *arXiv*, 2020. URL `https://arxiv.org/abs/2003.04887`.

Cullen Bash and George Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. In *USENIX Annual Technical Conference*, volume 138, page 140, 2007.

Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7:441–499, 2004.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. In *IJCAI*, pages 4348–4355, 2021.

Imran Ali Chaudhry and Abid Ali Khan. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591, 2016.

Tianrui Chen, Xinruo Zhang, Minglei You, Gan Zheng, and Sangarapillai Lambotharan. A gnn-based supervised learning framework for resource allocation in wireless iot networks. *IEEE Internet of Things Journal*, 9(3):1712–1724, 2021.

Alberto Colorni, Marco Dorigo, Francesco Maffioli, Vittorio Maniezzo, GIOVANNI Righini, and Marco Trubian. Heuristics from nature for hard combinatorial optimization problems. *International Transactions in Operational Research*, 3(1):1–21, 1996.

Yves Crama. Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, 99(1):136–153, 1997.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Helga Ingimundardottir and Thomas Philip Runarsson. Supervised learning linear priority dispatch rules for job-shop scheduling. In *International conference on learning and intelligent optimization*, pages 263–277. Springer, 2011.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Behice Meltem Kayhan and Gokalp Yildiz. Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. *Journal of Intelligent Manufacturing*, pages 1–25, 2021.

Bikram Kumar, Lokesh Sharma, and Shih-Lin Wu. Job allocation schemes for mobile service robots in hospitals. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1323–1326. IEEE, 2018.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Luiz Antonio N Lorena and Marcelo G Narciso. Relaxation heuristics for a generalized assignment problem. *European Journal of Operational Research*, 91(3):600–610, 1996.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Mufti Mahmud, Mohammed Shamim Kaiser, Amir Hussain, and Stefano Vassanelli. Applications of deep learning and reinforcement learning to biological data. *IEEE transactions on neural networks and learning systems*, 29(6):2063–2079, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Mohammad Owliya, Mozafar Saadat, Rachid Anane, and Mahbod Goharian. A new agents-based model for dynamic job allocation in manufacturing shopfloors. *IEEE Systems Journal*, 6(2): 353–361, 2012.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

Satish Penmatsa and Anthony T Chronopoulos. Price-based user-optimal job allocation scheme for grid systems. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 8–pp. IEEE, 2006.

Michael L Pinedo. *Scheduling*, volume 29. Springer, 2012.

Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

John Turek, Joel L Wolf, and Philip S Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 323–332, 1992.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Vivek Vijayakumar, Fabio Sgarbossa, W Patrick Neumann, and Ahmad Sobhani. Framework for incorporating human factors into production and logistics systems. *International Journal of Production Research*, 60(2):402–419, 2022.

Zheyuan Wang and Matthew Gombolay. Learning scheduling policies for multi-robot coordination with graph attention networks. *IEEE Robotics and Automation Letters*, 5(3):4509–4516, 2020.

Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

Gal Yehuda, Moshe Gabel, and Assaf Schuster. It's not what machines can learn, it's what we cannot teach. In *International conference on machine learning*, pages 10831–10841. PMLR, 2020.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1621–1632, 2020.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to search for job shop scheduling via deep reinforcement learning, 2022.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Contents of the appendix

We describe the contents of the supplementary materials below:

- In Appendix A, we discuss the specific hyperparameters that were used for the training of the models. We also describe some of the key statistics of the datasets.
- In Appendix B, we discuss additional experiments performed with the framework regarding looking into the learned representation of the model.

A Experimental Details

For the experiments in Sec. 6, the hyperparameters described in Table 3 are used. Furthermore, the most important statistics of the datasets are described in Table 4.

Table 3: Hyperparameters employed in the training process.

Name	Value
Learning rate	0.001
Optimizer	AdamW
Hidden layer size	16
Output layer size	8
Number of GEM layers (K)	3
Batch size	2048
Number of episodes	200
ϵ (epsilon-greedy)	0.10
γ (discount factor)	1.0
θ (soft update rate)	0.025
Replay memory size	10^6
α (PER)	0.6
β (PER)	0.4

Table 4: Key statistics about the datasets.

Dataset Name	#Graphs	Mean #jobs	Mean #people	Mean #job conflicts	Mean #assignments	Mean density
Planny	20	507.2	25.2	14344.5	5777.9	0.079
Erdős–Rényi	20	300.0	15.0	8998.5	2998.75	0.152
Barabási–Albert	20	300.0	15.0	1782.0	3137.65	0.081

B Additional Experiments with Learned Model Representations

In order to gain insights into the results of the model after training, we look at two figures. Fig. 6 shows that the loss explodes during training. This is not surprising, as a reward of 1 at every step can lead to instability over training as many less terminal states are encountered. Since terminal states are the only states where the model can observe that the reward doesn't just always happens, it is expected that the Q-values can explode. In order to address this instability, Double Deep Q-Learning and Prioritized Experience Replay (PER) were utilized as described in Sec. 5. Resulting from the performance of the model in Sec. 6, it can be concluded that the model was still able to learn a good order of states with the learned Q-values, even though they explode.

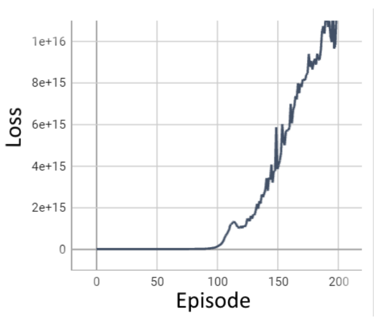


Figure 6: Exploding loss over the training of a GNN agent.

In order to confirm this idea, Fig. 7 plots the estimated Q-values of a trained GNN divided by the maximum estimated Q-value over one episode. Since the state graphs in the episode should get smaller at each step, it is expected that the Q-values should decrease. From Fig. 7, it becomes apparent that the model does learn some decrease in these Q-values, although it does not seem decreasing monotonically.

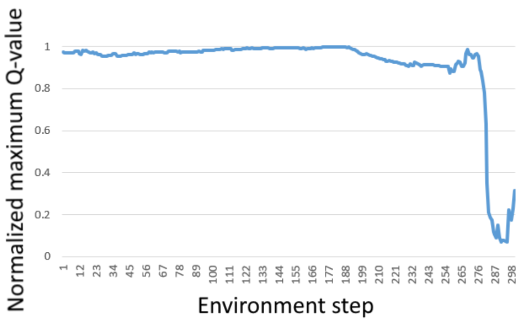


Figure 7: Maximum normalized estimated Q-values from a trained GNN agent over sequential timesteps of an episode.