# Learning-based Scheduling

by

Siddharth Nagar Nayak

B.Tech., Indian Institute of Technology Madras (2020)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022



Author .....
Department of Aeronautics and Astronautics
May 17, 2022

Certified by .....
Hamsa Balakrishnan
William E. Leonhard (1940) Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notion herein.

**Learning-based Scheduling**

by

Siddharth Nagar Nayak

Submitted to the Department of Aeronautics and Astronautics
on May 17, 2022, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

## Abstract

Integer programs provide a powerful abstraction for representing a wide range of real-world scheduling problems. Despite their ability to model general scheduling problems, solving large-scale integer programs (IP) remains a computational challenge in practice. The incorporation of more complex objectives such as robustness to disruptions further exacerbates the computational challenge. With the advent of deep learning in solving various hard problems, this thesis aims to tackle different computationally intensive aspects of scheduling with learning-based methods. First, we apply reinforcement learning (RL) to the Air Force crew-scheduling problem and compare it against IP formulations which explicitly optimize for minimization of overqualification and maximization of training requirements completed. We show that the RL agent is equally effective as its IP counterpart when the reward function is engineered according to the objective we want to optimize. We also show that the RL formulation is able to optimize for multiple objectives with simple modifications to the reward structure, whereas the IP methods require separate formulations for their objective functions. Then we present Neural network IP Coefficient Extraction (NICE), a novel technique that combines reinforcement learning and integer programming to tackle the problem of robust scheduling. More specifically, NICE uses reinforcement learning to approximately represent complex objectives in an integer programming formulation. We use NICE to determine assignments of pilots to a flight crew schedule so as to reduce the impact of disruptions. We compare NICE with (1) a baseline integer programming formulation that produces a feasible crew schedule, and (2) a robust integer programming formulation that explicitly tries to minimize the impact of disruptions. Our experiments show that NICE produces schedules that are more robust to disruptions than the baseline formulation, with computation times that are lower than those of the corresponding robust integer program.

Thesis Supervisor: Hamsa Balakrishnan
Title: William E. Leonhard (1940) Professor of Aeronautics and Astronautics

# Acknowledgments

I would like to acknowledge a number of people who have shaped my experience in graduate school over the last two years.

I am indebted to Prof. Hamsa Balakrishnan, my advisor and would like to thank her. She is the most supportive advisor one could ever hope for – she has always encouraged me in every aspect of my grad school life, and went above and beyond to support me at the times that I really needed it. She has been very accommodating when I ventured into research areas that were probably not the lab's primary focus at the time, and has been extremely supportive of my collaborations with other researchers over the last two years. She has always advocated for a healthy work-life balance for her students and has fostered an extremely friendly and collaborative culture within her research group.

I would like to thank Luke Kenworthy, Christopher Chin, Matthew Koch and Travis Smith with whom I worked closely on the USAF-MIT AI Accelerator project. Christopher and Matthew helped me in getting up to speed with the Air Force scheduler terminologies. I thoroughly enjoyed discussing the different learning-based formulations for the project with Luke and Travis. Part of this thesis would not have been possible without Luke. I would also like to thank the other members of the AI Accelerator team including, Michael Snyder, Kendrick Cancio, Jessamyn Liu, Amy Alexander, Ronisha Carter and Allison Chang.

I enjoyed working with Karthik Gopalakrishnan, Victor Qin and Geoffrey Ding on the multi-agent navigation problem (and I look forward to continue working with them through my PhD). I would like to thank Karthik for encouraging me to explore problems in this field for my PhD, mentoring me and being a patient collaborator to bounce ideas off of.

I have been fortunate to collaborate with MEng students Simran Pabla and Carson Smith and UROP Akila Saravanan on their MEng thesis and UROP projects respectively. Working with them has been quite fun and broadened my research perspectives towards different applications.

I would like to express my gratitude to Prof. Balaraman Ravindran (IIT Madras, India), Dr. Harshad Khadilkar, Prof. Abhishek Sinha (IIT Madras, India) who mentored me during my undergraduate studies, and helped me find my feet in research.

I would also like to thank everyone I collaborated with on the problem sets and the projects for the courses I took here at MIT. Their help made working on the psets more fun (and easy). I would also like to thank my friends for being really good Zoom buddies (and in-person) throughout the pandemic. Special thanks to Rebecca Jiang, Adwait Kulkarni, my aunt and her family for helping me out in setting myself up after moving to Cambridge and taking care of me after my ACL injury. I am also grateful to the MIT Cricket Club and the MIT Korean Karate Club and the members of these two clubs for letting me continue with cricket and learn martial arts respectively.

I would also like to thank the MIT SuperCloud [40] for providing high performance computing resources that have contributed to the research results reported within this thesis.

This section would be incomplete if I did not acknowledge projects that I was enthusiastic about, but did not work out. These projects taught me a lot about the research problems I was working on and that helped me in shaping my current research interests.

Finally, I couldn't be where I am today without the unconditional support of my parents and my sister. They have cheered me through every accomplishment and setback, acted as my sounding board and reassured me with their boundless love.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Air Force Crew scheduling involves assigning pilots to events according to the pilot qualifications, the event requirements, the pilot requirements, etc. Due to these complex constraints, crew-scheduling is a challenging combinatorial optimisation problem. The creation of a typical schedule can take 3 human schedulers working 9 hours/day, a total of 27 hours. Thus, there is a scope to improve the scheduling process by automating it so that these human schedulers who are typically airmen from the squadrons, can focus on other important duties.

Furthermore, even the most experienced schedulers can run into problems with scheduling due to unforeseen disruptions in flights (missions popping-up/dropping randomly), pilot availability (emergency leaves). This can cause a variety of issues for the schedulers where they have to make major changes in the schedules or even completely discard the previously created schedules to create new ones. This issue has been alleviated with recent work by Chin [10] and Koch [25] where they introduce various integer programming (IP) formulations to optimize for different metrics and also account for disruptions for robust scheduling. Although, these IP formulations can be used to explicitly optimize for certain metrics, they need to be done separately for each metric the schedulers might care about. In essence, it is hard to combine

multiple objective functions in a single optimization formulation. Along with this, the robust scheduling IP can sometimes fail or take a lot of time to produce schedules due to its complexity in a heavily disrupted scenario. To tackle these issues, this thesis aims to use reinforcement learning to account for multiple objective functions with a single scheduling agent. Also, this thesis introduces a hybrid approach for the robust scheduling problem which is able to produce more robust schedules with significantly lesser computation times.

## 1.2 Project Background

With the need of improving and simplifying the scheduling process, the Air Force has been developing *Puckboard* which is a web-based software application to assist human schedulers. Traditionally, the schedulers have used pucks on whiteboards as a representation of pilot assignments whilst creating schedules; and hence the name *Puckboard*). The goal of Puckboard is not to replace the human schedulers but to assist them. The advantage of Puckboard is that the schedulers do not have to cross-reference multiple whiteboards when creating schedules. Currently, Puckboard is used to automatically generate pilot assignments for flights. Puckboard accounts for the availability of the pilots, the qualification levels of the pilots, etc. when creating schedules. However, the human schedulers note that it is more time-consuming to clean the schedules created using Puckbboard than to manually assign pilots to flights. Prior work by Chin [10] and Koch [25] has improved the Puckboard functionality by including specialized integer programming formulations to optimize for feasibility of schedules, minimization of overqualification, maximization of training requirements completed and accounting for robustness towards disruptions in the schedules.

## 1.3 Related Work

### 1.3.1 Integer Programming Methods for Scheduling

Personnel scheduling has been a long-standing challenge in Operations Research, and has been the focus of much research over the past several decades [14, 49]. Integer programs (IPs) and mixed integer programs (MIPs)[1] have been widely-used for personnel scheduling, in large part due to their ability to represent general scheduling problems. However, despite the power of IPs to model scheduling problems, solving large-scale IPs in practice is often computationally challenging [38]. Most real-world applications also need robust schedules, namely, schedules that do not require considerable adjustments to personnel assignments in the event of an unforeseen disruption. While robust MIP-based formulations of scheduling problems can be developed, they are usually at least as computationally challenging as their non-robust counterparts [50, 52, 13, 5].

The scheduling of flight crews (e.g., pilots) is a personnel scheduling problem that arises in the context of aviation [19, 8, 21, 60]. Similar to other scheduling problems, crew scheduling has traditionally been tackled using large-scale IPs, both for airline and military flight crews [26, 47]. Although there are quite a lot of similarities in the military crew scheduling problem and the airline crew scheduling problem, there aren't a lot of research works focused solely on the military crew scheduling. Both have to consider pilot/crew qualifications, training requirements of the pilots/crew, rest requirements and leave days of the pilots/crew.

In [21], the author constructs an integer program for joint flight scheduling and crew scheduling. The crew assignments are made by maximizing the total "rewards" across all pilots, where the reward is associated with the number of training requirements completed. This work considers three qualification levels (instructors, leads, students) whereas, in our case, Puckboard has around 32 qualification levels along with more granularized qualification levels like night and air-drop qualifications. Sim-

[1] We use the terms integer program (IP) and integer linear program (ILP) interchangeably unless noted otherwise; similarly for mixed-integer program (MIP) and mixed-integer linear program (MILP).

ilarly, in [31], the author maximizes the number of scheduled flights whilst considering the constraints but does not consider details in departure and arrival times.

### 1.3.2 Learning-Based Approaches to Scheduling

Successes in deep (reinforcement) learning have motivated research that focuses on obtaining end-to-end solutions to combinatorial optimization problems; e.g., the travelling salesman problem [37, 30, 59, 56, 28] or the satisfiability problem (SAT) [2, 58]. RL has also been used for resource management and scheduling in a diverse set of real-world applications. For example, Gomes [15] uses an asynchronous variation of the actor-critic method (A3C) [34] to minimize the waiting times of patients at healthcare clinics. The lack of readily available optimization methods for this problem due to the ad-hoc nature of patient appointment scheduling motivates the usage of RL for this application to model the uncertainty. Mao et al. [32] introduce DeepRM which uses resource occupancy status in the form of images as the states and uses neural networks for training the RL agent. Chen et al. [9] improve upon DeepRM by modifying the state-space, reward structure and the network used in the DeepRM paper. Chinchali et al. [11] use RL for cellular network traffic scheduling. They incorporate a history of the states observed to re-cast the problem as a Markov Decision Process (MDP) [39] from a non-Markovian setting. Along with this modification, they construct a reward function that can be modified according to user preferences. In all of the works above, the models used application-specific state space, action space, and reward structures to optimize special-purpose objective functions to create schedules.

In recent work, Nair et al. [36] use a bipartite graph representation of a MIP and leverage graph neural networks [42, 24] to train a generative model over assignments of the MIP's integer variables. We do not use generative models to solve an IP formulation, but instead use RL to formulate the IP itself. Very recent work by Ichnowski et al. [20] uses RL to speed up the convergence rates in quadratic optimization problems by tuning the inner parameters of the solver.

### 1.3.3 Robust Scheduling

Flight delays are the main cause of disruption to crew schedules in commercial aviation; buffers (or slack in the schedule) have therefore been considered as a mechanism to achieve schedule stability amidst flight delays [55, 7]. A *buffer* refers to the amount of time between the successive flights flown by a particular pilot. By increasing these buffers, a new pilot assignment is less likely to be needed due the initially-planned pilot being delayed, and therefore being unable to make the flight. In [47], the author builds on top of [31] to include departure and arrival times.

## 1.4 Outline

In Chapter 2, we present a reinforcement learning based formulation for the Air-Force crew scheduling problem. We also discuss the effect of the reward structure on the performance of the reinforcement learning agent and compare metrics like overqualification and training requirements completed. In Chapter 3, we combine reinforcement learning and integer programming to introduce a hybrid approach called NICE. We show that this approach can be used to tackle the problem of robust scheduling. The work presented in this chapter is largely based on research that was published in [22]. We finally end with conclusions and future work in Chapter 4. The code for reproducing our experiments with the RL formulation and the NICE approach can be found at https://github.com/nsidn98/pilotRLNew and https://github.com/nsidn98/NICE respectively.

# Chapter 2

# Reinforcement Learning for Scheduling

Deep reinforcement learning (RL) has proved to be very successful in achieving super-human performances in a wide variety of tasks like playing Go [45], Chess [46], Atari-video games [35, 33], high-dimensional robot control [29] and solving physics-based control problems [17]. There have been many works which have used reinforcement learning for obtaininig end-to-end solutions for combinatorial optimization problems like the travelling salesman problem [37, 30, 59, 56, 28], satisfiability problem (SAT) [2, 58], scheduling [15, 32, 9, 11]. All of the works mentioned above require application-specific state space, action space and reward structure to optimize for special-purpose objective functions. Similarly, we introduce a reinforcement learning formulation for obtaining schedules for the Air-Force Crew Scheduling problem. Our reinforcement learning formulation has an easily understandable reward structure to mimic objective functions used in similar integer programming (IP) formulations. We show that the RL agent [1] is able to perform as well as the IP formulations with respect to metrics like feasibility of schedules, minimizing overqualification and completion of training requirements. The advantage of the RL formulation is that multiple metrics can be combined in the reward function to optimize with a single agent whereas the IP formulation require separate objective functions which in turn create different

[1]The code is available at https://github.com/nsidn98/pilotRLNew

schedules when optimizing for different metrics.

We begin this chapter with some background on reinforcement learning where we introduce the reinforcement learning notation and a reinforcement learning algorithm-Proximal Policy Optimisation in Section 2.1. In Section 2.2 we introduce the Markov Decision Process for the crew scheduling problem, the environment we use for simulating the crew scheduling problem along with the different reward structures used for optimizing different metrics. This is followed by the experiments and the results in Section 2.3. And finally the chapter ends with the discussion of the results, conclusions and future work in Section 2.4 and 2.5.

## 2.1 Background

Below, we give a brief summary on the notation used in context of reinforcement learning (RL) and an on-policy RL algorithm: Proximal Policy Optimization (PPO) [43].

### 2.1.1 Reinforcement Learning Notation

Reinforcement learning tries to solve sequential decision problems by learning from trial and error. An RL setting consists of a Markov Decision Process which is based on the Markov assumption of "given the present, the future does not depend on the past". A standard MDP consists of a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}_a, \mathcal{P}_a \rangle$. The set of states or state space is denoted by $\mathcal{S}$, the set of actions or the action space is denoted by $\mathcal{A}$, the immediate reward received after transitioning from state $s$ to $s'$ due to action $a$ is denoted by $\mathcal{R}_a(s, s')$. The dynamics of the MDP is modelled by the transition probability function $\mathcal{P}_a(s, s')$ which denotes the probability that action $a$ in state $s$ will lead to state $s'$. In practice, an RL agent interacts with an environment $\mathcal{E}$ which has the MDP tuple mentioned above embedded in it. At each time step $t$, the agent receives a state $s \in \mathcal{S}$ and selects an action $a \in \mathcal{A}$ according to its policy $\pi$. After the action, the agent observes a scalar reward $r = \mathcal{R}_a(s, s')$ and receives the next state $s'$. The goal of the agent is to choose actions to maximize the cumulative sum of

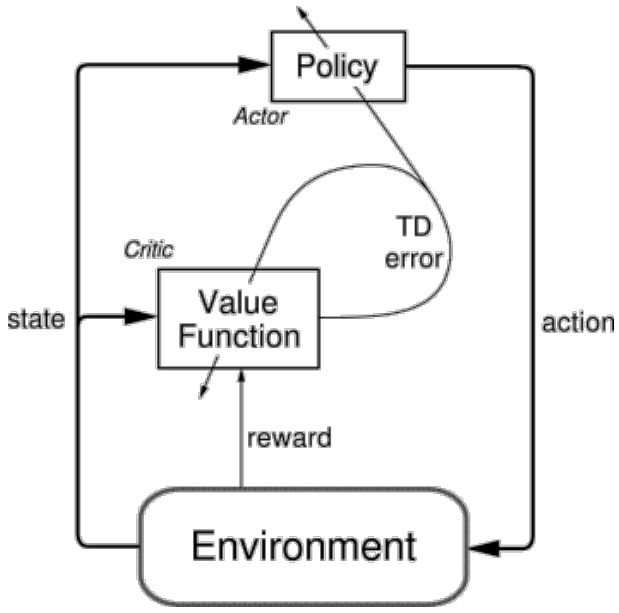


Figure 2-1: Actor Critic: The actor (policy) takes in the state as the input and gives out the action to take. The critic takes in the state and the action suggested by the critic to evaluate the actor. (Image taken from [48])

rewards over time. In other words, the action selection implicitly considers the future rewards. The discounted return is defined as $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where $\gamma \in [0, 1]$ is a discount factor that trades-off the importance of recent and future rewards.

Reinforcement Learning algorithms can be divided into two main sub-classes: Value-based and Policy-based methods. In value-based methods, values are assigned to states by calculating an expected cumulative score of the current state. Thus, the states which get more rewards, get higher values. In policy-based methods, the goal is to learn a map from the states to actions, which can be stochastic as well as deterministic. A class of algorithms called actor-critic methods [27] lie in the intersection of value-based methods and policy-based methods, where the critic learns a value function and the actor updates the policy in a direction suggested by the critic.

### 2.1.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [43] is an actor-critic method used to train RL agents. The motivation of PPO is "how can one take the biggest possible improvement

step on a policy using the data the agent currently has access to, without stepping too far away to cause a drop in the performance". In essence, PPO ensures that a new update of the current policy does not change it too much from the previous policy. This leads to less variance in training at the cost of some bias, but ensures smoother training and also makes sure the agent does not go down an unrecoverable path of taking unreasonable actions. PPO uses a clipped surrogate objective function which is a first order trust region approximation. The purpose of the clipped surrogate objective is to stabilize training via constraining the policy changes at each step. PPO updates policies as:

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{2.1}$$

where $\theta_k$ are the parameters of the policy $\pi$ at the $k$th iteration and is usually parametrized with neural networks. Each update typically includes taking multiple steps of stochastic gradient descent (SGD) [23, 41] to maximize the objective. Here $L$ is given by:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\ \ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s, a)\right) \tag{2.2}$$

where $\epsilon$ is a hyperparameter which signifies how far away the new policy is allowed to go from the old. The above objective function can be simplified to (the proof can be viewed in Appendix A):

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right) \tag{2.3}$$

where,

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

When the advantage $A$ is positive, its contribution to the objective can be written as:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s, a) \tag{2.4}$$

Because the advantage is positive, the objective will increase if the action becomes more likely. But the min in the equation puts a limit to how much the objective can increase. Once $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$, the min in the equation limits this term to hit a ceiling of $(1+\epsilon)A^{\pi_{\theta_k}}(s,a)$. Thus, the new policy does not benefit by going far away from the old policy.

When the advantage $A$ is negative, its contribution can be written as:

$$L(s,a,\theta_k,\theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s,a) \tag{2.5}$$

Because the advantage is negative, the objective will increase if the action becomes less likely. But the max in the equation limits it to how much the objective can increase. Once $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$, the max in the equation limits this term to hit a ceiling of $(1-\epsilon)A^{\pi_{\theta_k}}(s,a)$. Thus, again the new policy does not benefit by going far away from the old policy.

## 2.2 Markov Decision Process for Crew Scheduling

### 2.2.1 Reinforcement Learning Formulation

In this section, we introduce the task of building schedules for a version of the flight crew scheduling problem, hereafter referred to as the "crew scheduling problem," which has a long history in both commercial and military aviation [3, 16, 12]. We consider the scenario in which we are given a collection of flights that must be flown by a given squadron (i.e., a group) of pilots. Every flight has multiple slots, each of which must be filled by a different pilot. Each slot has qualification requirements that must be satisfied by any pilot who is assigned to that slot. Depending on their qualification, a pilot would only be eligible to fill a subset of slots. Finally, every pilot has some specified availability. We discretize our schedule into days, although other time discretizations could be used. In other words, a feasible schedule assigns pilots to slots such that every flight in the schedule horizon is fully covered, and the qualification requirements for the slots and availability restrictions of the pilots are

satisfied.

Since this work was carried out in collaboration with the MIT-US Air Force AI Accelerator, our problem formulation focuses on the constraints and preferences for US Air Force C-17 Squadrons. Consequently, our formulation differs from some of the standard crew scheduling formulations in prior literature, which have been largely in the context of airline flight crews. For example, all of the squadron's flights start and end at the same place, so we do not factor in crew relocation. Also, in the data we received, the start and end dates of the flights included the required crew rest time, so we did not need to explicitly model this. However, flying squadrons have more granular pilot qualification levels than what have been considered in airline crew scheduling.

We model the building of a valid schedule with a discrete event simulation (DES): at each time step, we take an action, which affects the state of our system. The main idea of our reinforcement learning scheduling approach is to order the slots that need to be scheduled and, at each slot, pick a pilot to assign to that slot. If we get through all of the slots, we end up with a complete schedule, though filling all of the slots is not a guarantee; the RL scheduling agent could back itself into a corner, leaving no pilots to assign to a given slot based on its previous decisions. We use Proximal Policy Optimization (PPO) [43], which is an actor-critic method where the actor chooses the action for the agent and the critic estimates the value function. The actor network gives a probability distribution over the pilots to choose given the state input and the action is chosen by sampling from this distribution.

### 2.2.2 Environment

To train our RL agent, we create an OpenAI Gym [6] type environment for the crew scheduling problem. We utilize an anonymized dataset from a US Air Force squadron to construct a random event generator. The dataset contains 87 pilots with 32 different qualifications and 801 flights across over six months, each containing between 2–3 slots. There are 16 different types of flights, where the type determines the qualification requirements for the slots on the flight. These flights are subdivided

into two categories: missions and simulators, which we treat equivalently except for the purposes of our random event generation. There are 7 mission types and 9 simulator types. We train our RL agent on randomly generated flights based on this dataset. More information about the dataset can be found in Appendix B.

### 2.2.3 Random Event Generation

To simulate the environment in our OpenAI Gym type environment, we generate flights randomly by modelling the randomness according to the dataset. To generate the random flights, we first divid our dataset into simulators and missions that started in 26 different full-week intervals. For each week, we create $\alpha$ random mission-based flights, where $\alpha$ is drawn from a normal distribution with a mean and standard deviation equal to the mean and standard deviation of missions across all 26 weeks. Similarly, we also create $\beta$ random simulator-based flights, where $\beta$ is drawn from a similar distribution that uses simulators instead of missions. The dataset also contains a variety of training requirements that each pilot, ideally, would fulfill; the number of times the pilot should fulfill each requirement; and information about which flight satisfied which training requirements. Along with the training requirement information, each flight contains two binary training requirement qualifiers (TRQs) to help determine which training requirements the flight fulfills.

For each flight generated, we pick a random day in the scheduling week for it to start. We then randomly pick a type for the flight, where the probability of picking that type of mission was proportional to the number of times it showed up in the dataset. To determine the length of duration for the flight, we randomly sample a flight length from a flight of the selected type from the dataset. We follow an identical process to generate each simulator, except each simulator started and ended on the same day, so we did not randomly pick a length. The dataset provides the dates each pilot is on leave, which we used directly. Then, we created a fixed ordering of slots that need to be scheduled. We order the slots first by the corresponding flight's start date and use the flight's arbitrary unique ID as a tie-breaker. We order slots of the same flight by ascending qualification. The assignment of a pilot to a slot serves as a

time-step in our DES.

We define the MDP constructed for the Crew Scheduling problem in the next section.

### 2.2.4 State Space

The state space for our RL agent includes:

1. A binary vector for the pilots available for the current slot
2. A flattened vector encoding the current event to be scheduled, consisting of:
    (a) A one-hot encoding of the event type
    (b) A binary vector indicating whether each training requirement was true or false
    (c) A binary vector representing the pilots assigned to the current event (Each event may have 2-3 pilot slots)
    (d) The event duration (in days)
    (e) The number of days between the start of the scheduling episode and the start of the event
    (f) The number of days between the start of the scheduling episode and the end of the event
3. A vector containing the total number of training requirement fulfillments each pilot could receive for flying that event, if it were flown a sufficient number of times

Note that we do not use the training requirement information outside of the state space formulation for our RL scheduler. In our experiments, we found that including the training requirements from our dataset in the state space helped our RL agent perform better. We suspect that they helped our neural network better reason about trade-offs when selecting a particular pilot for a slot.

### 2.2.5 Action Space

The RL agent has to select a pilot from the given set of pilots for any given state. This technique, previously used by Washneck et al. [53] for production scheduling, gives us an action space size equal to the number of pilots in the squadron.

### 2.2.6 Reward Function

There are a lot of different objective functions the schedulers care about while creating these schedules. Accordingly, we construct reward functions which mimic those objectives.

#### Feasibility

According to the user studies conducted by the Puckboard team [10, 25], the number one priority of the schedulers is to create a valid feasible schedule. Hence, we give a score of +1 for placing a valid pilot to slot and penalize the agent by giving a score of -10 if the agent does not place a valid pilot to the slot or it fails to complete the assignment of pilots to all the slots in the current episode. At the end of the episode, we give a score of +10 if it is successful in completing all the assignments in the episode.

$$r_{place} = \begin{cases} +1, & \text{valid pilot placed} \\ -10, & \text{invalid pilot placed or} \\ & \text{incomplete assignments in episode} \\ +10, & \text{complete assignments in episode} \end{cases} \tag{2.6}$$

#### Overqualification

After feasibility, schedulers prioritize minimizing overqualifications. Overqualification is the difference between the qualification rank of the pilot allotted to the slot and the minimum qualification rank required for that slot. For example, if the minimum qualification rank required for slot 'A' is "FPNC" (which is equivalent to the lowest

rank and hence rank=1) and the qualification rank of the pilot assigned to slot A is "FPQC" (which is the sixth lowest rank and hence rank=6), the overqualification for that slot is $6 - 1 = 5$. The schedulers want to minimize overqualifications so that the highly qualified pilots are available for the harder flight missions but the schedulers allow for some slack in the overqualifications. For instance, having an overqualification of 3 is fine as it might be possible that there weren't any low ranking pilots available for the event but having an overqualification of 20 is not preferred as it is highly unlikely that there weren't any other pilots within 20 ranks for that particular slot. Hence, we construct our overqualification score as:

$$r_{OQ} = \tanh(cutoff - OQ), \tag{2.7}$$

where $OQ$ is the overqualification for the current slot to which the pilot is assigned and $cutoff$ is the slack allowed for overqualification. We use $cutoff = 5$ for our experiments. Figure 2-2 shows the variation of the overqualification score as a function of overqualification.

**Training Requirements**

The schedulers also care for the completion of the training requirements of the pilots in the squadron. Pilots need to complete certain minimum training requirements to progress through the ranks along with staying current in their rank. Also, this prevents pilots from flying the same type of flight repeatedly. To incentivize completion of training requirements for pilots, we give a +1 score if the agent places a pilot to a slot which reduces the training requirement of the pilot by one and give a zero otherwise.

$$r_{req} = \begin{cases} +1 & \text{training requirement satisfied} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

**Reward structure**: Combining the feasibility, overqualification and training re-

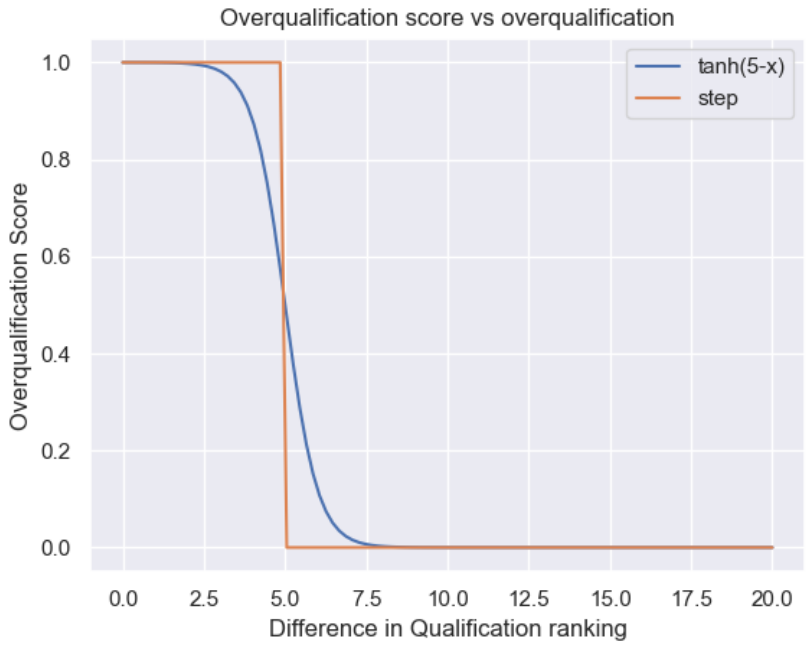


Figure 2-2: Variation of the overqualification score as a function of overqualification. The tanh function allows for some slack in overqualification upto the cutoff and then gives a score of zero when the overqualification is much larger than the cutoff

quirements scores from Equations 2.6, 2.7 and 2.8, we get the reward function as:

$$r = pilotPlace_{wt} \cdot r_{place} + OQ_{wt} \cdot r_{OQ} + req_{wt} \cdot r_{req} \tag{2.9}$$

where $OQ_{wt}$ and $req_{wt}$ are weights to scale the relative importance of placing valid pilots to slots, overqualification and completion of training requirements.

## 2.3 Experiments

### 2.3.1 Evaluation Procedure

We explain the evaluation procedure we use for all of our experiments in this section. We train all of our RL agents using Proximal Policy Optimisation (PPO) for $10,000$ epochs, where each epoch consists of 4000 environment steps. While training our RL agent, after every $10,000$ environment steps, we evaluate the model for 10 episodes. We average the results over these 10 episodes and plot them. All the

hyperparameters for our experiments are available in Appendix C. The episodes are randomly generated in the simulator using the procedure described in Section 2.2.3. For comparison against baseline methods, we create 100 random episodes and create schedules for those set of events using the corresponding MIP formulations for military crew-scheduling introduced in [10, 25].

### 2.3.2 Action space masking

With the discrete action space defined in Section 2.2.5, the number of valid actions possible at any time step changes as the episodes progress through. For example, at a particular state we have 10 pilots available for the current slot but in the next time-step, we have 14 pilots available. This might be due to various different reasons like some new pilots got freed up from their previous assignments, the new event had a lower qualification requirement and hence more pilots were eligible for it, etc. To avoid repeatedly sampling invalid actions in this discrete action space, we apply PPO in conjunction with a technique known as action-masking, which "masks out" invalid actions and then just sample from those actions that are valid as used in recent works on RL for game-based environments like StarCraft, DOTA, etc. [4, 51, 57]. In essence, action-masking re-normalizes the probability distribution predicted by the actor network to only account for valid pilots and then sample actions from this re-normalized distribution.

Figure 2-3 shows the learning curve for the RL agent with and without action-masking. Clearly, the action-masking helps in improving the sample complexity of the agent and is able to assign pilots to all the events in the episode. Whereas, on the other hand the "naïve" method (without action-masking) is not able to complete all the assignments even after training for $12,000$ episodes. For this experiment we use $OQ_{wt} = 0$ and $req_{wt} = 0$ in Equation 2.9 as we just care about finding feasible schedules. For all further experiments in this chapter, we use action-masking since it has a clear advantage with sample complexity as compared to the naïve method.

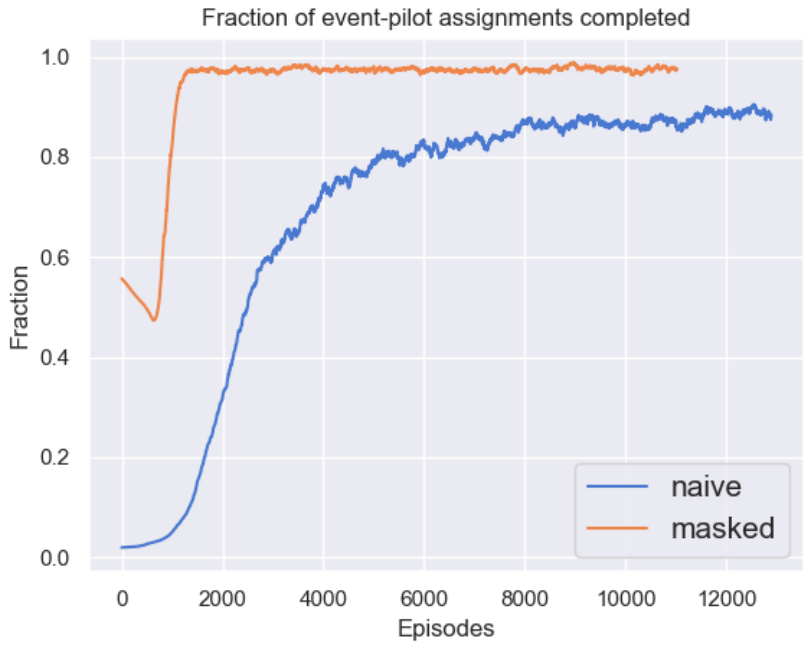


Figure 2-3: Fraction of the total events in the episode for which the agent is able to assign pilots

### 2.3.3 Minimizing Overqualification

Before using the complex reward function as described in Equation 2.9 which combines feasibility, overqualification and completion of training requirements, we show the impact of $OQ_{wt}$ on the performance of the agent. We vary $OQ_{wt}$ and set $pilotPlace_{wt} = 1 - OQ_{wt}$. Hence the reward at any time step is:

$$r_t = OQ_{wt} \cdot \tanh(cutoff - OQ) + (1 - OQ_{wt}) \tag{2.10}$$

We compare this against the overqualification minimizing mixed-integer program (MIP) introduced by Koch [25]. Koch introduces two different formulations for minimizing overqualifications. The first formulation attempts to assign the lowest qualified pilot feasible for each slot. We denote this formulation as MIP in our results. The second formulation, which we call Advanced OQ MIP in our results, uses a subset of qualifications for a few type events (AD and SOLL II). Since only special qualified pilots are able to fly the AD and SOLL II events, it would be unnecessary to consider

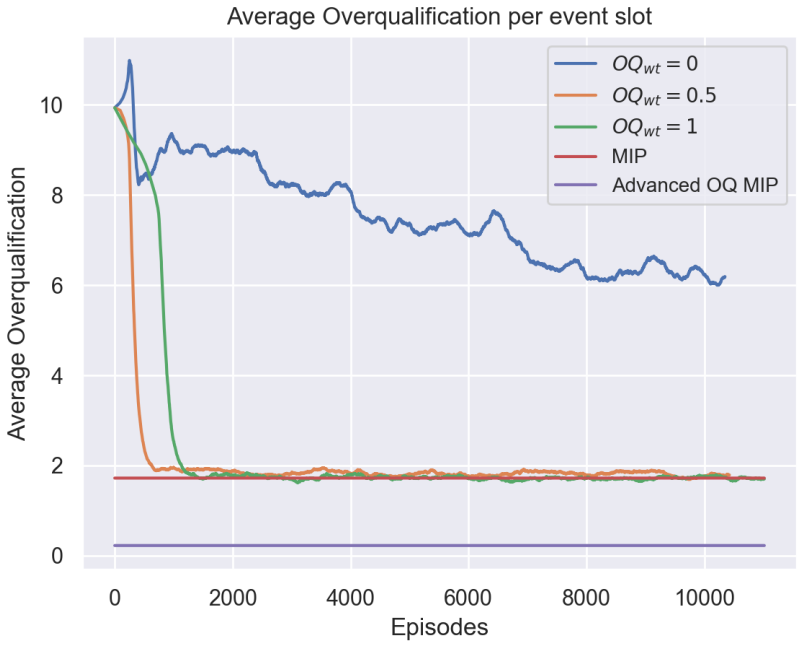


Figure 2-4: Average overqualification obtained by the RL agent for different values of $OQ_{wt}$ compared against the MIP and the advanced OQ MIP formulations.

the entire set of qualification ranks to evaluate the overqualification. And hence the advanced OQ MIP formulation adjusts for these ranks while evaluating the objective function.

Figure 2-4 shows the average overqualification per event slot for the RL agent and both of the MIP formulations. The average overqualification obtained by the MIP formulation on a set of randomly created events for a duration of 6 months is $1.7223 \pm 0.0233$ and that for the advanced MIP formulation is $0.2225 \pm 0.0832$. The RL agent is able to perform almost as well as the MIP formulation when the overqualification weights $OQ_{wt} = 0.5$ and $OQ_{wt} = 1$ but is outperformed by the Advanced OQ MIP. With the specialized objective function for the Advanced OQ MIP the optimizer is able to find much better solutions with respect to overqualification.

### 2.3.4 Maximizing Completion of Training Requirements

For experiments with maximisation of training requirements, we set $OQ_{wt} = 1$ and $pilotPlace_{wt} = 0.1$ and vary the value of $req_{wt}$. We set $OQ_{wt} = 1$ because it gave the

least average overqualification as shown in the previous section. We set $pilotPlace_{wt} = 0.1$ because we did not want the that term to dominate the reward value. Hence the reward at any time step is:

$$r_t = 0.1 \cdot r_{place} + \tanh(cutoff - OQ) + req_{wt} \cdot r_{req} \tag{2.11}$$

Since there are two different metrics: overqualification and number of training requirements completed, we use MIP formulations with two different objective functions for comparisons. We use the same overqualification (and advanced OQ) MIP formulations as defined in the previous section for the overqualification metric. For comparing the training requirements metric, we use a different MIP formulation which explicitly maximizes the number of training requirements completed in the schedule introduced by Koch [25]. This formulations works with similar constraints as the OQ minimizing MIP. But the schedules created by the OQ minimizing MIP and the training requirements maximizing MIP can be different since they optimize different objective functions. On the other hand the schedule created with the RL agent tries to optimize for both overqualification and training requirements in a single schedule.

Figure 2-5 shows the variation in the number of training requirements completed in an episode with different values of $req_{wt}$ for the RL agent and the training requirement MIP formulation. The average number of training requirements completed by the MIP formulation on a set of randomly created events for a duration of 6 months is $68.5321 \pm 1.0421$. Figure 2-6 shows the variation of average overqualification obtained with the same RL agent for different values of $req_{wt}$. The RL agent is able to perform as well as the training requirement MIP formulation and the overqualification MIP formulation even whilst trying to optimize two different objective functions at the same time. Again, it is still not able to perform as well as the Advanced OQ MIP.

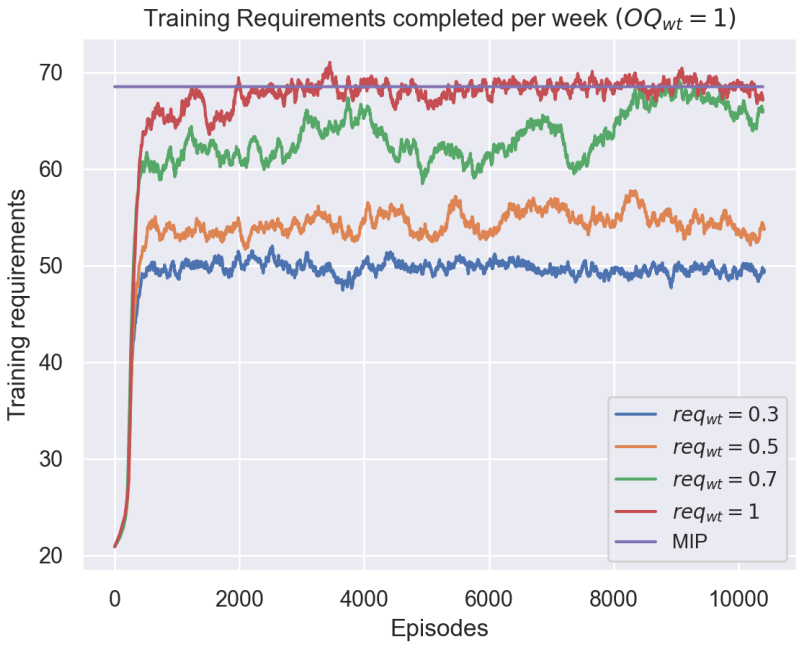


Figure 2-5: Number of training requirements completed by the RL agent for different values of $req_{wt}$ and $OQ_{wt} = 1$ compared against the Training Requirement MIP

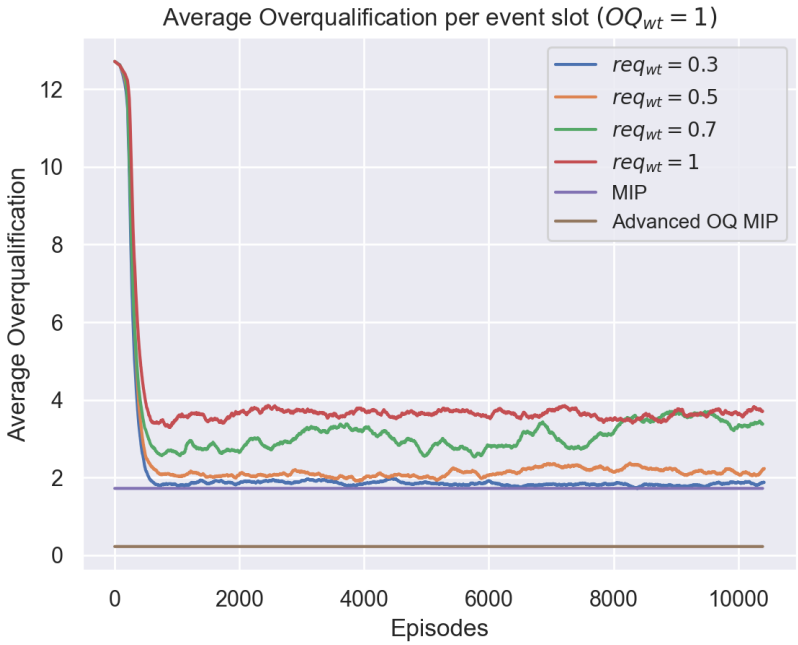


Figure 2-6: Average overqualification obtained by the RL agent for different values of $req_{wt}$ and $OQ_{wt} = 1$ compared against the OQ MIP and the advanced OQ MIP formulations.

## 2.4 Discussion

The results show that the RL agent is able to learn to create schedules which optimize for different objective functions. The RL agent is able to perform as well as the overqualification MIP (on average) when the reward structure just optimizes for overqualification but is not able to perform as well as the advanced OQ MIP formulation. This is because of the modified overqualification definitions used in the MIP formulation. When optimizing for both completion of training requirements and overqualifications, the RL agent is again able to perform as well as the overqualification MIP and the training requirement MIP (on average). The advantage of the RL agent is that it is able to optimize for both together whereas there is a requirement of having two different formulations (objective functions) for these two metrics in the MIP case.

## 2.5 Conclusions and Future Work

In this chapter we introduced a reinforcement learning formulation for the Air-Force crew scheduling problem. We defined the state space, action space and the reward structure for the RL formulation. The reward structures were defined according to the metrics (minimizing overqualification and maximizing completion of training requirements) we wanted to optimize. We first show that action masking is important in improving the sample complexity of training the RL agent with the defined MDP formulation. Then we show that the RL agents with a reward structure engineered to optimize for certain metrics are able to perform as well as their MIP counterparts. We show that the RL agent can optimize for multiple metrics in a single formulation whereas the MIP formulations require separate objective functions. The final conclusion is that, although the RL agent is able to perform as well as their MIP formulations, it is never able to perform better than it. This is partly because the MIP formulation has a global view of the problem and can backtrack its decisions whereas the RL agent has a local view of the problem and cannot make any changes to the

actions it took in the beginning of the episode. To this end we combine reinforcement learning with integer programming in the next chapter to get the advantages of both. Future work includes experimenting with more metrics, having a user preference variable to adjust the trade-off between the different metrics to optimize with the RL agent.

# Chapter 3

# Combining Reinforcement Learning and Integer Programming

In many practical applications, it is important that the schedules are robust to uncertainties. Robust scheduling involves the building of schedules that will undergo minimal change when faced with unknown future disruptions. Whilst MIP formulations of scheduling problems can be extended to account for robustness, the resulting problems are often much more computationally challenging than their baseline, non-robust counterparts. Even with state-of-the-art solvers, such modifications to accommodate robustness can add hours to the time needed to compute an optimal schedule, sometimes making them impractical for real-world use.

Thus we propose a technique, Neural network IP Coefficient Extraction (NICE), that seeks to find a quick-but-approximate solution to a scheduling problem with an additional robustness objective, by using reinforcement learning (RL) to guide the IP formulation. First a *feasible* schedule is created using a baseline IP. Simultaneously, we train an RL model to build a schedule for the same problem, using a reward function that leads to more robust schedules (but that would have added considerable computational burden if encoded directly in the IP formulation). Then, rather than use the RL model to create a schedule directly, we use the probabilities in its output layer to assign coefficient weights to the decision variables in our simpler IP to create a feasible schedule. By doing so, we leverage the intuition behind knowledge distillation

[18] that the distribution of values in the output layer of a neural network contains valuable information about the problem.

NICE allows us to approximate the robust scheduling formulation with significantly fewer variables and constraints. Across a variety of disruption scenarios, we find that NICE creates schedules with 33–48% fewer changes than the baseline. Moreover, in certain practical problem instances, NICE finds a solution in a matter of seconds; the corresponding IP that explicitly optimizes for robustness fails to produce a solution within 90 minutes for the same scenarios.

In essence, NICE is a technique to approximate complicated IP formulations using RL. To the best of our knowledge, NICE[1] is the first method to use information extracted from neural networks in IP construction. We illustrate the performance of NICE in creating robust (disruption-resistant) crew schedules (i.e., assignments of pilots to flights). However, robust crew scheduling is only one application of NICE; we believe that the method is potentially applicable to a wider range of discrete optimization problems.

We begin this chapter with some background on knowledge distillation (upon which NICE is based) in Section 3.1. In Section 3.2 we show the baseline integer programming (IP) formulation, a modified buffer IP formulation, introduce the motivation behind NICE and the algorithmic details of our NICE formulation. This is followed by experiments and results in Section 3.3 where we show the effectiveness of NICE in different scenarios. Finally we end with the discussions of the results obtained, conclusions and future work in Sections 3.4 and 3.5.

## 3.1 Knowledge Distillation

Hinton et al. [18] explored the use of internal neural network values to distill the knowledge learned by a model. They reasoned that the probabilities in a neural network's output layer carry useful information, even if only the maximum probability value is used for ultimate classification: "An image of a BMW, for example, may only

[1]The code is available at https://github.com/nsidn98/NICE

have a very small chance of being mistaken for a garbage truck, but that mistake is still many times more probable than mistaking it for a carrot." In this work, they used the values from the input to the output layer of a larger neural network, as well as the training data itself, to train a smaller neural network. This smaller neural network achieved fewer classification errors than a network of the same size trained only on the training data. Using a similar approach, they trained a neural network on speech recognition data with the same architecture as a neural network trained on the data directly. They found that the new, distilled model performed better than the original one; it also matched 80% of the accuracy gains attained by averaging an ensemble of 10 neural networks with the same architecture, each initialized with different random weights at the beginning of training. Similarly, our approach uses the probabilities output by a neural network to extract objective function coefficient weights.

## 3.2 Approach

In this section, we seek to build robust schedules for the "crew scheduling problem" and use the same setup of assigning pilots to slots in the flights as described in Section 2.2.1. To reiterate, we consider the scenario in which we are given a collection of flights that must be flown by a given squadron of pilots. Each flight has multiple slots, each of which must be filled by a different pilot. Each slot has qualification requirements that must be satisfied by any pilot who is assigned to that slot. Depending on their qualification, a pilot would only be eligible to fill a subset of slots. Finally, every pilot has some specified availability. We discretize our schedule into days. A feasible schedule assigns pilots to slots such that every flight in the schedule horizon is fully covered, and the qualification requirements for the slots and availability restrictions of the pilots are satisfied.

### 3.2.1 Baseline Integer Program Formulation

Chin [10] and Koch [25] created a baseline IP for the crew scheduling problem, producing a satisfactory assignment with respect to all relevant constraints. We use a similar construction for our baseline IP, with the primary difference of using a decision variable for the assignment of each pilot to each slot, rather than one for each pilot to each flight. We define the following sets and subsets:

| | |
|---|---|
| $i \in I$ | The set of pilots |
| $f \in F$ | The set of flights |
| $s \in S$ | The set of all slots |
| $U_f \subset F$ | Flights that conflict with flight $f$ |
| $L_i \subset F$ | Flights that conflict with pilot $i$'s leave |
| $S_f \subset S$ | The set of slots belonging to flight $f$ |
| $Q_i \subset S$ | Slots that pilot $i$ qualifies for |

We use the binary decision variable $X_{is}$, which is 1 if pilot $i \in I$ is assigned to slot $s \in S$, and 0 otherwise. We now have the following equation:

$$\max \sum_{i \in I} \sum_{s \in S} X_{is} \tag{1.1}$$

such that:

$$X_{is} = 0 \qquad \forall i \in I, f \in L_i, s \in S_f \tag{1.2}$$

$$X_{is} = 0 \qquad \forall i \in I, s \in S \setminus Q_i \tag{1.3}$$

$$\sum_{s \in S_f} X_{is} \leq 1 \qquad \forall i \in I, f \in F \tag{1.4}$$

$$\sum_{i \in I} X_{is} = 1 \qquad \forall s \in S \tag{1.5}$$

$$\sum_{s \in S_f} X_{is} + \sum_{s' \in S_{f'}} X_{is'} \leq 1 \qquad \forall i \in I, f \in F, f' \in U_f \tag{1.6}$$

$$X_{is} \in \{0, 1\} \qquad \forall i \in I, s \in S \tag{1.7}$$

Constraints (1.2) and (1.3) ensure that pilots are only assigned to slots that they are qualified for, and that pilots will never be assigned to flights that conflict with their leave. Constraint (1.4) prevents the scheduling of pilots to multiple slots on the same flight. Constraint (1.5) ensures that every slot gets filled by exactly one pilot. Constraint (1.6) prevents assignment of pilots to conflicting flights. We make our pilot-slot decision variable binary with Constraint (1.7). Finally, Equation (1.1) ensures that we fill as many slots as possible. Due to our requirement that each slot have exactly one pilot associated with it, this equation always produces the same objective value.

### 3.2.2 Buffer Formulation

Chin [10] utilizes additional constraints and decision variables to optimize schedules for robustness by increasing the amount of buffer time between flights. For our purposes, we consider the buffer time between two flights to be the number of full days between their start and end time. For example, suppose pilot $X$ is assigned to flights $A$ and $B$; flight $A$ ends on day 1, and flight $B$ starts on day 5. We say that there is a buffer time of 3 days for pilot $X$.

To incorporate buffers into an IP, Chin [10] identifies all flight pairings that would create a buffer less than or equal to some maximum threshold, $T_{\text{buffer}}$, used to dampen the complexity of realistic problem instances. Then, $\{0, 1\}$ decision variables $B_{iff'}$ for all pilots $i \in I$ and flights $f, f' \in F \times F$ are created. Constraints are used to ensure that $B_{iff'}$ is 1 if and only if the following conditions are met:

1. $f \neq f'$
2. Pilot $i$ qualifies for at least one slot in both $f$ and $f'$.
3. $f$ and $f'$ have a buffer between 0 and $T_{\text{buffer}}$, inclusive.
4. Pilot $i$ is assigned to consecutive flights $f$ and $f'$.

Chin [10] then defines a buffer penalty $b_{iff'} \in [-1, 0)$ that gets more negative for lower buffers: down to -1 when the buffer between $f$ and $f'$ is 0, and closest to 0 when the buffer is $T_{\text{buffer}}$. Note that buffers longer than $T_{\text{buffer}}$ effectively have a penalty of 0. Finally, he incorporates all of this into the following objective function to optimize buffer time:

$$\max \sum_{i \in I} \sum_{f,f' \in F \times F} b_{iff'} B_{iff'} \tag{3.2}$$

In our buffer IP, we incorporate this formulation with the assistance of additional auxiliary {0,1} decision variables for each pilot-flight combination, setting it to 1 if a pilot is assigned to any slot on a given flight, and 0 otherwise. In our experiments, we found that, in certain problem instances, the buffer IP was highly effective in producing robust schedules.

### 3.2.3 NICE

#### Motivation

As seen in Chapter 2, when building schedules with non-robustness objectives, the RL-produced schedules would often approach the effectiveness of the IP schedules with respect to the optimized metric (over-qualification, completion of training requirements), but would never do better. This observation gave us two insights.

First, the RL model assigns pilots to slots sequentially, operating in a "greedy" fashion. Once an assignment is made, it cannot be changed. Thus, the RL agent lacks a global view of the full schedule in its state space, meaning it has imperfect information at the time of each pilot assignment. In contrast, the IP scheduling approach, which optimizes an objective function across all pilot assignments, can factor in tradeoffs created by the complex interplay of related constraints.

Second, it was clear that our RL scheduling agent was capable of learning. While it could not match the performance of the IP schedules in our preliminary exploration, it still produced schedules with considerably better objective performance than the baseline. When scheduling objectives can easily be captured in an IP, this observation

is not particularly helpful. However, for more complicated optimizations that integer programming struggles with, this insight proves useful: to avoid the greedy pitfalls of RL for scheduling while still leveraging the knowledge learned by our neural network, we can use the probabilities produced by the output layer in our IP scheduling formulation.

These two observations motivated the creation of NICE. NICE uses RL to approximate sophisticated integer programs with a simpler formulation. We apply this technique to the crew scheduling problem.

### NICE IP Formulation

As an IP, NICE closely resembles the baseline formulation. The only difference is in the coefficients for the objective function. Recall that our decision variable, $X_{is}$, is 1 if pilot $i$ is assigned to slot $s$ and 0 otherwise. This variable aligns neatly with our RL scheduling formulation, which considers slots in a fixed order. At each slot, it produces a probability for each pilot that captures how likely assigning that pilot to that slot is to maximize reward in a given scheduling episode. It then assigns the pilot with the maximum probability to that slot. We can use $a_{is}$ to refer to the probabilities output by the network for the assignment of pilot $i$ to slot $s$. Now, to leverage the knowledge learned by the RL scheduling approach in our IP formulation, we can incorporate $a_{is}$ into our objective function:

$$\max \sum_{i \in I} \sum_{s \in S} a_{is} X_{is} \tag{3.3}$$

With this new equation, our IP scheduling approach is incentivized to pick the pilot with the highest probability possible at each slot, subject to constraints and possible rewards for other slots. This new formulation approximately captures the reward function used by the RL agent while giving it a global view of pilot assignment. Figure 3-1 shows the procedure for executing NICE.

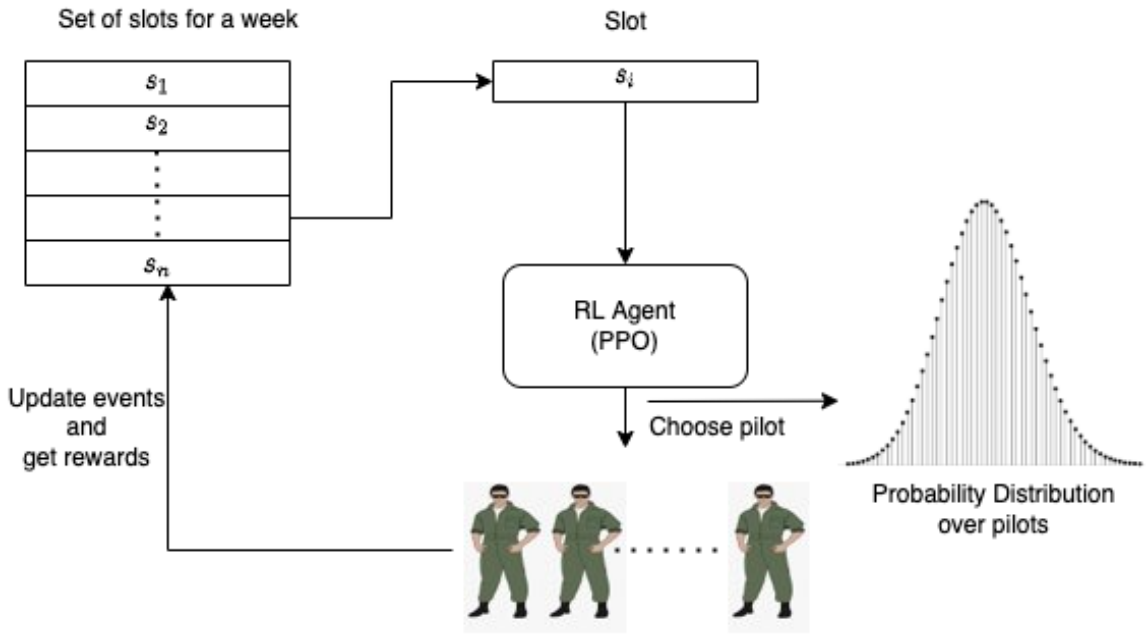


Figure 3-1: The procedure for executing NICE is quite similar to the RL model from Chapter 2. But instead of choosing the pilot suggested by the RL agent (choosing the pilot with the maximum probability), we use the predicted probability distribution over the pilots to feed in the IP formulation

### Extracting Probability Weights

An important issue we had to address was the extraction of $a_{is}$ from our RL neural network. The actions taken at each state of the RL scheduling process can impact the pilot probability vector at later states. Thus, the order that the RL scheduler fills the slots has a potentially confounding impact on the $a_{is}$ weights. Extracting the probability vector at each slot while running the RL scheduling process as normal could cause the specific actions taken to bias our $a_{is}$ values, diminishing the advantage of the IP scheduler's global outlook.

In our experiments, we used two different approaches. In the first one, we took a Monte Carlo approach, trying to approximate the average weight across all possible orders of scheduling the slots. In this approach, we randomly shuffled the order of slots that the RL scheduler had to assign and then recorded the $a_{is}$ weights at each step of the scheduling process. We ran this process $n$ times to get $n$ total $a_{is}$ values for each pilot-slot pair. If the RL agent could not fill a slot in that round of scheduling,

we set the $a_{is}$ value to 0. We then averaged the $n$ values for each pilot-slot pair to get our final $a_{is}$ values. Note that this method causes the scheduling process to take longer for higher values of $n$ because it has to run more RL scheduling rounds.

The next approach, which we call the "blank slate" approach, exploits the fact that the probability weights for the pilots produced by the first slot do not depend on any previous actions taken. Thus, we can make each slot our first scheduled slot to get weights that do not depend on previous decisions. To do so, for each slot $s$ in our fixed order, we initialized a new RL agent with the same underlying neural network, cutting out all of the states that occur before $s$ then extracting the $a_{is}$ values for the pilots on that slot.

Figure 3-2 demonstrates using the Monte Carlo approach to extract weights from the neural network, then feeding those weights into the IP scheduler. To summarize, to build our NICE schedule:

1. We train a neural network on the DES version of the crew scheduling problem.

2. We use either the Monte Carlo or "blank slate" approach to extract probabilities from the output layer of the neural network for the assignment of each pilot to each slot.

3. For each pilot-slot combination, we use the extracted probability as the coefficient ($a_{is}$) for its respective pilot-slot decision variable ($X_{is}$)

4. Using this objective function and its constraints, we solve the IP to obtain our scheduling solution.

We note that while we apply NICE specifically to the crew scheduling problem in this paper, NICE is highly generalizable as it can be used to obtain a solution to any problem where there is an isomorphism between the IP and DES formulation. Figure 3-3 show the isomorphism for the robust crew scheduling problem.

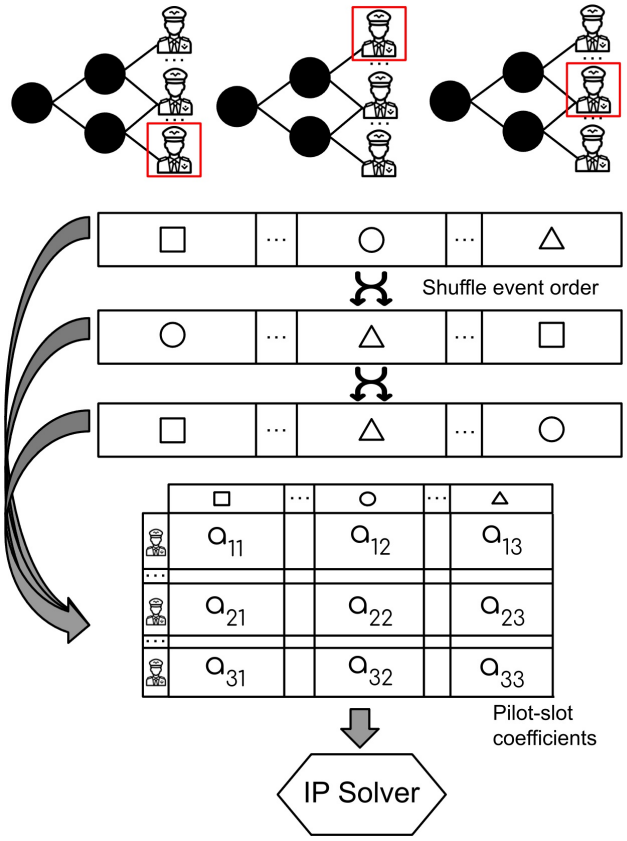


Figure 3-2: Once we train our network, in the Monte Carlo approach, we run the scheduler $n$ times, shuffling the order of slots each time. We then average the probability values in the output layer across the $n$ runs over pilots and slots to obtain our NICE coefficients. Finally, we pass these coefficients to the IP solver to obtain our NICE-generated schedule.

### Incorporating Robustness

To use NICE to build robust schedules, we train an RL scheduler to optimize buffer time in its assignments. To do so, we include a reward of $b+1$ whenever the RL agent places a pilot on an event that forms a buffer of length $b$ with the pilot's most recent event. We add the $+1$ term to reward the agent for making a placement, regardless of buffer. We give the agent a reward of $T-1$ when it places a pilot with no previous events scheduled. We chose $T-1$ because this is the maximum reward for the $T$-day schedules that the RL agent trains on. For example, imagine pilot $X$ is assigned to a 0-day flight starting and ending on day 1. If the pilot were assigned to a flight starting on day 7, that assignment would earn a reward of 6, because there are 5

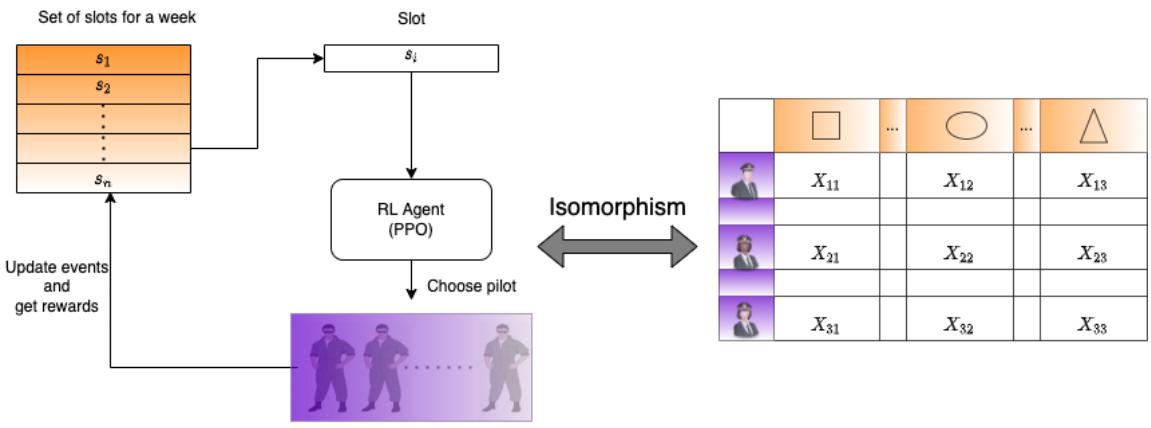


Figure 3-3: The isomorphism between the IP and DES formulation for the crew scheduling problem

days between day 1 and day 7, exclusive. We do not use a maximum buffer value like $T_{\text{buffer}}$ in the IP formulation because larger $T_{\text{buffer}}$ values do not noticeably affect the run time of our program like it does with the IP. We included two exceptions to this reward policy. First, to incentivize building full schedules, the agent earns a reward of 25 when all slots in an episode are scheduled. Second, to deter the creation of incomplete schedules, we give the agent a reward of $-10$ when it is unable to schedule all events in an episode.

In short, we give local rewards to our RL agent to help it build complete, robust schedules using buffers as a heuristic; our hope is that this training method will ultimately produce probability weights that, when extracted, lead our IP solver to build robust schedules.

## 3.3 Experiments

### 3.3.1 RL Training

To train our RL agent, we use the same Open AI Gym type environment introduced in Chapter 2. To build the neural network for our NICE scheduler, we trained a variety of models with different hyperparameter combinations. One of the hyperparameters of particular interest was the training schedule density, $d$. Recall that we scheduled

$\alpha$ flights and $\beta$ simulators in a round of scheduling. During training, we multiplied $\alpha$ and $\beta$ by $d$ so the agent would schedule more flights in the same time period. We trained RL models with $d \in \{1, 2, 3\}$. We trained a model with 5 different seeds for each value of $d$, ultimately creating $3 \times 5 = 15$ different neural networks for testing. The hyperparameters for all the experiments are listed in the Appendix C. We note that, because the output layer is equal to the size of the number of pilots, adding a pilot would require us to re-train the network with the new shape.

**Configuration**

The state space for the RL agent includes:

1. A binary vector for the pilots available for the current slot
2. A flattened vector encoding the current event to be scheduled, consisting of:
   (a) A one-hot encoding of the event type
   (b) A binary vector indicating whether each training requirement was true or false
   (c) A binary vector representing the pilots assigned to the current event (Each event may have 2-3 pilot slots)
   (d) The event duration (in days)
   (e) The number of days between the start of the scheduling episode and the start of the event
   (f) The number of days between the start of the scheduling episode and the end of the event
3. A vector containing the total number of training requirement fulfillments each pilot could receive for flying that event, if it were flown a sufficient number of times

| % Flights Delayed | Number of Disruptions | | | | Significance of Difference ($p$-value) | | |
|---|---|---|---|---|---|---|---|
| | NICE | Baseline IP | RL | Buffer IP | NICE-Baseline | NICE-RL | NICE-Buffer |
| 25 | $\mathbf{0.34 \pm 0.71}$ | $0.61 \pm 1.07$ | $32.6 \pm 7.33$ | $0 \pm 0$ | 0.03 | $< 0.01$ | $< 0.01$ |
| 50 | $\mathbf{0.67 \pm 0.92}$ | $1.16 \pm 1.55$ | $27.1 \pm 7.13$ | $0 \pm 0$ | $< 0.01$ | $< 0.01$ | $< 0.01$ |
| 75 | $\mathbf{0.66 \pm 0.99}$ | $1.13 \pm 1.73$ | $23.0 \pm 6.34$ | $0 \pm 0$ | 0.01 | $< 0.01$ | $< 0.01$ |
| 100 | $\mathbf{0.63 \pm 0.82}$ | $1.06 \pm 1.50$ | $17.9 \pm 6.29$ | $0 \pm 0$ | 0.01 | $< 0.01$ | $< 0.01$ |

Table 3.1: Average and standard deviation of disruptions across scheduling methods when flights are delayed (lower values are better). Scheduling density of 1.

### 3.3.2 Scheduling Parameter Selection

For further experimentation, we had to pick the best combination of RL model and weight extraction method to use. We parameterize our weight extraction methods with the variable $n$, where $n = 0$ represents the "blank slate" extraction method mentioned previously, and $n > 0$ represents the $n$ value used in the Monte Carlo approach.

To select the best combination, we built an environment where, using the same flight generation process to train the RL scheduler, we generate 1 week's worth of flights. From these flights and associated slots, we generate pilot-slot pairings using the baseline integer program and the NICE schedule with weights extracted from the selected RL network and the given $n$ value. Next, one day into the schedule, we delayed 50% of the flights that had not already left. We pushed back each flight by a number of days randomly chosen uniformly between 1 and 3, figuring that delays longer than 3 days were relatively rare. To fix this disruption in both of these schedules, we used an integer program that minimized the number of changes to the pilot-slot pairings. From this disruption resolver, we end up with the number of disruptions that occurred under each schedule. Sometimes, we would randomly generate a series of flights that made it impossible for the IP or NICE approach to schedule because a pilot-slot pairing did not exist that met all of the constraints. In these cases, we skipped to the next series of randomly-generated flights, not recording any disruption data because there was no schedule to disrupt. Then, for each neural network and for each $n$ value $n \in \{0, 2, 4, 8\}$ (60 experiments total), we ran this

| % Flights Delayed | Number of Disruptions | | | Significance of Difference ($p$-value) | |
|---|---|---|---|---|---|
| | NICE | Baseline IP | RL | NICE-Baseline | NICE-RL |
| 25 | $\mathbf{1.99 \pm 1.99}$ | $2.99 \pm 2.68$ | $59.2 \pm 13.1$ | $< 0.01$ | $< 0.01$ |
| 50 | $\mathbf{3.17 \pm 2.27}$ | $5.01 \pm 4.33$ | $51.2 \pm 12.2$ | $< 0.01$ | $< 0.01$ |
| 75 | $\mathbf{3.32 \pm 2.33}$ | $6.32 \pm 5.64$ | $43.1 \pm 12.4$ | $< 0.01$ | $< 0.01$ |
| 100 | $\mathbf{2.81 \pm 2.23}$ | $4.70 \pm 4.49$ | $35.4 \pm 10.5$ | $< 0.01$ | $< 0.01$ |

Table 3.2: Average and standard deviation of disruptions across scheduling methods when flights are delayed (lower values are better). Scheduling density of 2. Buffer IP did not build a single schedule for 90 minutes and timed out, so we do not include it.

scenario 20 times, comparing the average number of schedule disruptions between the baseline IP and the NICE scheduler across those 20 runs.

We calculated the ratio of disruptions, $r$, between the two methods, where $r < 1$ indicates that the NICE method provided a schedule with fewer disruptions on average than the IP baseline. We then determined the median $r$ value over seed values for each training density and weight extraction method combination. We used the lowest median $r$ value to determine our final scheduling method and underlying neural network for NICE. As a result of this process, we obtained a network trained with a density of 1 and an $n$ value of 2. In this case, across seed values, three networks produced the same median value for this combination, so we arbitrarily chose a network from those three models. The median $r$ value was 0.25, with a range of 0.08 (0.25 to 0.33). We used this network and $n$ value in our further experiments.

Overall, the combinations had fairly stable performance over seed values, with the highest range in $r$ across seeds being 0.81. Notably, the $n$ value of 0 produced the 3 highest ranges (0.81, 0.69, and 0.56), indicating that the "blank slate" weight extraction method led to a wider range of disruptions in produced schedules across random seeds.

### 3.3.3 Baseline Scheduling Performance

To show the efficacy of NICE scheduling, we ran our best scheduling combination on the same disruption scenario, this time using different percentages $f$ of flights delayed. We selected $f$ values of 25%, 50%, 75%, and 100% and ran each scenario

100 times. We compared NICE to directly using our RL agent or using the buffer integer program (with $T_{\text{buffer}} = 4$ days). We show the results of our buffer-rewarded NICE scheduler in Table 3.1. During these runs, the buffer IP and NICE scheduler had similar run times, both averaging less than 0.85 seconds to create each schedule.

We note that, because the constraints are exactly the same between the IP and NICE scheduling approaches, we skipped recording the disruption value for the IP scheduler if and only if we also skipped the disruption value for the NICE scheduler. Because of this alignment, the $p$-values comparing the two methods were obtained using a 2-tailed dependent t-test for paired samples between the NICE and IP schedulers. This was not the case for the RL approach, which can generate a partial schedule, stopping when it is not able to assign a pilot to the next slot due to its previous decisions. When the RL approach created a partial schedule, we did not record its performance on that schedule to include in the average because it was unable to produce a full schedule like the baseline IP and NICE schedulers. For this reason, in the NICE vs. RL comparison, we use Welch's t-test [54] for independent samples.

### 3.3.4 Highly-Constrained Scheduling Scenarios

In the schedule disruption scenario that we considered, the buffer IP formulation had little trouble building a robust schedule in a reasonable amount of time. However, the time advantages of NICE become apparent in a more constrained scheduling setting. To demonstrate this efficiency, we performed the same experiment on NICE, averaging over 100 trials. This time, though, we used a scheduling density of 2, creating twice the number of flights on average in each round of scheduling. This scenario is realistic in settings where, due to outside factors, many flights must be filled. For timing reasons, we only compared NICE with the baseline IP and the pure RL scheduler. We show the results for this experiment in Table 3.2. Importantly, across all flight delay percentages, the NICE scheduler took an average of between 1.85 and 1.90 seconds to build a schedule, with a standard deviation between 0.55 and 0.60 seconds. We ran the exact same experiment for each disruption percentage for 10 iterations with the buffer-optimizing IP. However, we ended each experiment after

90 minutes, at which point the buffer IP had not finished building a single schedule.

### 3.3.5 Move-up Crews

During our experimentation, we considered another factor that can lead to more robust schedules, move-up crews [44]. We found that move-up crews, even when optimally scheduled, did not create particularly robust schedules, but we include our experimental results here for reference. For the sake of clarity, because we dealt with individual pilots rather than crews, we will depart from the literature and use the term move-up *pilot* rather than move-up crew. A move-up pilot is someone who, by nature of their qualification and one of their scheduled flights, is readily available to move up to another flight should someone on that flight become unavailable, perhaps due to a delayed flight.

#### IP Formulation

We use a similar IP formulation to Chin [10] to increase move-up pilots in our schedules. We first define a threshold, $T_{\text{move}}$, for how far out we should look for move-up pilots. Now, we give a more formal definition of a move-up pilot: pilot $j \in I$, assigned to flight $g \in F$, is a move-up pilot for slot $s \in S_f$ on flight $f \in F$ if and only if all of the following conditions hold:

1. $f \neq g$
2. Flight $g$ starts at the same time as or later than flight $f$, and no later than $T_{\text{move}}$ days after $f$ starts.
3. Flight $g$ ends at the same time as or later than flight $f$.
4. Pilot $j$ is not on leave that overlaps with f.
5. Pilot $j$ is not scheduled to any flights that start before $f$ and overlap with $f$.
6. Pilot $j$ is qualified for slot $s$.

Using constraints, we define the binary decision variable $M_{jg,fs}$ to be 1 if pilot $j$ assigned to flight $g$ is a move-up pilot for slot $s$ on flight $f$. We then use additional variables to build an objective function to maximize the number of move-up pilots in our final schedule. We can achieve this with the following objective function:

$$\max \sum_{i \in I} \sum_{f,g \in F \times F} \sum_{s \in S_f} M_{jg,fs} \tag{3.4}$$

### Reinforcement Learning Agent Training

To train an RL scheduler to optimize for move-up pilots, we followed the same experimental procedure for the buffer-optimized RL scheduler, but we changed the reward function. For move-up pilots, we give a reward of $m + 1$ whenever our agent assigns pilot $j$ to a slot on flight $g$, where $m$ is the number slots on other flights that pilot $j$ can serve as a move-up pilot for. We use a maximum move-up time (like $T_{\text{move}}$) of 2 days. We did not do so out of concern for program run time; on a practical level, moving a pilot to a flight any more than 2 days earlier would cause a significant burden on the pilot rather than supply the convenient schedule alleviation that move-up pilots are supposed to provide. Just like the reward function for buffers, we include a -10 penalty for incomplete schedules and a +25 reward for complete schedules. We trained 15 total neural networks with the move-up pilot reward structure, using 5 different random seeds and schedule densities of 1, 2, and 3.

### Model Selection

We followed the same procedure as the buffer-rewarded networks to select the best network and $n$ value to use for our weight extraction method. We used $n$ values of 0, 2, 4, and 8. We used this procedure to obtain $r$, the ratio of average disruptions in the baseline IP schedule to average disruptions in the NICE schedule with the chosen parameters. $r < 1$ indicates that the NICE schedule had fewer disruptions on average. The best median $r$ value was 0.46, with a range of 0.38 (0.25 to 0.63), produced with a scheduling density of 2 and an $n$ value of 2. Two networks with these parameters

and different seed values produced the median value, so we picked one arbitrarily. We used this model in our subsequent experiments.

The range across seeds for our move-up-rewarded NICE scheduler (0.38) was notably higher than the range for our best buffer-rewarded scheduling method, which was 0.08. Like the buffer-rewarded schedulers, the $n$/density combinations also had fairly stable performance across random seeds. The 3 highest ranges across random seeds were 0.88, 0.69, and 0.68. Similar to the buffer-rewarded schedulers, the 2 highest ranges were produced by the "blank slate" weight extraction method ($n = 0$).

### Baseline Scheduling Performance

Using the selected model and $n$ value, we ran the same disruption scenario as the buffer-rewarded NICE scheduler over 100 iterations. We compared the NICE scheduler against the baseline IP scheduler, the RL scheduler with the same underlying neural network, and the move-up IP scheduler with a $T_{\text{move}}$ value of 2. The results are shown in Table 3.3.

| % Flights Delayed | Number of Disruptions | | | | Significance of Difference ($p$-value) | | |
|---|---|---|---|---|---|---|---|
| | NICE | Baseline IP | RL | Buffer IP | NICE-Baseline | NICE-RL | NICE-Buffer |
| 25 | $\mathbf{0.63 \pm 0.93}$ | $0.61 \pm 1.07$ | $32.6 \pm 7.42$ | $0.33 \pm 0.63$ | .88 | $< .01$ | $< .01$ |
| 50 | $\mathbf{1.05 \pm 1.13}$ | $1.16 \pm 1.55$ | $27.6 \pm 7.24$ | $0.68 \pm 0.90$ | .51 | $< .01$ | $< .01$ |
| 75 | $\mathbf{1.19 \pm 1.25}$ | $1.13 \pm 1.73$ | $23.9 \pm 6.44$ | $0.74 \pm 1.04$ | .73 | $< .01$ | $< .01$ |
| 100 | $\mathbf{1.11 \pm 1.14}$ | $1.06 \pm 1.50$ | $19.1 \pm 6.47$ | $0.65 \pm 0.88$ | .76 | $< .01$ | $< .01$ |

Table 3.3: Average and standard deviation of disruptions across scheduling methods when flights are delayed (lower the better). Scheduling density of 1. The NICE and RL schedulers used the move-up reward function in their underlying neural network.

## 3.4 Discussion

Our results clearly show the advantages of the NICE approach. In the baseline scheduling scenario, NICE produced schedules that provided 40% to 45% of the disruption reduction of the buffer IP compared to the baseline IP. In a powerful display

of its usefulness, in a dense scheduling environment, NICE performed 33% to 48% better than the baseline IP, producing 100 schedules with an average time of less than 2 seconds while the buffer IP failed to produce a single schedule in 90 minutes. In all of these experiments, the NICE scheduler overwhelmingly outperformed the RL scheduler from which it was derived. These outcomes indicate that NICE can harness various advantages of IP and RL scheduling to build a hybrid approach that improves on both methods used independently.

In the less-constrained baseline scheduling environment, NICE performed worse than the buffer IP, though it still did better than the baseline. In this scenario, in a similar amount of time as NICE, the buffer IP produced perfect schedules with no disruptions after flights were delayed. This result highlights the ideal use-case for NICE: situations where approximate results are useful and the size of the integer program makes it infeasible to solve in a reasonable amount of time. The more constrained (density = 2) scenario fits this description well; the high number of flights in a shorter interval created more constraints and variables in the buffer formulation than our IP solver could handle. By contrast, NICE was able to produce a robust schedule in under 2 seconds, on average.

The move-up IP scheduler produced more robust schedules than any of the other methods, but it did not perform as strongly as the buffer IP scheduler, which entirely eliminated schedule disruptions across all percentages of flights delayed in our previous experiment. This weakness likely explains the performance of the NICE scheduler based on the move-up reward function, which showed no significant difference in schedule disruptions compared to the baseline IP scheduler. Because of the relative inefficacy of increasing the number of move-up pilots in producing robust schedules, we decided to focus our efforts on using buffers to reduce disruptions.

## 3.5 Conclusions and Future Work

In this chapter, we introduced NICE, a novel method for incorporating knowledge gleaned from reinforcement learning into an integer programming formulation. We

applied this technique to a robust crew scheduling problem, looking at the assignment of pilots to flights so as to minimize schedule disruptions due to flight delays. We used NICE to build a scheduler for this problem where the RL agent proposes weights for the selection of crew members, and the IP assigns the crew members using those weights. In our experiments, NICE outperformed both the baseline IP and the RL scheduler in creating schedules that are resistant to disruptions. Furthermore, in certain practical environments that caused the robust scheduling (buffer IP) formulation to be prohibitively slow, NICE was able to create a robust schedule in a matter of seconds.

The introduction of nonlinear objectives or constraints can deteriorate the computational performance of MIP and IP solvers. The underlying formulations are no longer integer linear programs or mixed-integer linear programs. However, the reward structure used to train an RL agent is not bound by such restrictions. While this paper used NICE to approximate linear constraints and additional variables in an IP, it would be interesting to see how the approach performs when faced with nonlinear constraints.

Finally, we have shown the efficacy of NICE in robust crew scheduling. Given that IPs have long been a mainstay of discrete optimization, we believe that the approach could be useful in addressing other scheduling problems. Understanding the types of optimization problems for which NICE is most effective is an interesting topic for further research.

# Chapter 4

# Conclusions

In this thesis we introduced two learning-based solution approaches for the crew-scheduling problem: reinforcement learning for optimizing different metrics, and a hybrid approach combining reinforcement learning with integer programming for robust scheduling.

While developing an RL approach to crew scheduling, we first described the state space, action space and the reward structure. The reward structures were defined according to the objectives (minimizing overqualification and maximizing completion of training requirements) that schedulers wanted to optimize. Action masking was found to be important in improving the sample complexity of training the RL agent with the defined MDP formulation. Then we showed that the RL agents with a reward structure engineered to optimize for certain metrics were able to perform as well as their IP counterparts. Further, we showed that the RL agent could optimize for multiple metrics in a single formulation, whereas the IP formulations required separate objective functions. It is worth noting that although the RL agent was able to perform as well as a corresponding IP formulation, it is never able to perform better than it. A major reason for this is that the IP formulation has a global view of the problem over the entire planning period, whereas the RL agent has a local view of the problem and cannot make any changes to the actions it took in the beginning of the episode.

To this end we combined reinforcement learning with integer programming to

introduce NICE. We applied this technique to a robust crew scheduling problem, looking at the assignment of pilots to flights so as to minimize schedule disruptions due to flight delays. We used NICE to build a scheduler for the problem where the RL agent proposed weights for the selection of crew members, and the IP assigned the crew members using those weights. We showed that NICE outperformed both the baseline IP and the RL scheduler in creating schedules that were resistant to disruptions. Furthermore, in certain practical environments that caused the robust scheduling (buffer IP) formulation to be prohibitively slow, NICE was able to create a robust schedule in a matter of seconds. Given that IPs have long been a mainstay of discrete optimization, a promising direction for further research would be the application of NICE to other scheduling problems.

# Appendix A

# Proofs

## A.1 Proof of Simplified PPO Objective Function

This proof was taken from the SpinningUp Repository [1]. The PPO objective function is:

$$L_{\theta_k}^{CLIP}(\theta) = \mathop{\mathbb{E}}_{s,a\sim\theta_k}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\theta_k}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A^{\theta_k}(s,a)\right)\right],$$

where $\theta_k$ are the parameters of the policy iteration $k$ and $\epsilon$ is a small hyperparameter.

**Proposition**: The PPO-Clip objective can be simplified to:

$$L_{\theta_k}^{CLIP}(\theta) = \mathop{\mathbb{E}}_{s,a\sim\theta_k}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\theta_k}(s,a), g\left(\epsilon, A^{\theta_k}(s,a)\right)\right)\right],$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & \text{otherwise} \end{cases}$$

*Proof.* Suppose $\epsilon \in (0,1)$. Define

$$F(r, A, \epsilon) = \min\left(rA, clip(r, 1-\epsilon, 1+\epsilon)A\right)$$

When $A \geq 0$,

$$
\begin{aligned}
F(r, A, \epsilon) &= \min\left(rA, clip(r, 1-\epsilon, 1+\epsilon)A\right) \\
&= A \min\left(r, clip(r, 1-\epsilon, 1+\epsilon)\right) \\
&= A \min\left(r, \begin{cases} 1+\epsilon, & r \geq 1+\epsilon \\ r, & r \in (1-\epsilon, 1+\epsilon) \\ 1-\epsilon, & r \leq 1-\epsilon \end{cases}\right) \\
&= A \min \begin{cases} \min(r, 1+\epsilon), & r \geq 1+\epsilon \\ \min(r, r), & r \in (1-\epsilon, 1+\epsilon) \\ \min(r, 1-\epsilon), & r \leq 1-\epsilon \end{cases} \\
&= A \min \begin{cases} 1+\epsilon, & r \geq 1+\epsilon \\ r, & r \in (1-\epsilon, 1+\epsilon) \\ r, & r \leq 1-\epsilon \end{cases} \\
&= A \min(r, (1+\epsilon)) \\
\therefore F(r, A, \epsilon) &= \min(rA, (1+\epsilon)A)
\end{aligned}
$$

When $A < 0$,

$$
\begin{aligned}
F(r, A, \epsilon) &= \min\left(rA, clip(r, 1-\epsilon, 1+\epsilon)A\right) \\
&= A \max\left(r, clip(r, 1-\epsilon, 1+\epsilon)\right) \\
&= A \max\left(r, \begin{cases} 1+\epsilon, & r \geq 1+\epsilon \\ r, & r \in (1-\epsilon, 1+\epsilon) \\ 1-\epsilon, & r \leq 1-\epsilon \end{cases}\right) \\
&= A \min \begin{cases} \max(r, 1+\epsilon), & r \geq 1+\epsilon \\ \max(r, r), & r \in (1-\epsilon, 1+\epsilon) \\ \max(r, 1-\epsilon), & r \leq 1-\epsilon \end{cases} \\
&= A \min \begin{cases} r, & r \geq 1+\epsilon \\ r, & r \in (1-\epsilon, 1+\epsilon) \\ 1-\epsilon, & r \leq 1-\epsilon \end{cases} \\
&= A \max(r, (1-\epsilon)) \\
\therefore F(r, A, \epsilon) &= \min(rA, (1-\epsilon)A)
\end{aligned}
$$

Summarising for all cases,

$$F(r, A, \epsilon) = \min(rA, g(\epsilon, A)),$$

where, as before, $g(\epsilon, A) = (1+\epsilon)A$ for $A \geq 0$, and $g(\epsilon, A) = (1-\epsilon)A$ for $A < 0$. $\square$

The intuition from this: if a given state-action pair has negative advantage $A$, the optimisation wants to make $\pi_\theta(a|s)$ smaller, but no additional benefit to the objective function is conferred by making $\pi_\theta(a|s)$ smaller than $(1-\epsilon)\pi_{\theta_k}(a|s)$. If a state-action pair has positive advantage $A$, the optimisation wants to make $\pi_\theta(a|s)$ larger, but no additional benefit is gained by making $\pi_\theta(a|s)$ larger than $(1+\epsilon)\pi_{\theta_k}(a|s)$

# Appendix B

# Data

## B.1 Dataset

The data was obtained from multiple databases from the US Air Force, which includes Aviation Resource Management System (ARMS), Graduate Training Integration Management System (GTIMS) and Global Reach. The data sources include historical flights (dates/type/crew type/etc.), crew requirements for each type of flight, pilot training requirements, pilot evaluation dates, pilot unavailability, and pilots' qualifications. The following section is a more detailed description of the dataset used.

### B.1.1 Flight Data

There are three different types of flight events: simulators, locals and missions. These flights can be broken down further into different flight types: Air Land (AL), Airdrop (AD), and Special Operations Low Level II (SOLL II). AL flights are the most common type of flight and do not require special qualified pilots, whereas the AD and SOLL II flights require the pilots to have special qualifications. Additionally, all types of flights may require Air Refueling (AR), if the flight happens to have a long flying time.

To fly these flights, there are two types of crews: augmented and basic. Augmented

crew flights require a minimum of three pilots to fly a flight and basic crew flights have a minimum requirement of two pilots. Within the augmented crew types, there are special augmented designations if the flight requires AD, SOLL II, or AR. Thus, the pilots with special qualifications are required for these more specialized flights. To create schedules, the schedulers are given the date, start time, and type of flight training. And each mission, has a set of attributes associated with it like required crew, special requirements, itinerary, schedule of stops and locations, etc.

### B.1.2 Pilot Data

The two most imporatant attributes assiciated with each pilot in the squadron is their unavailability and qualification ranks. Unavailability can be due to various reasons like leave, temporary duties (TDYs), additional duties, appointments, etc. The unavailability for the pilots is described with a start and end date. The qualification rank is generally denoted by a five-character string (FPQC5, FPCC5, etc.). Here the first three letters denote their Air Land (AL) Qualification, the fourth letter is their training level and the last letter is their special duty qualification. The training level is generally denoted designated as level A, B or C and is set by the squadron commander to establish the more experienced pilots. The special duty qualifications are for flights/missions that require Air Drop (AD), Special Operations Low Level II (SOLL II) or other complext miscellaneous tasks. The pilots in the dataset have four different groups of qualifications ranging from highest to lowest as: Evaluator Pilots (EP), Instructor Pilots (IP), Mission Pilots (MP) and Flight Pilots (FP). The dataset consists of 86 pilots where there are 8 EPs, 37 IPs, 8 MPs, 33 FPs. Along with qualification ranks and unavailability, each pilot has training requirements which get renewed periodically as each training requirement has a different evaluation date and the renewal of these requirements assures that the pilots are current in their qualification. Also discussed above in regards to personnel data are the pilot's semi-annual or periodic training requirements and their different evaluation dates. Pilots have required types of flights they must fly semi-annually or periodically to stay current with their qualifications and ability to fly. So usually, pilots prefer to fly on flights

that help them to constantly work towards meeting their training requirements. For instance, a specific pilot would be against flying the same type of simulator over and over again because that repetition would not help them progress in their flying career. Furthermore, pilots typically like to be in control of their own schedule for various reasons, so they will voluntarily request to fly flights they are interested in or are necessary to continue to move up the ranks.

### B.1.3 Assumptions

In this thesis we make a few assumptions to simplify the proposed methods. First, we assume each pilot can only fly one event per day. In reality, a pilot can fly two simulators in the same day, but it does not happen very often and really only happens when a specific squadron is struggling to find available pilots. In determining pilot availability, we also have to account for crew rest, which is the rest time a pilot is legally required to take after completing a flight. There is typically no crew rest for simulator flights, but for training and mission flights, the necessary crew rest can be from hours to days. We assume crew rest is taken care of for the locals based on our first assumption that only one flight can be flown per day. However, for the missions, we are given historical data on how long a pilot is on crew rest for based on the day the postmission debrief occurs, so we assume that accounts for crew rest for those flights. Scheduling a future event has a projected duration, but of course this time is truly uncertain. Thus, adding in a buffer time, like in the commercial airline industry, and using crew rest regulations allow us to obtain a projected duration that a pilot is unavailable both for the flight and the accompanied crew rest.

# Appendix C

# Hyperparameters for Experiments

Below is the table of hyperparameters used for our experiments for the reinforcement learning experiments:

| Hyperparameters for the RL environment | | |
|---|---|---|
| Parameter Name | Value | Definition |
| num_pilots | 87 | Number of pilots in the squadron |
| num_event_types | 13 | Number of types of events |
| use_training_req | True | Whether to use training requirements in the state representation |
| max_duration | 7 | Number of days to schedule for |
| avg_assignments_week | 80 | Average number of pilot-event assignments per week (used for random event generation) |
| flight_density | 1 | The density of flights per week, where 1 is average. (2 would be twice the average and is only used for experiments with NICE) |
| num_parallel_envs | 50 | Number of parallel environments to run to collect episodes |

| Hyperparameters for PPO | | |
|---|---|---|
| Parameter Name | Value | Definition |
| steps_per_epoch | 4000 | Number of steps of interaction (state-action pairs) for the agent and the environment in each epoch |
| epochs | 10000 | Number of epochs of interaction (equivalent to number of policy updates) to perform |
| $\gamma$ | 0.99 | Discount factor |
| clip-ratio | 0.2 | Parameter for clipping in the policy objective. Roughly: how far can the new policy go from the old policy while still improving the objective function |
| learning_rate$_\pi$ | $3 \times 10^{-4}$ | Learning rate for the policy network |
| learning_rate$_v$ | $1 \times 10^{-3}$ | Learning rate for the value function network |
| train_pi_iters | 80 | Maximum number of gradient descent steps to take on policy loss per epoch |
| train_v_iters | 80 | Maximum number of gradient descent steps to take on value function loss loss per epoch. |
| $\lambda$ | 0.97 | Lambda for GAE-Lambda |
| target_KL | 0.01 | KL-divergence allowed between the new and old policies after an update |
| eval_freq | 10000 | Number of steps after which the policy is evaluated |
| num_test_rollouts | 10 | Number of episodes for which the policy is evaluated |

# Bibliography


[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. *GitHub repository*, 2018.

[2] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2019.

[3] JP Arabeyre, J Fearnley, FC Steiger, and W Teather. The airline crew scheduling problem: A survey. *Transportation Science*, 3(2):140–163, 1969.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[5] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming Series B*, 98:49–71, 2003.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[7] Jan K. Brueckner, Achim I. Czerny, and Alberto A. Gaggero. Airline mitigation of propagated delays via schedule buffers: Theory and empirics. *Transportation Research Part E: Logistics and Transportation Review*, 150:102333, 2021.

[8] Alberto Caprara, Paolo Toth, Daniele Vigo, and Matteo Fischetti. Modeling and solving the crew rostering problem. *Operations Research*, 46(6):820–830, 1998.

[9] Weijia Chen, Yuedong Xu, and Xiaofeng Wu. Deep reinforcement learning for multi-resource multi-machine job scheduling. *CoRR*, abs/1711.07440, 2017.

[10] Christopher Ho-Yen Chin. Disruptions and robustness in air force crew scheduling. Master's thesis, MIT, 2021.

[11] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. Cellular network traffic scheduling with deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[12] Todd E Combs and James T Moore. A hybrid tabu search/set partitioning approach to tanker crew scheduling. *Military Operations Research*, pages 43–56, 2004.

[13] Emily Craparo, Mumtaz Karatas, and Dashi I. Singham. A robust optimization approach to hybrid microgrid operation using ensemble weather forecasts. *Applied Energy*, 201:135–147, 2017.

[14] George B. Dantzig. Letter to the Editor — A Comment on Edie's "Traffic Delays at Toll Booths". *Journal of the Operations Research Society of America*, 2(3):339–341, 1954.

[15] Tiago Salgado de Magalhães Taveira-Gomes. Reinforcement learning for primary care appointment scheduling. Master's thesis, University of Porto, 2017.

[16] Balaji Gopalakrishnan and Ellis L Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140(1):305–337, 2005.

[17] Nicolas Heess, Greg Wayne, David Silver, Timothy P. Lillicrap, Yuval Tassa, and Tom Erez. Learning continuous control policies by stochastic value gradients. *CoRR*, abs/1510.09142, 2015.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[19] Craig Gibson Honour. A computer solution to the daily flight schedule problem. Master's thesis, Naval Postgraduate School, 1975.

[20] Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E. Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. *CoRR*, abs/2107.10847, 2021.

[21] S. Jacobs, Roger. Optimization of daily flight training schedules. Master's thesis, Naval Postgraduate School, 2014.

[22] Luke Kenworthy, Siddharth Nayak, Christopher Chin, and Hamsa Balakrishnan. Nice: Robust scheduling through reinforcement learning-guided integer programming, 2021.

[23] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466, 1952.

[24] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[25] Matthew J Koch. Air force crew scheduling: An integer optimization approach. Master's thesis, MIT, 2021.

[26] Niklas Kohl and Stefan E. Karisch. Airline Crew Rostering: Problem Types, Modeling, and Optimization. *Annals of Operations Research*, 127:223–257, 2004.

[27] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000.

[28] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019.

[29] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

[30] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2020.

[31] D Macey. Fallon range training complex flight event timetabling. Master's thesis, Naval Postgraduate School, 2017.

[32] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, page 50–56, New York, NY, USA, 2016. Association for Computing Machinery.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, Andrei A. Rusu, J. Veness, Marc G. Bellemare, A. Graves, Martin A. Riedmiller, A. Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, A. Sadik, Ioannis Antonoglou, Helen King, D. Kumaran, Daan Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[34] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[36] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2021.

[37] MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Deep reinforcement learning for solving the vehicle routing problem. *CoRR*, abs/1802.04240, 2018.

[38] Christos Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[39] Martin L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[40] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[41] H. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[42] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[44] Sergey Shebalov and Diego Klabjan. Robust airline crew pairing: Move-up crews. *Transportation science*, 40(3):300–312, 2006.

[45] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[46] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[47] J. Slye, Robert. Optimizing training event schedules at naval air station fallon. Master's thesis, Naval Postgraduate School, 2018.

[48] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[49] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[50] Dori van Hulst, Dick Den Hertog, and Wim Nuijten. Robust shift generation in workforce planning. *Computational Management Science*, 14(1):115–134, 2017.

[51] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John P. Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.

[52] Robin Vujanic, Paul Goulart, and Manfred Morari. Robust optimization of schedules affected by uncertain events. *Journal of Optimization Theory and Applications*, 171(3):1033–1054, 2016.

[53] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72:1264–1269, 2018. 51st CIRP Conference on Manufacturing Systems.

[54] B. L. Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947. Publisher: [Oxford University Press, Biometrika Trust].

[55] Cheng-Lung Wu. Inherent delays and operational reliability of airline schedules. *Journal of Air Transport Management*, 11(4):273–282, 2005.

[56] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving the travelling salesman problem. *CoRR*, abs/1912.05784, 2019.

[57] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. Mastering complex control in MOBA games with deep reinforcement learning. *CoRR*, abs/1912.09729, 2019.

[58] Emre Yolcu and Barnabas Poczos. Learning local search heuristics for boolean satisfiability. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[59] Rongkai Zhang, Anatolii Prokhorchuk, and Justin Dauwels. Deep reinforcement learning for traveling salesman problem with time windows and rejections. In

*2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[60] Zizhen Zhang, MengChu Zhou, and Jiahai Wang. Construction-based optimization approaches to airline crew rostering problem. *IEEE Transactions on Automation Science and Engineering*, 17(3):1399–1409, 2020.