Automated Personnel Scheduling with Reinforcement Learning and Graph Neural Networks

Benjamin Platten[1,2], Matthew Macfarlane[1], David Graus[2] and Sepideh Mesbah[2]

[1]*Universiteit van Amsterdam, Binnengasthuisstraat 9, 1012 ZA Amsterdam, The Netherlands*
[2]*Randstad Groep Nederland, Diemermere 25, 1112 TC Diemen, The Netherlands*

Abstract

This research investigates the extent to which a reinforcement learning agent can learn the heuristics of a combinatorial optimisation problem (CO), personnel scheduling, from a graph representation. In recent years, reinforcement learning has emerged as an effective data-driven approach to solving CO problems, a subset of mathematical optimisation that appear in many real-world tasks and can be costly to solve in terms of computing power and/or human resource. CO problems often have an underlying graph structure with relational inductive biases that can be exploited by powerful graph neural networks. Across a range of problem complexities, our approach was able return consistently high average reward, performing significantly better than baselines.

Keywords

Combinatorial optimisation, Reinforcement learning, Graph neural networks, Personnel scheduling

1. Introduction

Matching personnel with shifts in a way that ensures coverage, legality and employee well-being is a process that becomes exponentially more complicated as scale increases [1]. *Anonymized for review* is a human resource consulting firm responsible for scheduling over 40,000 employees into more than 70,000 shifts every week. Employees are matched with work from several different industries such as retail, healthcare, and catering. Staff shortages and increased demand have made this process even more challenging in recent years. Personnel scheduling (PS) as an operations research problem has generally been known as the Nurse Rostering Problem (NRP) [2]—a reference to the difficult task of assigning nurses to shifts in a 24-7 cycle—and has been studied since the 1960s by researchers from computer science, mathematics, operations research and medicine [2]. Scheduling problems carry hard and soft constraints; a typical hard constraint in NRPs is that there is sufficient staff coverage at all times. A soft constraint could be an employee's preference not to work on weekends [2]. For *anonymized for review*, personnel scheduling is characterised by high variance between clients (e.g. different retail companies) in terms of problem requirements; planning horizon, shift types, requisite skills and volume of staff and shifts vary significantly. For example, a single client could require as many as 2,000 shifts to be scheduled in one week, or as few as 2. The average is somewhere closer to 80 shifts per week per client. In this context, to help clients/planners to fill shifts in an efficient way there is a need for an automated technique that recommends personnel for given shifts by considering the constraints.

Researchers have developed linear models that find optimal solutions for scheduling problems by exploring the search space of possible solutions [2]. Known as exact methods, these algorithms simulate a decision tree where each node of the tree is a possible state that the scheduling problem can take and the solutions are situated at the leaves of the tree. Due to the enormous and complex search spaces of modern scheduling problems, exact methods alone are not viable. Furthermore, an 'optimal' solution is usually not the goal; administrators want to quickly generate a high quality schedule that satisfies all hard constraints and as many of a wide range of soft constraints as possible. The other broad class of solutions is heuristic methods which make use of higher level knowledge of a problem to take shortcuts through the search space. Reliance on domain expertise and hand-crafted features means that these methods are susceptible to changes in the problem formulation. Neither of these approaches generalise well to new problems [2].

Personnel scheduling can be framed as a combinatorial optimisation problem and there is a growing body of research using machine learning techniques to solve such problems [3]. CO is the process of searching for an optimal solution amongst a finite set of possibilities. Classic problems include the Travelling Salesman (TSP)—where the objective is to find the shortest route through a list of cities—and the knapsack problem, where the objective is to maximise the value of objects in a knapsack

RecSys in HR'22: The 2nd Workshop on Recommender Systems for Human Resources, in conjunction with the 16th ACM Conference on Recommender Systems, September 18–23, 2022, Seattle, USA.
platten.ben@gmail.com (B. Platten); m.v.macfarlane@uva.nl (M. Macfarlane); david.graus@randstadgroep.nl (D. Graus); sepideh.mesbah@randstadgroep.nl (S. Mesbah)

without violating a weight constraint [3]. If a CO problem is modelled as a sequential decision-making process, neural network function approximation can be used to learn features of a problem and use them to optimise the decisions of an agent [4]. This is the essence of Deep Reinforcement Learning (RL), a class of algorithms that have attained some notoriety in recent years due to eye-catching performance on various games [5]. Many CO problems naturally induce an underlying graph structure of nodes and edges [6]. By representing a problem using graph neural networks (GNNs), the learned vector representation encodes crucial structures that improve the ability of an algorithm to learn from such problems and solve them efficiently [7]. CO is at the heart of many real world problems; vehicle routing, microchip design, big data processing, and numerous forms of scheduling tasks, to name a few. This paper seeks to investigate whether a RL agent can learn to solve personnel scheduling problems represented as graphs.

Research question: To what extent can reinforcement learning optimise personnel scheduling problems?

There are many ways to design experiments to answer this research question. We chose to keep the RL algorithm and problem constraints fixed throughout the experiments and instead to focus on *problem complexity* and *reward function.* Problem complexity can be controlled by the number of shifts and also by the ratio between shifts and employees, both of which determine the probability that a correct action can be taken by chance. An increase in problem complexity will therefore lead to increased constraint violations from a random agent. A reward function determines whether an agent's actions are rewarded with a positive value or punished with a negative value. Reward is averaged over a set number of episodes (complete runs through a problem) as this provides a stable measure of an agent's performance over time.

In this context, we focus on the following *sub-research questions:*

1. To what extent does increasing problem complexity affect average reward?
2. To what extent does reward shaping affect average reward?

2. Related Work

Substantial research has been conducted into personnel scheduling and combinatorial optimisation more generally.

2.1. Personnel Scheduling Methods

The most common methods - exact, heuristic, and hybrid - have different ways of dealing with the complexity of scheduling problems. The complexity of a NRP is determined by the combination of constraints and parameters [1]. Usually, problems with a large number of possible shift types, a large number of nurses and/or a long planning period, are expected to require more computational effort. In their comprehensive review, Burke et al [2], declared that exact methods "cannot cope with the enormous search spaces that are represented by real problems (at least on their own)". However, innovative modelling techniques have been used to formulate integer linear programs (ILPs) with huge numbers of variables. Branch-and-price, a technique that augments linear relaxation - a method for solving hard ILPs by temporarily relaxing the integer constraints - by dropping a proportion of the constraints and then checking if the subsequent solution is feasible, was used by [8]. Modelling complex ILPs is challenging and powerful solvers are needed to run them [9].

The most common form of heuristic optimisation is meta-heuristics, which use strategies inspired by other systems to guide the search process. For example, Ant colony optimisation meta-heuristic algorithms build solutions by mimicking the foraging behaviour of ants [10]. Although effective and simpler to implement than exact methods, heuristics need to be revised if the problem statement changes slightly [2]. Furthermore, costly and error-prone feature engineering can introduce biases that do not align with the real world and lead to imprecise decisions.

Methods that hybridise exact methods with heuristic approaches are known as hybrid approaches and Burke et al. [2] went so far as to say that "some kind of (hybrid) heuristic method offers the only realistic way of tackling such difficult and challenging problems in the foreseeable future."

In general, the models seen in the literature are complex, narrowly applicable to a certain environment, and lack generality, but it is difficult draw conclusions when there is so much variance between studies. We can say that, by considering each instance of a problem in isolation, these existing methods do not exploit the fact that the problem instances often come from a common real world underlying distribution.

2.2. Machine Learning Methods for CO problems

For real world CO problems, it is highly likely that problem instances will share certain characteristics or patterns [6]. Data-dependent machine learning approaches can exploit these patterns, leading to the development of faster solutions for practical cases.

A neural network model was deployed on a CO problem as far back as 1996 when a Hopfield network was used by Mańdziuk [11] to solve instances of the Travelling Salesman Problem. Chen et al. [12] used neural networks to guide a tree search. By representing schedules as matrices, a network could analyse existing solutions and learn to predict the probability of each node leading to a solution. Václavík et al. [13] use classifiers to discard bad solutions and speed up a subsequent heuristic search.

Reinforcement learning (RL) refers to a computational approach to learning from interaction [9]. In RL, an agent interacts with a Markov Decision Process (MDP) with the goal of maximising some reward generated by the environment. A combination of a state of the environment and an available action, a state-action pair (s, a), has an expected return known as a state-action value function, or Q-value. A policy is a conditional distribution that the agent uses to select actions. The main aim of RL is to find an optimal policy - a conditional distribution used by the agent as a strategy to select actions - such that the state-action value function is optimised.

Broadly, there are two categories of RL algorithm; those which learn by optimising the value function to increase reward at each step and those which directly optimise the policy to increase total expected reward [14]. Policy-based methods directly model the agent's policy as a parametric function. By collecting previous decisions that the agent made in the environment we can optimise the parameters of the network to maximise the expected reward of the process [14].

$$\theta_{t+1} = \theta_t + \alpha R_t \frac{\nabla\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)}$$

An episode refers to all the states of an environment that come in between an initial-state and a terminal-state, generated from the sequence of actions taken and changes observed in the environment. An episode can be terminated due to success or failure, and, over many episodes, the parameters of a policy network are trained until they can achieve good performance on an unseen problem. Stronger convergence guarantees are available for policy-based methods than for value-based methods. One of the main strengths of RL is that agents can learn the rules of an environment without the need for explicit programming or examples of optimal endpoints. A key aim of RL research is to train models that make decisions to solve real problems [15]

Reinforcement Learning (RL) methods were first applied to CO by Zhang and Dietterich, 1995, with a model designed to solve the NP-Hard job-shop problem [16].

A growing body of research uses RL in combination with graph neural networks (GNNs), a type of function approximator designed to operate directly on a graph structure [17]. Whenever there is a set of entities that interact with one another or are related in any way, they can be expressed in the form of a graph [6]. GNNs are structurally similar to convolutional neural networks (CNNs) which are mostly used for image classification. CNNs get their name from the process of convolving the input vectors with a sliding kernel which reduces the size of an image whilst exploiting its spatial structure. Accounting for the underlying structure of the data introduces an inductive bias which promotes learning and leads to better results [7]. The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs for new data. GNNs also perform computations over input data in a way that exploits spatial structure, a process known as message passing. Node and edge feature vectors are propagated by an exchange of information (messages) between adjacent nodes.

Intuitively, it makes sense that a node might be represented by the average of the connections to its neighbours and in this way the spatial information of the graph is maintained. It also means that the number of message passing layers determines how far a signal can travel; with 2-layers, the message passing happens twice so the message travels two hops away. There are many GNN architectures optimised for different functions; node, edge, or graph classification, using features stored at the node, edge or both. The promise of GNNs is that the learned vector representation encodes crucial graph structures that help solve a CO problem more efficiently [6].

Mirhoseini et al. [18] developed a learning-based approach to chip placement, one of the most complex and time-consuming stages of the chip design process. Exploiting the graph structure of netlists (a description of the connectivity of an electronic circuit), Mirhoseini et al. [18] used a GNN to compute features of netlist nodes and create a rich representation for learning. Mirhoseini et al. [18] claim that their method is the first that can quickly generalise to previously unseen netlists and produce 'superhuman' results without the need for human experts in the loop. In the opinion of Cappart et al. [6], this study is only scratching the surface of "the innovations that can be enabled from a careful synergy of GNNs and CO".

Kool et al. [15] represented travelling salesman problems as graphs and used an encoder-decoder GNN as a parameterised policy to make decisions on which node to visit next. An encoder transforms input data into embeddings of a specific dimensions. The decoder then transforms the embeddings to the required output dimensions. The solution is constructed incrementally, adding one node at a time. The policy network is trained using the REINFORCE algorithm, a policy gradient method. The motivation for this choice was greater efficiency than a value function approach. Kool et al. [15] report significantly improved results over recent learned heuristic approaches to TSP. Their solution worked on problems with up to 100 nodes and generalised to 2 variants of the problem using the same hyper-parameters. They are clear that their goal is not to outperform specialised TSP algorithms, but to show that their approach is flexible enough to generalise to similar problems.

After a thorough search of the literature, there was no credible study found to be using an RL algorithm and graph representations for personnel scheduling. The research most relevant to our aims is Kool et al. [15] which has many useful insights.

3. Methodology

Our goal is to train an agent to sequentially assign employees to shifts in an acceptable configuration. A schedule is acceptable if the constraints are not violated. The foundation of this research is a pipeline that ingests a pool and a shift schedule and outputs a trained scheduling agent. We train models on a fixed set of problems in order to generate an agent that is capable of solving general, previously unseen problems. The pipeline was designed to take pool and shift schedule input of any size, adapting the parameters of the environment, graph representation, and Policy GNN accordingly.

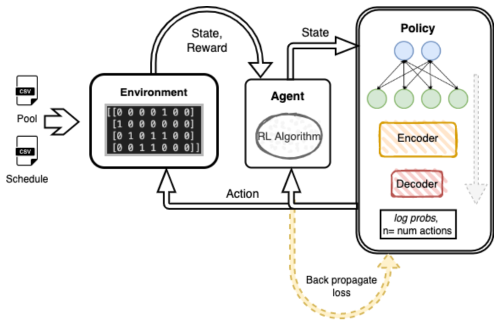


Figure 1: Pipeline

- A pool csv containing a list of employee ids, and a schedule csv containing a list of shift ids, are the initial input. The schedule csv also contains features for each shift: day of week and time of day (morning or evening).
- A custom OpenAI Gym RL environment takes the input csvs and combines them into a single matrix.
- The length of an episode is equal to the number of shifts in the schedule csv. At each step in an episode, the matrix is converted to a bipartite graph and passed through the GNN.
- The GNN outputs a policy; a probability distribution over the employee / action space.
- The agent takes actions according to the policy and updates the weights of the policy network based on the loss function.
- Repeat until a terminal state is reached: all shifts are assigned.

Shifts	Problem Count	Avg. Ratio	Min Ratio	Max Ratio
3	70	0.74	0.375	1.5
4	70	0.98	0.500	2.0
5	70	1.23	0.625	2.5
6	70	1.47	0.750	3.0
7	70	1.72	0.875	3.5
8	70	1.96	1.000	4.0

Table 1
Overview of the scheduling problems used for training the scheduling agent. 420 in total problems with 6 different shift counts and range of shift-employee ratios. The number of shifts, and the ratio between shifts and employees, determine the complexity of our scheduling problems: the probability that a correct action can be taken by chance.

3.1. Data

At *anonymized for review*, employees are grouped into pools which act as a pre-filter for eligibility; anyone assigned to a pool is eligible for shifts assigned to that pool. A pool represents an instance of a scheduling problem; the employees and shifts must be combined in a way that provides acceptable coverage and respects employment law and employee preference. Computational complexity increases with shift count and shift-employee ratio meaning that *anonymized for review* data includes many highly complex scheduling problems. Furthermore, the data contains many features with high cardinality, such as shift length and start time of a shift. To test the viability of an RL approach, simpler problem instances were required.

Inspired by the characteristics of *anonymized for review* data, we created synthetic schedule and pool data sets that can be combined to created simplified scheduling problems. The aim is to create problems that test an agent's ability to learn to satisfy a single constraint: the same employee can’t work sequential shifts. This constraint has to be learned from the features of the problems.

Synthetic schedules contain shift id and 2 features; day of week (Monday - Friday) and time of day (morning or evening). Schedule shift count ranges from 3 to 8 and pool employee count ranges from 2 to 8. By combining these schedules and pools, we created a training set of 420 unique problems. Full details of the training set can be seen in Table 1. Functions were created to manage the problem combinations and check for uniqueness / duplicate problems. Another function randomly generates new, unseen problems of variable shift count and shift-employee ratio, for testing.

3.2. Pipeline

So that a model can learn to find acceptable solutions to the scheduling problems, we define a Reinforcement Learning framework specific for this task, similar to Kool et al. [15]'s framework for tackling Travelling Salesman problems. The aim is to optimise the reward received for taking actions (assigning workers) at each step (shift). In this framework, an encoder-decoder graph neural network is used to estimate a probability distribution over the actions available at each state of a problem. The encoder produces embeddings of all input nodes, whilst the decoder outputs log probabilities for each available action. To train this policy model, we used a policy gradient method called REINFORCE.

3.2.1. REINFORCE

REINFORCE was proposed by Williams [19]. We start by initialising a random policy which the REINFORCE algorithm takes as input. The agent collects the trajectory of an episode from current policy, storing actions, states and rewards as a history. For each episode we calculate the reward and use it to evaluate the policy. For each action in the episode, the loss is calculated as the probability (according to the policy) of choosing the action, multiplied by the reward received. The loss is then back-propagated through the network to update the policy, increasing the probability that the agent will choose the actions that lead to higher reward in the next episode.

$$\theta_{t+1} = \theta_t + \alpha R_t \frac{\nabla \pi(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta)}$$

The encoder-decoder policy network is implemented using the Deep Graph Library.

3.2.2. Policy Network

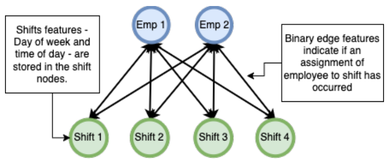

Shifts features - Day of week and time of day - are stored in the shift nodes.
Emp 1
Emp 2
Binary edge features indicate if an assignment of employee to shift has occurred
Shift 1
Shift 2
Shift 3
Shift 4

Figure 2: Example of a graph representation of a scheduling problem with 4 shifts and 2 employees. A bipartite graph has two distinct sets of nodes with edges connecting each node in one set with each node in the other. These graphs can be used as input to a graph neural network.

For the agent's policy, we implemented an encoder-decoder GNN architecture, similar to the model used by Kool et al. [15]. The building blocks of the GNN are from the Deep Graph Library, which supports a range of GNN architectures on top of the PyTorch framework and also provides graph building functions. An example scheduling problem graph representation can be seen in Figure 2.

The role of graph neural networks is to distill information about the type and connectivity of a node within a large graph into low-dimensional vector representations which can be used in downstream tasks. Our downstream task is to output log probabilities for each action in the action space.

The encoder produces embeddings of all input nodes. To do this, it takes a graph representation of the current state of a scheduling problem as input and uses message passing convolutional layers to update and aggregate the embeddings.

$$h_i^{l+1} = \sigma\left(\sum_{m \in M_i} g_m(h_i^{(l)}, h_j^{(l)})\right)$$

Then, at each time step, the decoder network calculates the inner product of the current shift's node features and the worker edge features, which serves as a compatibility score. The decoder calculate this score for problems of any size. The softmax output of these compatibility scores are the probabilities for the policy.

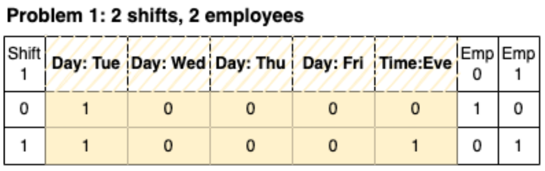
The number of nodes in the input graph reflects the count of shifts and employees in the scheduling problem. Shift features are consistent (5 days of week and 2 times of day) but the employee features - which indicates whether an employee as been assigned to a specific shift - vary with employee and shift count. The network needs to be able to process graphs of different sizes without being re-initialised with the parameters of the current graph, which would reset the learned weights. To achieve this, we designed the architecture to store the employee feature data in the edges, linear transformation of employee features could be performed at each step. This is illustrated in Figure 3. We use a message passing layer that incorporates edge data in it's update and aggregation functions. The design of this layer was proposed by [20] :

$$h_i^{(l+1)} = \sigma\left(\sum_{r \epsilon R} \sum_{j \epsilon N^r(i)} e_{j,i} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right)$$

where $N^r(i)$ is the neighbor set of node i w.r.t. relation r. $e_{j,i}$ is the normaliser. σ is an activation function. W_0 is the self-loop weight.

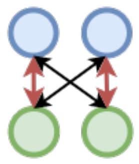
These edge features would allow us to measure important statistics such as total shifts allocated for each employee without any hard-coding with of problem size. These statistics could be used to add additional constraints such as minimum / maximum hours worked per employee.

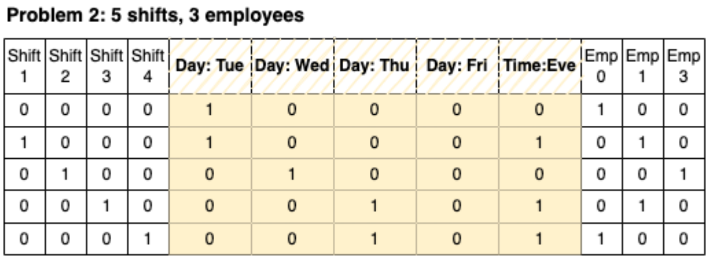
The network has a fixed learning rate of 1e-3. The encoder uses 2 convolutional layers, 5×32 shift embeddings and $n \times 32$ employee embeddings, where n = to the



Problem 1: 2 shifts, 2 employees

Shift 1	Day: Tue	Day: Wed	Day: Thu	Day: Fri	Time:Eve	Emp 0	Emp 1
0	1	0	0	0	0	1	0
1	1	0	0	0	1	0	1





Problem 2: 5 shifts, 3 employees

Shift 1	Shift 2	Shift 3	Shift 4	Day: Tue	Day: Wed	Day: Thu	Day: Fri	Time:Eve	Emp 0	Emp 1	Emp 3
0	0	0	0	1	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	1	0
0	1	0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0	1	0	1	0
0	0	0	1	0	0	1	0	1	1	0	0

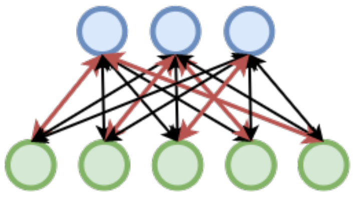


Figure 3: This figure shows 2 scheduling problems with different shift counts in matrix and graph form. By using a fixed number of shift features for each problem size (highlighted in yellow), the same linear transformation (5×32) can be applied to the feature data of different sized graph representations. The employee features, which indicate an assignment of employee to shift, are stored in the edges. A red edge indicates an assignment. The employee feature data is transformed to $n \times 32$, where n = to the number of employee nodes. Indexing starts at 0 (e.g. shift 0) and all features are one-hot encoded. With the exception of employee features, the first value for each feature is dropped to reduce collinearity, in accordance with standard practice.

number of employee nodes. The decoder uses 32×32 embeddings of the features for the current shift and 32×32 embeddings for each employee.

The process of repeating episodes to train the policy network is managed by a custom Open AI Gym environment.

3.2.3. Open AI Gym

OpenAI's Gym library is a popular choice for implementing environments for RL. It's main building block is a Python class, Env, which simulates the environment you want to train your agent in. There are many standard environments available, such as the Atari games used by DeepMind [21], and it supports custom environments. The key components of an environment are the state space, action space, reward function. For the state space of our custom scheduling environment, we combine a pool and a schedule into a matrix. The length of the matrix equals the number of shifts, and width of the matrix equals the number of shift features plus the number of employees in the pool. For each shift, the agent can assign an employee from the pool, therefore the action space is equal to the number of employees. The reward function is adjusted as part of the experimental setup but essentially it rewards the agent when an acceptable schedule is created. Or rather, it punishes the agent when constraints are violated. Another key part of the environment is the step function, which takes the agents action as input and updates the state and reward accordingly.

4. Experimental Setup

There are many possible variants for the design of this research, such as RL algorithm, training problem complexity, feature engineering, reward function and network architecture, to name a few. We have chosen to focus on reward function and problem complexity. Reward is based on performance with regards to a single constraint: an employee cannot work simultaneous or consecutive shifts. Given time constraints, hyper-parameter tuning was not performed for this study. The code can be viewed on github.

Average reward is used to evaluate a models performance. Reward is averaged across the previous 100 episodes to better capture an agent's long-term behavior. In a real world setting, the constraint we used is considered a hard constraint meaning that a violation renders the solution unacceptable. A solution could feasibly have a high reward but be unacceptable due to a single constraint violation. We used an additional metric, 'percentage acceptable' to measure the number of acceptable solutions produced by the agent: a solution with a maximum score and therefore no constraint violations.

4.1. Reward Functions

TSPs have an inbuilt cost, tour length (the cumulative distance between points), which the agent in Kool et al. [15] tries to minimise. There are examples of TSP solvers that apply cost (negative reward) incrementally (after each stop)—which has been shown to encourage greedy behaviour—and at terminal states (when the tour is complete) [15]. As there is no implicit reward or cost for scheduling problems, we chose an arbitrary total reward value of 1. Constraint violations results in a negative reward. Constraint violations are calculated based on the features in the schedule data: Assigning 2 shifts on the same day to an employee is a constraint violation. Applying shifts on subsequent days to the same employee is a constraint violation only if shift 1 is in the evening and shift 2 is in the morning. The cost of a constraint violation is 1/(number of shifts-1) because it is not possible to have a violation on the first assignment.

The effect of reward function on performance is tested by comparing the performance of 3 reward functions; 1) calculating reward at the terminal state of an episode (`Terminal`), 2) calculating reward at every step (`Step`), and 3) assigning half of the reward stepwise, and the remaining half at a terminal state as reward for an acceptable solution (`Step Bonus`).

4.2. Problem Complexity

Problem complexity can be increased in two ways; by the number of shifts and also by the ratio between shifts and employees. More shifts means more decisions for the agent to take, so there is more chance of a constraint violation occurring. The effect of shift-employee ratio is illustrated in 4, which shows a schedule and two pools. Combining the schedule with pool 1 has a higher ratio than for pool 2, which reduces the number of actions an agent can take without incurring a constraint violation.

Models were trained on 420 problems with between 3 and 8 shifts across 10,000 episodes. This means that the agent would see the same problem many times during training.

Models were tested on 500 problems from a range of complexity distributions, as can be seen in Table 2. The agent sees each problem once only.

The test problems have two levels of shift-employee ratio: an 'average' ratio of between 1.2 and 2.8, which reflects the first and third quartiles for shift-employee ratio in the *anonymized for review* data, and a 'high' ratio of between 2.9 and 5 to reflect the more complex problems seen in the data.

5. Results

Models were trained on 5 different seeds and each model was tested on every problem twice. The results were averaged across both runs for all 5 models. We tested the significance of our results using Welch's t-test, which doesn't assume equal variance between groups. Results are summarised in Table 3.

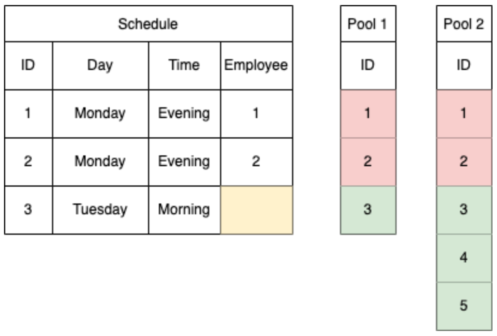


Figure 4: The effect of shift-employee ratio on action space. Green squares represent actions that will lead to an acceptable solution.

Shifts	Ratio Range	Problem Count
3-8	1.2-2.8	50
3-8	2.9-5	50
9-14	1.2-2.8	50
9-14	2.9-5	50
15-18	1.2-2.8	50
15-18	2.9-5	50
19-23	1.2-2.8	50
19-23	2.9-5	50
24-30	1.2-2.8	50
24-30	2.9-5	50

Table 2
Summary of test set problems showing size, shift-employee ratio and count of unique problems.

Across all experiments, the best performing model was the `Step Bonus` reward function (mean=0.89, standard deviation=0.22) which returned significantly higher average reward than `Random` baseline (m=0.49, sd=0.49), p<.05. `Step Bonus` also performed significantly better than the other reward functions `Step` (m=0.62, sd=0.62), p<.05, and `Terminal` (m=0.58, sd=0.57), p<.05.

In terms of problem complexity, average reward was significantly higher for Average ratio problems (m=0.74, sd=0.39) than for High ratio problems (m=0.65, sd=0.65), p<.05. This effect was observed for each model. The effect of increasing shift count is less clear. Excluding results from the `Random` agent, problems with a max shift count of 8 (ms8) (m=0.72, sd=0.36) show significantly higher average reward than problems with max shift counts of 14 (m=0.68, sd=0.68), 18 (m=0.68, sd=0.68) and 23 (m=0.68, sd=0.68). However, there was no significant difference in average reward observed between the smallest problems (ms8) and the largest problems (ms30) (m=0.72, sd=0.41). Furthermore, ms30 showed higher average reward than ms23, ms18, and ms14. These ef-

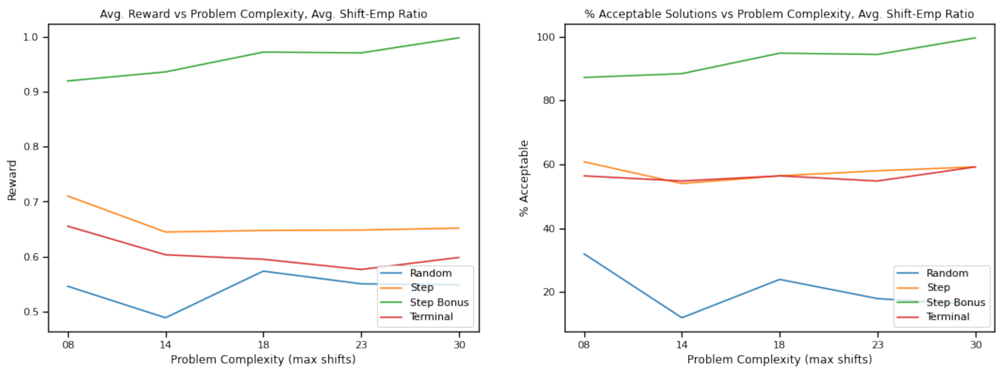


Figure 5: Performance on problems with Average Shift-Employee ratio. Performance on both metrics is stable across levels of Problem Complexity, even increasing slightly.

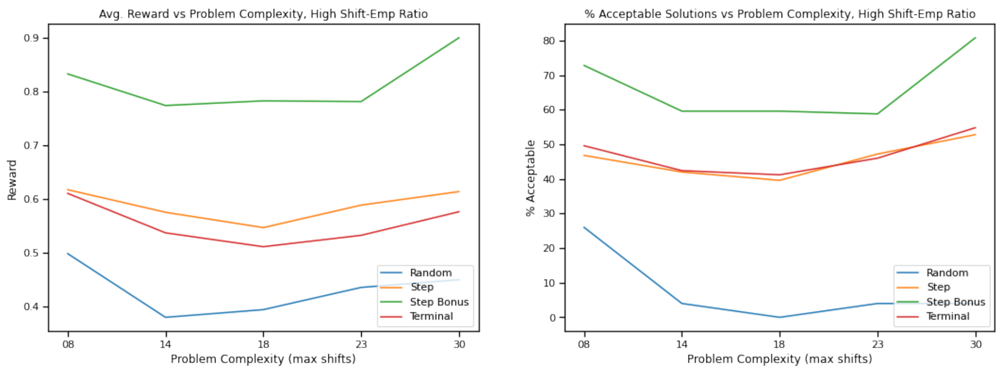


Figure 6: Performance on problems with High Shift-Employee ratio. Performance is less stable and significantly lower on high ratio problems.

fects were observed across both Average and High ratio problems.

Figure 5 shows the performance of the 3 reward functions against a `Random` agent baseline on average ratio scheduling problems of increasing shift count.

Figure 6 shows the performance of 3 reward functions against a `Random` agent baseline on high ratio scheduling problems of increasing shift count.

Model	Average Reward	% Acceptable
Random agent	0.49	14.00
Step	0.62	51.68
Terminal	0.58	51.56
Step Bonus	0.89	79.60

Table 3
Summary of results across all test problems by model (reward function). In a real world setting, a hard constraint violation invalidates a solution. % Acceptable indicates the percentage of solutions that had no constraint violations and earned a maximum reward of 1.

6. Discussion

Personnel scheduling is a real-world combinatorial optimisation task that requires significant time and expertise to manage effectively. In this paper, we developed a scheduling problem solver based on a framework from Kool et al. [15], which was designed to solve another CO problem, the travelling salesman problem. Compared to previous works, which used problem-specific approaches, this framework is an opportunity for a data-driven approach to personnel scheduling that could be of great practical value. We created a graph representation of each combination of shift schedule and employee pool for use in a GNN. The GNN functions as the policy, using valuable information about the node in the context of the graph to guide the decisions of an agent. The parameters of the network are trained using the REINFORCE algorithm. To measure the ability of the RL scheduler to optimise scheduling problems, we used average reward and percentage acceptable metrics. We compared performance across 3 different reward functions and 10 different levels of problem difficulty.

The Step Bonus model achieved consistently high performance across all problems, significantly better than the baseline and the other reward functions. Despite the fact that the problems in training set had a max shift count of 8, the trained Step Bonus agent was able to able achieve maximum score on 77% of High ratio, max shift 30 problems.

With a focus on an academic problem, rather than a real world application, [15] were able to test their model on several standard sets of travelling salesman problems (plus variants) and compare the performance to other studies that used the same data. There are standard test sets for personnel scheduling problems but they were out of scope for this research as these problems generally have more complicated constraints than the one we used. A comparison that can be made between this study and [15], is the effect of node count on performance. The cost for TSPs is positively correlated to the number of nodes, so performance appears worse (higher cost) for larger problems. However, [15] observed performance on 100 node TSP problems comparable to state of the art TSP solvers, suggesting that their graph model scaled well to larger problems. A similar case could be made for this research, as we saw no significant difference in performance between the hardest (ms30, High ratio) problems and the easiest (ms8, average ratio).

Because the size of the reward / cost for our system is negatively correlated to the number of shifts in a problem, violations are punished less harshly for big problems. However, the award of a bonus reward of .5 for an acceptable solution should counteract this.

Another explanation is that shift count and shift-employee ratio are not the best or only way to measure the complexity of a personnel scheduling problem. The consistently high rewards we observed for the Step Bonus model suggest that the network was able to learn the constraint - no simultaneous or consecutive employee assignments - and that it was no harder to exploit it for a large problem than a small one. Aside from the bonus reward, a decision made at time-step t doesn't effect $t + 1$. In a real world scheduling problem, this is unlikely to be the case as there will be additional constraints to consider. The most obvious constraint to add to this research would be one to ensure that each worker in the pool is assigned to at least one shift.

If, as expected, adding additional constraints increases complexity to an extent that performance is gets worse, there are architecture changes that could be made make the model more effective. The RL algorithm we have used, REINFORCE, is one of the most simple policy gradient methods. State of the art algorithms could be used, but a simple change worth testing is the use of a baseline, as seen in [15]: "A good baseline reduces gradient variance and therefore increases speed of learning". As Figure 7 shows, Step Bonus appears to converge on a stable reward after only a few hundred episodes, but this could well change if more constraints were added.

7. Conclusion

The results show that, with the right reward function, an RL agent was able to solve simplified personnel scheduling problems across a range of shift counts and shift-employee ratios. We have shown that an agent can learn the constraints of a combinatorial optimisation problem from data. For future work, additional evaluation using real world data is of importance to better understand the range of applicability of our approach.

References

[1] S. Den Hartog, et al., On the complexity of nurse scheduling problems, Master thesis (2016). URL: https://studenttheses.uu.nl/handle/20.500.12932/22268.

[2] E. K. Burke, P. De Causmaecker, G. V. Berghe, H. Van Landeghem, The state of the art of nurse rostering, Journal of scheduling 7 (2004) 441–499. URL: http://link.springer.com/10.1023/B:JOSH.0000046076.75950.0b.

[3] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European Journal of Operational Research 290 (2021) 405–421. URL: http://arxiv.org/abs/1811.06128.

[4] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, 5th International Conference on Learning Representations, ICLR (2017). URL: http://arxiv.org/abs/1611.09940.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (2015) 529–533. URL: http://www.nature.com/articles/nature14236.

[6] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, P. Veličković, Combinatorial optimization and reasoning with graph neural networks, Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21 (2021) 4348–4355. URL: https://doi.org/10.24963/ijcai.2021/595.

[7] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Ra-

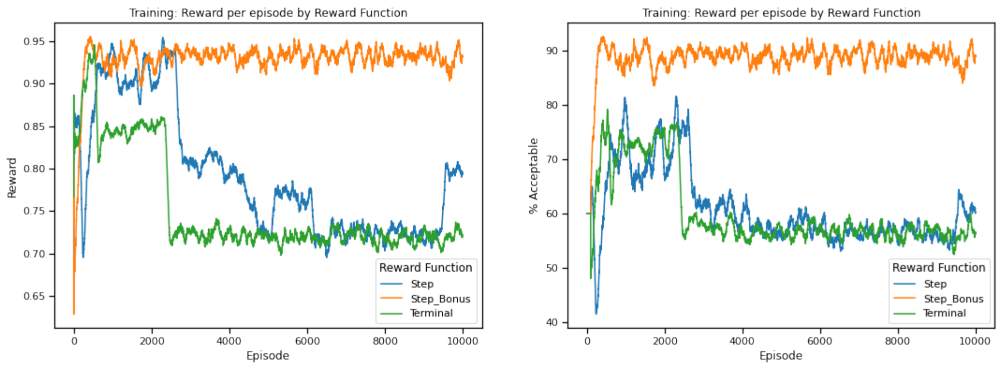


Figure 7: Performance on training data. The Step Bonus reward function converges after less than 1000 episodes.

poso, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, arXiv (2018). URL: https://arxiv.org/pdf/1806.01261.pdf.

[8] A. Legrain, J. Omer, S. Rosat, A rotation-based branch-and-price approach for the nurse scheduling problem, Mathematical Programming Computation 12 (2020) 417–450. URL: https://hal.archives-ouvertes.fr/hal-01545421.

[9] G. Koole, An Introduction to Business Analytics, Lulu. com, 2019.

[10] G. M. Jaradat, A. Al-Badareen, M. Ayob, M. Al-Smadi, I. Al-Marashdeh, M. Ash-Shuqran, E. Al-Odat, Hybrid elitist-ant system for nurse-rostering problem, Journal of King Saud University-Computer and Information Sciences 31 (2019) 378–384. URL: https://linkinghub.elsevier.com/retrieve/pii/S1319157818300363.

[11] J. Mańdziuk, Solving the travelling salesman problem with a hopfield-type neural network, Demonstratio Mathematica 29 (1996) 219–232.

[12] Z. Chen, P. De Causmaecker, Y. Dou, Deep neural networked assisted tree search for the personnel rostering problem, in: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT, volume 2, 2021.

[13] R. Václavík, P. Šcha, Z. Hanzálek, Roster evaluation based on classifiers for the nurse rostering problem, Journal of Heuristics 22 (2016) 667–697. URL: http://arxiv.org/abs/1804.05002.

[14] Y. Li, Deep reinforcement learning: An overview, arXiv preprint arXiv:1701.07274 (2017). URL: http://arxiv.org/abs/1701.07274.

[15] W. Kool, H. van Hoof, M. Welling, Attention, learn to solve routing problems!, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019. URL: https://openreview.net/forum?id=ByxBFsRqYm.

[16] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction (2018) 352.

[17] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE transactions on neural networks 20 (2008) 61–80.

[18] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, et al., Chip placement with deep reinforcement learning, arXiv preprint arXiv:2004.10746 (2020). URL: http://arxiv.org/abs/2004.10746.

[19] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine learning 8 (1992) 229–256.

[20] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European semantic web conference, Springer, 2018, pp. 593–607. URL: http://arxiv.org/abs/1703.06103.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (2015) 529–533. URL: http://www.nature.com/articles/nature14236.