Final Year Project

Quick Roster: A Volunteer Scheduling System

Thomas Creavin

Student ID: 17311103

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Dr. Deepak Ajwani

UCD School of Computer Science

University College Dublin

August 3, 2022

Table of Contents

1 **Project Specification** 4

1.1 Core requirements 4

1.2 Advanced requirements 4

2 **Introduction** 5

3 **Related Work and Ideas** 6

3.1 Volunteer Rostering Applications 6

3.2 Scheduling Applications 7

3.3 Nurse Scheduling Problem 8

3.4 Machine Learning Combinatorial Optimization 10

3.5 Serverless Applications 12

3.6 Summary 13

4 **Mixed-Integer Programming Formulation of the NSP** 14

4.1 Overview 14

4.2 Mixed Integer Programming Formulation 14

5 **Data Considerations** 17

5.1 Overview 17

5.2 Schedule Generation 17

5.3 Scheduling Benchmarks 18

5.4 User Volunteered Schedules 18

6 **Methodology and Implementation Details** 19

6.1 Nurse Scheduling Problem Solver 19

6.2 Back end 19

6.3 Front end 23

6.4 Data Preparation 24

6.5 Machine Learning 26

7 **Evaluation** 33

7.1 MIP Formulation of the NSP Algorithm Evaluation 33
7.2 Supervised Search Space Pruning Model Evaluation 33
8 Conclusions and Future Work 37
8.1 Conclusion 37
8.2 Future Work 37
9 Appendix 44
9.1 Link to the GitLab Repository 44
9.2 Full Feature Set 44
9.3 Supplementary Figures 46
9.4 Project Workplan 52

Abstract

Volunteering is widely practiced in Ireland. Most volunteering campaigns involve some degree of scheduling to organise personnel. For small events, the scheduling can be as simple as everyone agreeing to show up to the fundraiser at noon. However, more sophisticated scheduling is needed for large events, conferences, multi-day fundraisers, etc. Half of all volunteering in Ireland is work carried out directly by individual groups rather than through organisations. These volunteers are unlikely to have access to bespoke scheduling tools.

This project addresses the need for a straightforward and scalable volunteering application that caters to the scheduling constraints faced by volunteering groups and organisations in Ireland by creating a severless web application for rostering volunteers. The application uses a mathematical formulation capable of rostering volunteers for a wide range of scenarios and is implemented in a state-of-the-art mathematical programming solver. The solver is optimized by preprocessing the data with a supervised search space pruning model. The optimized solver reduces the average solve time by almost 50% without noticeably affecting the solution quality. The time reduction can be as great as 92% – reducing the solve time of a complex schedule from 13 hours to just 1 hour. This enables the application to solve even the most challenging problems in a reasonable time frame.

Chapter 1: **Project Specification**

Scheduling volunteers is a difficult task as there is a lack of straightforward applications that address the real-world scheduling needs of volunteer organisations. Typical scheduling systems such as Timecounts[1], When I Work[2], and OptaPlanner[3] require a non-trivial amount of setup and are designed to handle simple scheduling scenarios.

Scheduling volunteers introduces a new set of constraints [1] that are not considered by general scheduling algorithms. A competent volunteer scheduling algorithm should be capable of accommodating volunteers' varied availability; be able to schedule volunteers across concurrent shifts, and allow volunteers to do multiple shifts of various lengths per day. Additionally, a scheduling system for volunteers should be capable of quickly creating schedules for a large number of volunteers with minimal setup.

The goal of this project is twofold: to develop a scheduling algorithm that caters to the needs of volunteer organisations and to create an application that can utilise this algorithm to generate rosters.

Metaheuristic algorithms are predominantly employed to optimize this class of scheduling problem. In contrast to typical approaches, this project will use mathematical programming and will leverage modern machine learning techniques such as supervised search space pruning [2] to create a novel volunteer scheduling algorithm. Initially, mixed-integer techniques will be explored before experimenting with machine learning frameworks.

The machine learning model will be trained and evaluated using an artificial data set and a nurse scheduling problem data set provided by Scheduling Benchmarks[4].

1.1 Core requirements

- Identify a variant of the nurse scheduling problem suitable for scheduling volunteers.
- Develop a mixed-integer linear programming solver for the chosen nurse scheduling problem variant.
- Evaluate the effectiveness of the solver.
- Develop an application for scheduling volunteers.

1.2 Advanced requirements

- Employ supervised search space pruning to optimize the scheduling algorithm.

[1]https://timecounts.org/
[2]https://wheniwork.com
[3]https://www.optaplanner.org/
[4]http://www.schedulingbenchmarks.org/index.html

Chapter 2: **Introduction**

Volunteering is a popular activity in Ireland. The results of the 2013 Quarterly National Household Survey show that 28% of Irish people volunteer and half of all volunteering was work carried out directly by individuals (54.7% of hours worked) rather than through organisations (45.3%) [3].

Every volunteer group and organisation must deal with scheduling. Simple events can be scheduled manually or by having volunteers self-schedule themselves. As events grow in size and complexity, so does the scheduling complexity. Manually scheduling dozens of volunteers with varied availability is difficult. The problem is exacerbated by introducing additional constraints like *an experienced volunteer must be present in the morning* or *a volunteer with access to a car is needed in the afternoon.*

I know from experience that scheduling volunteers is a time-consuming and cumbersome process; it has been my mantle for the past two years to create rosters for UCD Netsoc's[1] Freshers' Week stand[2]. In this case, the scheduling process entails collecting a copy of each of the fourteen volunteers' timetables, perusing the timetables, scheduling the most exclusive volunteers, consulting the timetables, scheduling the people with greater availability, flicking through the timetables, balancing the workload, and re-checking the timetables for violations.

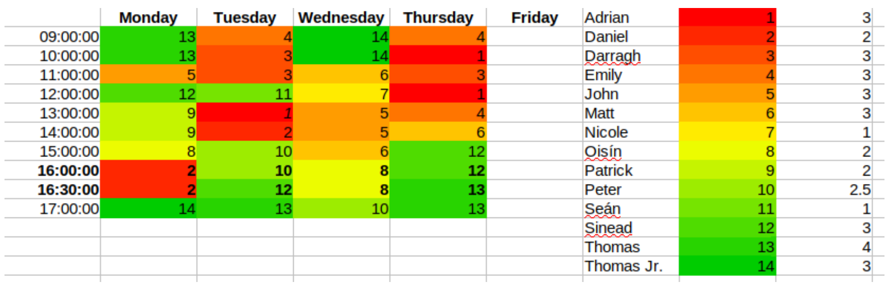
The roster I produced for Freshers' Week 2019 is presented in figure 2.1. It could be better; for instance, Nicole and Seán only work one shift while most work three shifts. The schedule meets the hard constraints that those working the opening and closing shifts must have access to the society locker, and the soft constraint that experienced committee members should work the busy midday shifts. If you would like to see a complex roster with concurrent shifts, I have included a roster made for the SISTEM[3] 2020 conference in figure 9.2 in the appendix.

Volunteer organisers would benefit from a volunteer rostering tool that can automatically schedule volunteers. This project aims to create an application that would allow volunteers to submit their availability and information in a structured way to the system. The organiser could quickly configure a roster for a single or multi-day event. This event can have multiple roles e.g volunteers could be scheduled to bring supplies to a stand or to supervise a stand. The application can then use the availability of the volunteers to schedule people to shifts. The whole process, from receiving the volunteers' availability to generating the roster, should take a few minutes.

[1]https://netsoc.com/ – UCD Internet & Computing Society

[2]A society registration stand which is staffed for the duration of trimester one week two

[3]https://sistem.intersocs.ie/



	Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	Adrian	1	3
09:00:00	13	4	14	4		Daniel	2	2
10:00:00	13	3	14	1		Darragh	3	3
11:00:00	5	3	6	3		Emily	4	3
12:00:00	12	11	7	1		John	5	3
13:00:00	9	*1*	5	4		Matt	6	3
14:00:00	9	2	5	6		Nicole	7	1
15:00:00	8	10	6	12		Oisín	8	2
16:00:00	**2**	**10**	**8**	**12**		Patrick	9	2
16:30:00	**2**	**12**	**8**	**13**		Peter	10	2.5
17:00:00	14	13	10	13		Seán	11	1
						Sinead	12	3
						Thomas	13	4
						Thomas Jr.	14	3

Figure 2.1: 2019 Roster for Netsoc's Freshers' Week stand

Chapter 3: Related Work and Ideas

To begin, I review existing volunteer rostering software. Next, I discuss the nurse scheduling problem and explain how research in this area is relevant to the scheduling of volunteers. I describe how machine learning techniques have the potential to enhance a nurse scheduling problem solver. Last, I touch on the merits of serverless technologies.

3.1 Volunteer Rostering Applications

A volunteer scheduling system is not a new idea; there are many existing volunteer rostering applications. Still, this project is unique because it can automatically schedule volunteers and it caters to a wider variety of volunteering scenarios.

Timecounts[1] is the epitome of a volunteer rostering systems. Timecounts allows an organiser to plan a single or multi-day event where qualified volunteers can choose their shifts. The management portal has an intuitive user interface that is easy to navigate and use. It offers advanced features like a volunteer directory, data insight tools, and a messaging system.

Timecounts is not a substitute for Quick Roster. It does not offer automatic scheduling; volunteers must instead sign-up for shifts (self-schedule). Timecounts limits rostering to one shift per time slot which means you cannot schedule volunteers for different tasks. Timecounts does not allow for skill relaxing; only those with the exact skill can sign-up for a shift. In contrast, Quick Roster automatically schedules volunteers, it allows for skill relaxing, and the roster can accommodate multiple tasks.

Other volunteer rostering applications include iVolunteer[2], RosterVolunteers[3], and SignUp [4]. They

[1]https://timecounts.org/
[2]https://ivolunteer.com/
[3]https://rostervolunteers.com/wp/
[4]https://signup.com

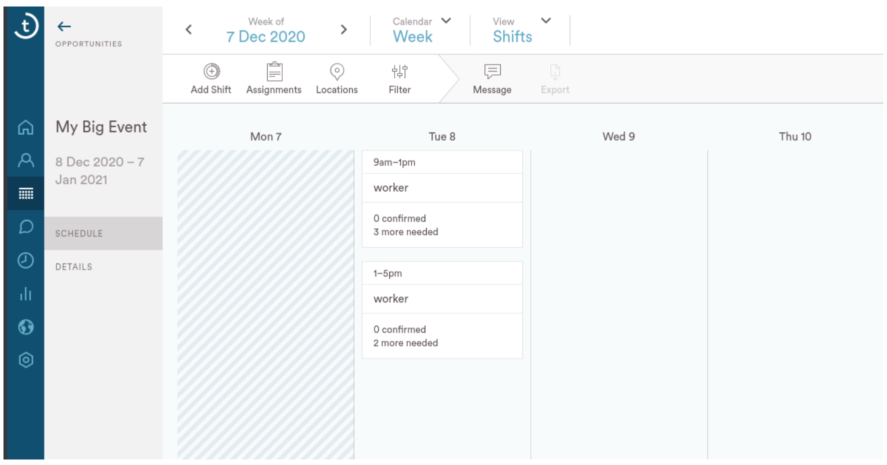


Figure 3.1: Screenshot of the Timecounts rostering interface

offer similar functionality to Timecounts, and schedule volunteers manually or with self-scheduling. I could not find a volunteer rostering application that offered automatic scheduling.

3.2 Scheduling Applications

Here, I examine employee scheduling software as employee scheduling is closely related to volunteer scheduling. The key differences are

- Employee scheduling applications are designed for business scheduling scenarios; typically employees are scheduled for single continuous shifts. Volunteers have limited availability and can only be scheduled in bursts e.g student volunteers availability is determined by their college timetable
- Employees have distinct roles that restrict them to specific tasks. Volunteers have more fungible roles and can be assigned to a range of tasks.
- Employee scheduling software can take a long time to set up but the application will save time in the long run. Volunteering groups run events less frequently and sometimes with different volunteers. A volunteering group may not benefit from the application if the setup time for the application is long.

When I Work[5] is the best employee scheduling tool I have found. Its main features include: an intuitive UI, automatic scheduling at a cost of $2.50 per user, allows employees to be scheduled for different tasks, and it has a companion app that allows employees to request time off.

This would not be suitable for volunteer scheduling because When I Work only allows an employee to do one shift per day, a shift's skill requirements aren't relaxable, and since the employees can't specify all their availability upfront, it is time-consuming to set up for a one-off event.

OptaPlanner[6] is an open-source AI constraint solver. It offers efficient automatic employee scheduling – the solver leverages algorithms such as Tabu Search, Simulated Annealing, Late Acceptance, and other meta-heuristics to perform quick scheduling. It provides a simple user interface for rostering employees.

OptaPlanner is a good employee scheduling tool but is not suitable for volunteer scheduling: employees can only be scheduled for a single block of time once per day; skills in the same skill group are interchangeable but it doesn't allow for skill relaxing outside of the skill group; it does not allow skill quotas for a task e.g it is not possible to schedule a master butcher and two trainees to the butcher counter; the organiser must input all the employees' availability manually, and it's a java application that must be compiled and configured.

OptaPlanner also offers a Java planning engine that can be configured to the scheduling problem at hand. A volunteer-orientated fork could be produced to satisfy this project's niche. This would require modifying the default constraints, changing the interface, building an interface for volunteers to select their availability, tasks handling would have to be fundamentally changed to allow further specification and the UI would have to change to reflect this. The planner would have to be benchmarked to verify that the planning engine can still deliver adequate performance. I have decided against this. It would be simpler to create a volunteer scheduler from the ground up than to largely reconfigure the OptaPlanner planning engine.

[5] https://wheniwork.com
[6] https://www.optaplanner.org/

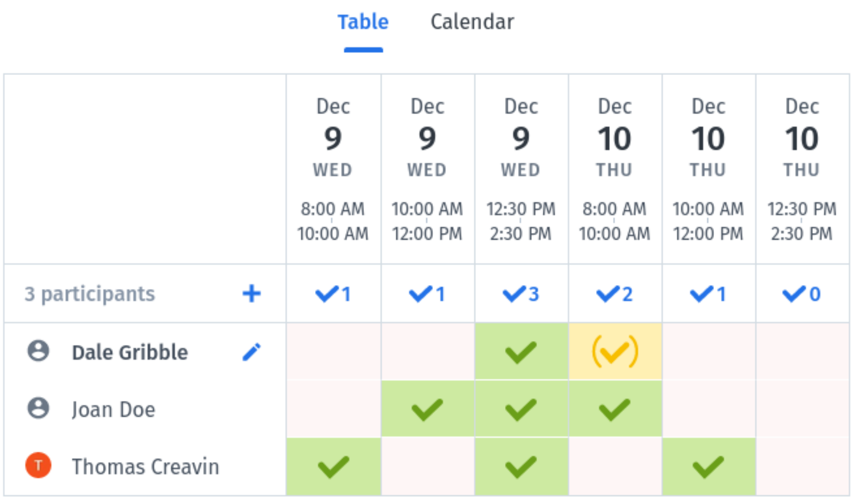


Table Calendar

	Dec 9 WED 8:00 AM 10:00 AM	Dec 9 WED 10:00 AM 12:00 PM	Dec 9 WED 12:30 PM 2:30 PM	Dec 10 THU 8:00 AM 10:00 AM	Dec 10 THU 10:00 AM 12:00 PM	Dec 10 THU 12:30 PM 2:30 PM
3 participants +	✓1	✓1	✓3	✓2	✓1	✓0
Dale Gribble			✓	(✓)		
Joan Doe		✓	✓	✓		
Thomas Creavin	✓		✓		✓	

Figure 3.2: Screenshot of the Doodle meeting-time selection interface

Doodle[7] is a popular web application for deciding meeting times. A meeting organiser creates a range of possible meeting times and shares a Doodle link with the participants. Each participant selects the timeslots which best suits them. The meeting organiser can then select the most appropriate meeting time.

I wish to model my application after Doodle: it is the gold standard for a self-scheduling application in terms of its user experience. There is no need to download and configure an application; the participants can fill out their availability which takes the burden off the meeting organiser, and it's free.

3.3 Nurse Scheduling Problem

The nurse scheduling problem (NSP) has been studied as far back as the 1960s [4]. The NSP is a combinatorial optimization problem concerned with optimally assigning nurses to shifts in a hospital ward while adhering to constraints such as *nurses can only work one shift per day*, and *there must be a twelve-hour rest period between shifts*. The assignment can be optimized for many objectives such as maximizing shift coverage (i.e ensuring each shift is properly staffed) or minimizing the number of nurse hours. The volunteer scheduling problem can be phrased as a NSP whereby the nurses are volunteers, wards are event locations, and many of the constraints of the NSP are relevant to the volunteer scheduling program.

The NSP is a diverse problem; it can have many different configurations depending on the scenario. De Causmaecker and Vanden Berghe devised a popular classification framework for the NSP that uses an $\alpha|\beta|\gamma$ notation to describe instances of the problem [5]. The constraint category α refers to personnel requirements such as availability or maximum hours per week; the category β refers to the schedule characteristics such as *do shifts overlap*, *are shifts time intervals*, *are shifts periods* e.g early shift or late shift; and γ refers to the optimization objective.

The variant of the NSP that I would like to address is: $A(a)BN|TN|LP$ i.e the solution must accommodate unavailabilities ($A(a)$); shifts must be equitably distributed (B); nurses can have many skills (N); the schedule comprises of time intervals (T); there is a variable number of shifts (N); the violations of personnel constraints (P) and of coverage constraints (L) should be

[7]https://doodle.com/en/

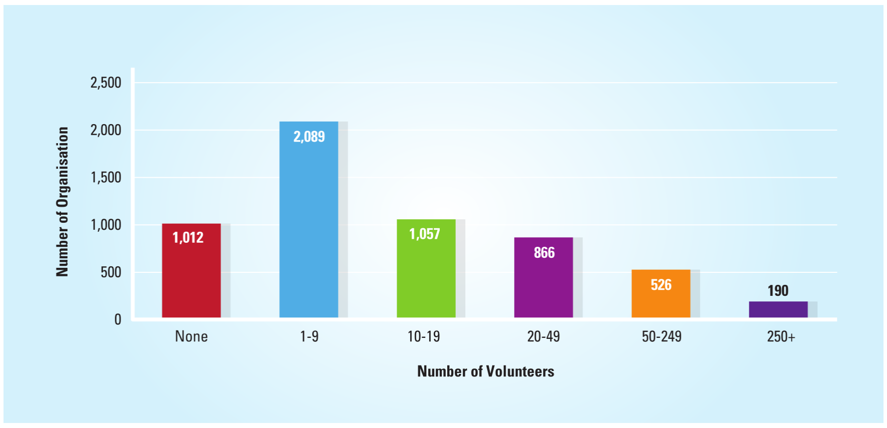


Figure 3.3: Number of Organisations by Number of Volunteers

Source: Indecon analysis of Charities Regulator data [13]

minimized.

In addition to these constraints, the algorithm should allow for concurrent shifts and allow volunteers to do multiple shifts of various lengths per day e.g it should be possible to schedule a volunteer to be in Location A for two hours in the morning and location B for an hour in the afternoon. Concurrent shifts are like speaker tracks at a conference: at nine am a speaker could be presenting in the main hall (track one) or they could be participating in a panel discussion in the alternate hall (track two).

None of the papers I have read address the new set of constraints (concurrent shifts and multiple shifts per day) or $A(a)BN|TN|LP$. It makes sense that the new constraints I have described are not catered for already: the nurse schedules are created on a per ward basis and nurses do at most a single shift per day. This project must create a custom solver to address this problem.

Thanks to decades of research many solving techniques exist for the NSP such as mixed-integer programming [6], local search algorithms [7], column-generation [8], memetic algorithm [[9], [10]], genetic algorithms [11], tabu-search [12] and more.

This begs the question, what is the right approach for this project? The correct approach must be able to handle many volunteers and can be developed within this project's time frame. Figure 3.3 from Registered Irish Charities [13] shows that over half of charitable organisations (54.8%) indicated they had between one and 20 volunteers. A further 15.1% of organisation have 20-49 volunteers. An ideal solver should schedule up to fifty volunteers.

Of the approaches I mentioned earlier, metaheuristics are most commonly employed to solve complex scheduling problems. Metaheuristic solvers like ANROM [14] can scalably handle large problems. This comes at a cost; as you can see from figure 3.4, these solvers have a sophisticated architecture. A good metaheuristic solver is not achievable within this project's time frame. Mixed-integer programming solvers on the other hand are simpler to create. Once the constraints of the problem and the objective function are clearly defined, the problem can easily be formulated using an integer programming solver like Gurobi or Xpress. MIP has been shown to provide quick and optimal solutions for small instances of the NSP [[1], [15]]. However, as the problem size increases, the MIP performance degrades sharply, unlike meta-heuristics which continue to scale.

This report must investigate if the performance of a MIP solver can be enhanced to tackle larger scheduling problems. Lin et al. [16] had a similar problem. The researchers were using MIP models to find the optimal production schedule of a set of generators while adhering to certain constraints. However, the MIP solver was not quick enough. The researchers could create quick approximate

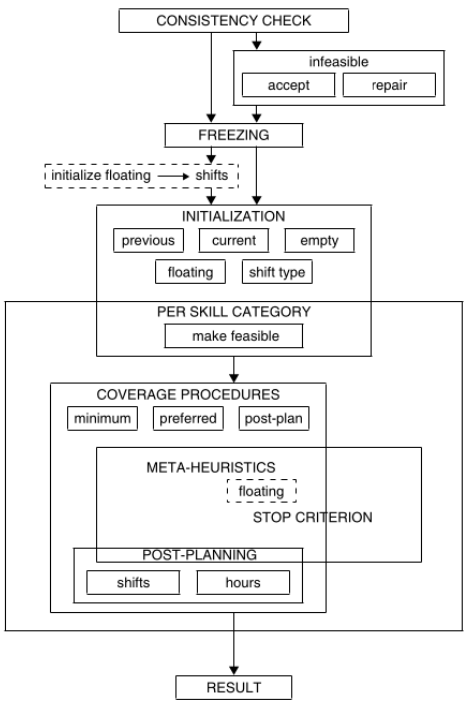


Figure 3.4: Overview of ANROM's solution framework

Source: Adapted from Handbook of Scheduling: Algorithms, Models, and Performance Analysis [4]

solutions to the MIP by using a linear programming relaxation model which is typically quicker to solve but the solutions proved too inaccurate. The U.S. Department of Energy employed machine learning to remedy this. Their model was trained on the features which best predict the link between coefficients in the MIP model, the LPR solution, and the optimal solution of the MIP problem. This resulted in a 78% reduction in the discrepancies between LPR and MIP solutions.

I would like to investigate if it is possible to improve the MIP solution to a NSP by applying machine learning.

3.4 Machine Learning Combinatorial Optimization

In this section I explore how machine learning can augment traditional algorithms; I discuss some of these recent developments and works, particularly in the area of combinatorial optimization; and I investigate if these machine learning techniques can be applied to the nurse scheduling problem.

3.4.1 Learning-Augmented Algorithms

Learning-augmented algorithms are modified versions of classical algorithms that use machine learning to adapt their behaviour to the properties of the input distribution [17]. This is a nascent research area that has already seen impressive developments. As an illustrative example of how machine learning can augment traditional data structures, let's look at the bloom filter. A bloom filter is a space-efficient data structure that can confirm if an element is not in a set with certainty

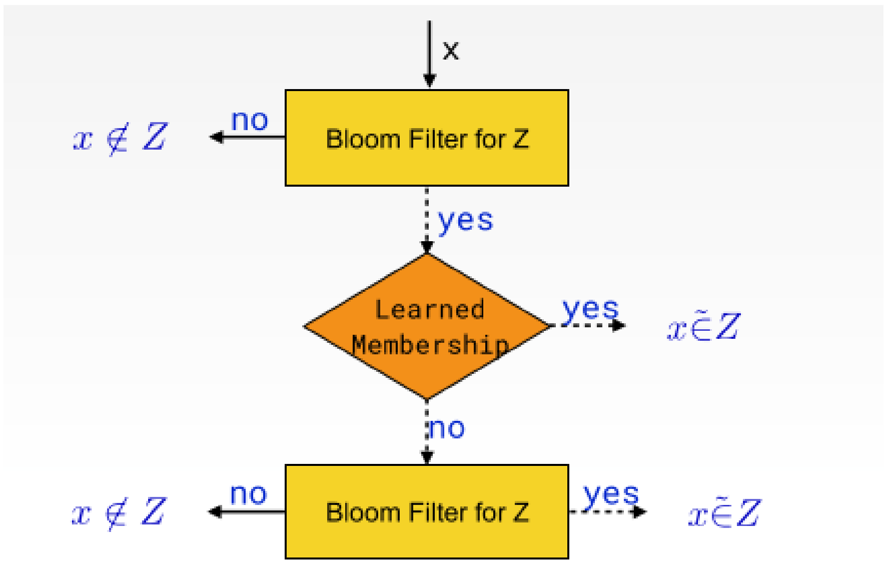


Figure 3.5: A learned bloom filter: Tier one outputs no if x it is absent from the set. Tier two outputs yes if it predicts x is in the set. Tier three asserts x is absent or predicts it might be in the set.

Source: Adapted from What Can ML Do For Algorithms? [19]

and predict if an element might be in the set. Mitzenmacher [18] utilizes a learned oracle to improve the performance of a bloom filter. A machine learning model can predict if an element belongs to a set by training the model on the input data. The learned model works in tandem with the bloom filter to improve the filter accuracy, as shown in figure 3.5.

Machine Learning has been used to augment other algorithms and data structures. Balcan et al. [20] apply machine learning to a tree search algorithm configuration and showed a nearly optimal mixture of branching rules can be learned. Kraska et al. [21] show that B-Tree indexing can be improved by using indexes learned from the distribution of the data to be indexed. Purohit et al. [22] use machine-learned predictions to improve the performance of online algorithms used to solve the classic ski rental problem and the non-clairvoyant job scheduling problem.

3.4.2 Machine Learning Combinatorial Optimization Techniques

The existing machine learning techniques for solving combinatorial optimization can broadly be divided into three categories: supervised learning, unsupervised learning, and reinforcement learning.

Supervised machine learning models automatically learn relationships or patterns between a set of features and a target feature from a repository of labeled data [23]. For instance, Vinyals et al. [24] created a new supervised deep learning neural architecture, Pointer Net, to learn approximate solutions to the Travelling Salesman problem and two other problems.

Unsupervised models learn underlying relationships by grouping unlabelled data into clusters and using those clusters to categorize data [25]. For example, Probst et al. [26] integrated a Restricted Boltzmann Machine into an Estimation of Distribution Algorithm to solve single object combinatorial problems.

Reinforcement learning is learning how to map situations to actions in order to maximise a reward. This is achieved through exploring the data to find new patterns, and through leveraging

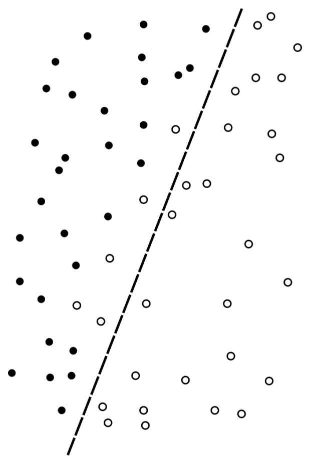


Figure 3.6: An illustration of the pruning framework in action. The black-filled circles represent elements in optimal subset while the white-filled circles represent the elements not in the optimal subset. The dashed line indicates the decision boundary.

Source: Modeled after Figure 1 from Learning fine-grained search space pruning and heuristics for combinatorial optimization [2].

the knowledge it has built up through past explorations [27]. For example, Dai et al. [28] created a reinforcement learning framework for automatically designing greedy heuristics for hard combinatorial optimization problems on graphs.

3.4.3 Supervised Pruning

The approach I'm most interested in using is supervised pruning. Lauri et al. [2] propose a novel framework for leveraging machine learning techniques to scale up exact combinatorial optimization algorithms. The researchers apply the framework to the classical maximum clique enumeration problem. The framework learns the most important local features for determining the maximum clique. Using these features, the framework (non-exhaustively) prunes elements of the problem that it has predicted are not in the optimal subset, as seen in figure 3.6. This can greatly reduce the problem size which allows solvers to handle larger instances.

This framework has the potential to be applied to the nurse scheduling problem. Through feature engineering, the most important features for determining who will be assigned to each shift can be identified. I can train a supervised machine learning model using these features to predict who won't be scheduled for a shift. This has the effect of reducing the sample space. Using these predictions, the MIP solver can create a solution faster. If the predictions are good, the MIP solver will be capable of handling larger instances of the problem.

3.5 Serverless Applications

There is an increasing trend to move applications to the cloud, and for good reason. Armbrust, M.et al. [29] highlight six of the many advantages of using cloud-native infrastructure over on-premises infrastructure:

1. The appearance of infinite computing resources on demand.

2. The elimination of an up-front commitment by cloud users.
3. The ability to pay for use of computing resources on a short-term basis as needed.
4. Economies of scale that significantly reduced cost due to many, very large data centres.
5. Simplifying operation and increasing utilization via resource virtualization.
6. Higher hardware utilization by multiplexing workloads from different organizations.

Cloud-native infrastructure offers the ability to develop serverless applications. A serverless application, as defined by Jonas, E. et al. [30], must scale automatically with no need for explicit provisioning, and be billed based on usage. That paper states that for many applications, serverless computing offers a significant cost saving compared to a serverful approach. There are some resource-intensive tasks that may not suitable to use cloud functions. The paper conducted experiments that found that large-scale linear algebra computations were 3x slower and that machine learning training at scale was 7x more expensive compared to a serverful approach.

By developing Quick Roster with Amazon Web Services[8], it can benefit from the advantages of a cloud-native infrastructure. I expect the application to be used infrequently so it would be more cost-effective to develop the application using a serverless architecture. The web components will be built using AWS Lambdas. I will build the scheduler service into a Fargate[9] container. AWS Fargate is a serverless compute engine for containers that can be invoked on-demand and only bills for the compute resources used when the container is invoked i.e the application won't incur any charges while the scheduling service is idle. Fargate is used instead of Lambdas because experiments conducted by Jonas, E. et al. show that building resource-intensive services such as the scheduling service using cloud functions can be costly.

I have chosen to use the cloud vendor AWS because I am familiar with its services and it integrates with Ruby on Jets[10]. Ruby on Jets is a serverless framework that can easily create and deploy serverless applications to AWS. I intend to use this to create the AWS infrastructure.

3.6 Summary

There are many volunteer rostering applications, but none of them provide auto-scheduling or is designed for complex scheduling scenarios. Similarly, there are many well-designed scheduling applications. When I Work is an excellent application but it is not suitable for volunteer scheduling. OptaPlanner planning is a close contender but it requires too much redesign to be suitable for volunteer scheduling. A bespoke solution is needed to solve this problem.

The volunteer scheduling problem is analogous to the nurse scheduling problem. The multi-integer programming techniques used to solve the NSP can be applied to this project's problem. A MIP solution may not be capable of handling large instances of the problem. Machine learning has been applied to similar problems with great success. Supervised pruning has to potential to speed up a MIP solution so that it is capable of handling practically large instances of the problem.

[8]https://aws.amazon.com/
[9]https://aws.amazon.com/fargate/
[10]https://rubyonjets.com/

Chapter 4: Mixed-Integer Programming Formulation of the NSP

4.1 Overview

The project is concerned with creating a solver which can handle the $A(a)BN|TN|LP$ variant of the NSP. This is accomplished by expressing the problem as a MIP formulation which then enables the problem to be solved by mathematical optimization solvers like Gurobi and Fico Xpress. Mathematical solvers can produce optimal solutions to the problem which allows for the creation of accurate training data.

Before describing the formulation of the integer programming problem, it is important to highlight the distinction between days, tracks, shifts, and skills. Consider the example in table 4.1. The organiser of this one-day event wishes to schedule their volunteers. There are two rooms to be staffed: the main theatre and the workshop room. These rooms are considered to be tracks because they have independent staffing requirements. The event runs across three time periods: 9 am, 10 am, and 11 am. These time periods are considered to be shifts. Last, the workshop track requires both novice and experienced volunteers. These volunteer qualities are considered to be skills.

To make this formulation and software implementation suitable for domains outside of volunteer scheduling, this project uses general terminology in the formulation and in the software implementation. Hence, volunteers are referred to as workers in the following formulation.

	Day 1			
	Main Theatre		Workshop Room	
	Novice	Experienced	Novice	Experienced
9 am	0	3	2	1
10 am	0	2	2	1
11 am	0	2	3	1

Table 4.1: Mock scheduling scenario that consists of one day (Day 1), two tracks (Main Theatre, Workshop Room), three shifts (9 am, 10 am, 11 am), and two skills (Novice, Experienced). The values represent the number of volunteers needed for each shift.

4.2 Mixed Integer Programming Formulation

Problem Parameters

W	Set of workers.
D	Set of days in the scheduling period.
T	Set of tracks.
S	Set of shifts.
K	Set of skills.
a_{wds}	Availability of worker w for day d, shift s.
r_{dtsk}	Required number of workers for shift (d,t,s,k).
p_{wdtsk}	Penalty for assigning worker w to shift (d,t,s,k).
m_w	Maximum number of shifts that worker w can be assigned to.
ta_w	Total shift assignments for worker w.
ss_{dtsk}	Shift slacking for shift (d,t,s,k).
ts	Total slacking for the entire schedule.
sp	Total skill penalty for the entire schdule.

The decision variable is x_{wdtsk}; it is a binary variable that denotes whether worker w is assigned to shift (d,t,s,k) or not.

Primary Constraints

1. A worker can do at most one shift at a time.
$$\sum_{t\in T}\sum_{k\in K} x_{wdtsk} \leq 1, \quad \forall w \in W, d \in D, s \in S$$
2. A worker cannot be scheduled if they are not available.
$$max(x_{wdtsk}) = a_{wds}, \quad \forall w \in W, d \in D, t \in T, s \in S, k \in K$$
3. A worker's total workload is the sum of all their assigned shifts.
$$ta_w = \sum_{d\in D}\sum_{t\in T}\sum_{s\in S}\sum_{k\in K} x_{wdtsk}, \quad \forall w \in W$$
4. A worker cannot be assigned to more shifts than their maximum shift limit.
$$ta_w \leq m_w, \quad \forall w \in W$$
5. The total skill penalty is the total cost of assigning workers with sub-optimal skills.
$$sp = \sum_{w\in W}\sum_{d\in D}\sum_{t\in T}\sum_{s\in S}\sum_{k\in K} x_{wdtsk} sp_{wdtsk}$$
6. Sometimes it is not possible to assign all the required workers so the requirements must be reduced or 'slacked'.
$$ss_{dtsk} = r_{dtsk} - \sum_{w\in W}\sum_{d\in D}\sum_{t\in T}\sum_{s\in S}\sum_{k\in K} x_{wdtsk}$$
7. The total amount of slacking should be limited as this affects the quality of the solution.
$$ts = \sum_{d\in D}\sum_{t\in T}\sum_{s\in S}\sum_{k\in K} ss_{dtsk}$$

Primary Optimization

The primary optimization creates a roster with the most coverage and with the most suitable workers by minimizing the linear combination of the number of slacked shifts and the total skill penalty for some choice of a and b.

$$Minimize\ a(ts) + b(ss)$$

Secondary Constraints

1. The average number of assignments is calculated by:

$$as = \frac{\sum_{w \in W} ta_w}{|W|}$$

2. The difference between each worker's workload and the average workload indicates the fairness of the workload.

$$diff_w = ta_w - as, \quad \forall w \in W$$

3. The total slack penalty is fixed to prevent it from being degraded in the secondary optimization.

$$ts = value(ts)$$

4. The total skill penalty is allowed to increase by up to a factor of k.

$$value(sp) \leq sp \leq k(sp)$$

Secondary Optimization

The secondary optimization improves the fairness of the roster by minimizing the difference between each worker's workload. This optimization reduces the risk of some workers being assigned an excessive number of shifts while other workers receive few assignments which can be an issue in sparse schedules.

$$Minimize \sum_{w \in W} (diff_w)^2$$

Chapter 5: Data Considerations

5.1 Overview

As a prerequisite for training a supervised model, the project needs a collection of labeled rosters to use as training data. The rosters must be relatively diverse: they must have varied dimensions in terms of the number of shifts and volunteers; the shifts must have varied staffing requirements, and volunteers must have varied availability and suitability for shifts.

There is a limited amount of benchmark data available for the NSP because problem descriptions and models vary drastically and depend on the needs of the particular scheduling scenario. As a result, there isn't an ideal data set for this variant of NSP. In the absence of benchmark data, Vanhoucke and Maenhout propose an approach for generating problem instances under a controlled design [31]. This project primarily uses their problem generation approach to create a suitable data set. Additionally, this project includes modified instances from a related NSP benchmark data set in the training data.

5.2 Schedule Generation

In Vanhoucke and Maenhout's approach, problem instances are generated using parameters that dictate the size of the problem, the preferences of the workers, and the coverage required. Based on their approach, the project uses the following parameters to generate instances: `num_workers`, `num_days`, `num_shifts`, `num_tracks`, `num_skills`, `mean_staffing_requirement`, and `mean_availability`.

The `mean_staffing_requirement` parameter dictates the average percentage of volunteers that are required for a set of shifts. The `mean_availability` parameter dictates the average percentage of shifts that volunteers are available for.

The generator uses these problem parameters to compute the following variables:

1. `worker_max_shifts` is the most amount of shifts a volunteer can be assigned to over the duration of the scheduling period. This is computed by subtracting a lognormal random variable from the total amount of shifts.
2. `worker_id_to_skill_map` maps each volunteer to a set of skills that they possess. These skills are uniformly distributed.
3. `worker_availability` indicates if a volunteer is available for a given shift. This is a Bernoulli random variable with a mean defined by the `mean_availability` parameter.
4. `shift_staffing_level_requirements` is the required number of volunteers for an individual shift. It is a Gaussian random variable; its mean is a function of the `mean_staffing_requirement` parameter.

The random variables' distribution parameters are chosen by reviewing plots of the distribution's probability density function and selecting the parameters that provide a varied and realistic spread of values. The notebook for creating these plots is included in the project repository.

5.3 Scheduling Benchmarks

Scheduling Benchmarks[1] hosts some of the most popular NSP data sets. It contains two types of problems: nurse rostering and multi-activity shift scheduling. This project is interested in using the multi-activity shift scheduling data set because it is most similar to the volunteer scheduling problem.

The data set contains 255 instances in an XML and txt format. The scheduling period spans 7, 14, 21, or 28 days; each day is broken into fifteen minute shifts; the number of workers ranges from 10 to 150; the number of tasks ranges from 1 to 19, and only a small fraction of the staff are required at any one time.

The key difference between the benchmark problem and the volunteer scheduling problem is the benchmark problem lacks skill requirements and workers do not have any availability preferences. However, these multi-activity problems can be modified to include these elements. After running some small experiments, I found that because these instances are so large and workers are sparsely scheduled, only a few of these instances can be solved within a few hours and are representative of a volunteer scheduling problem. Large sparse schedules are not too challenging when scheduling employees who work for long continuous blocks of time but volunteers can be scheduled discontinuously which results in a much larger search space.

This project had intended to make extensive use of this data set but because of the challenges mentioned above, only a few instances are included in the training data.

5.4 User Volunteered Schedules

This project has the opportunity to create a genuine volunteer scheduling data set by collecting user volunteered schedules. After a user successfully creates and solves a schedule, they are asked if their schedule can be retained by the application. If the user opts-in, identifying information like volunteer names, skill names, etc are removed and the anonymous schedule is stored securely in the cloud. This data can be reused later to improve the application.

[1] www.schedulingbenchmarks.org

Chapter 6: Methodology and Implementation Details

6.1 Nurse Scheduling Problem Solver

Earlier, this report described a formulation for the volunteer scheduling problem. The next step is to implement the formulation in a mixed-integer programming solver. There is a wide variety of open-source and propriety solvers available. This project considers using the popular Gurobi and FICO Xpress solvers. An early version of the formulation was implemented in both solvers and I found that both solvers performed similarly but Gurobi is easier to work with because it provides a range of examples and clear documentation. Additionally, I found that Gurobi models are easier to dockerize; this is an important quality because serverless compute tasks must be containerised. For these reasons, I opt to implement the formulation in Gurobi.

The codebase for the solver is written in Python. The solver and its related components are implemented using an object-orientated approach whereby schedules, rosters, worker data, shift data, skill data, and the scheduler are expressed as objects. The scheduler object takes in schedules as an argument and produces rosters. Each schedule is composed of worker data, shift data, and skill data. The roster object is a superset of the schedule object; it contains the schedule object as well as roster assignments and meta-data from the Gurobi model. I implement a preprocessing step whereby workers who are unavailable for a shift are pruned; this optimization is unrelated to the supervised search space pruning that will be experimented with later in this project.

Last, the solver needs to be dockerised so it can be easily ported to the cloud. This involves creating a main script that informs the solver where to read the input files from and where to store the output. A Dockerfile is used to put the service together and install the necessary dependencies like Gurobi and the AWS command-line interface; the Dockerfile is presented in figure 6.1.

6.2 Back end

Originally, this project planned to use the Ruby on Jets framework to deploy the Amazon Web Services (AWS) back end. After planning out the back end in finer detail, it was clear that Jets could not easily support the new design. Instead, I opt to configure the back end manually in six stages: creating the scheduling service, automating the scheduling service, user management, front end infrastructure, and advanced configuration. Because I use the AWS console to configure the service, it is difficult to review how the back end is assembled. To account for this, I describe the back end setup in great detail.

```
FROM continuumio/miniconda3
ADD environment.yml /tmp/environment.yml
RUN conda env create -f /tmp/environment.yml
RUN echo "conda activate $(head -1 /tmp/environment.yml | cut -d
    ' ' -f2)" >> ~/.bashrc
ENV PATH /opt/conda/envs/$(head -1 /tmp/environment.yml | cut -d
    ' ' -f2)/bin:$PATH
ENV CONDA_DEFAULT_ENV $(head -1 /tmp/environment.yml | cut -d' '
     -f2)
WORKDIR /usr/src/app
COPY service/ ./
ENTRYPOINT ["conda", "run", "-n", "gurobi-solver", "bash", "main
    .sh"]
```

Figure 6.1: The Dockerfile that containerises the solver service. Conda installs the required dependencies which are read from the environment.yml file. The main.sh file which bootstraps the service is launched within a Conda environment.

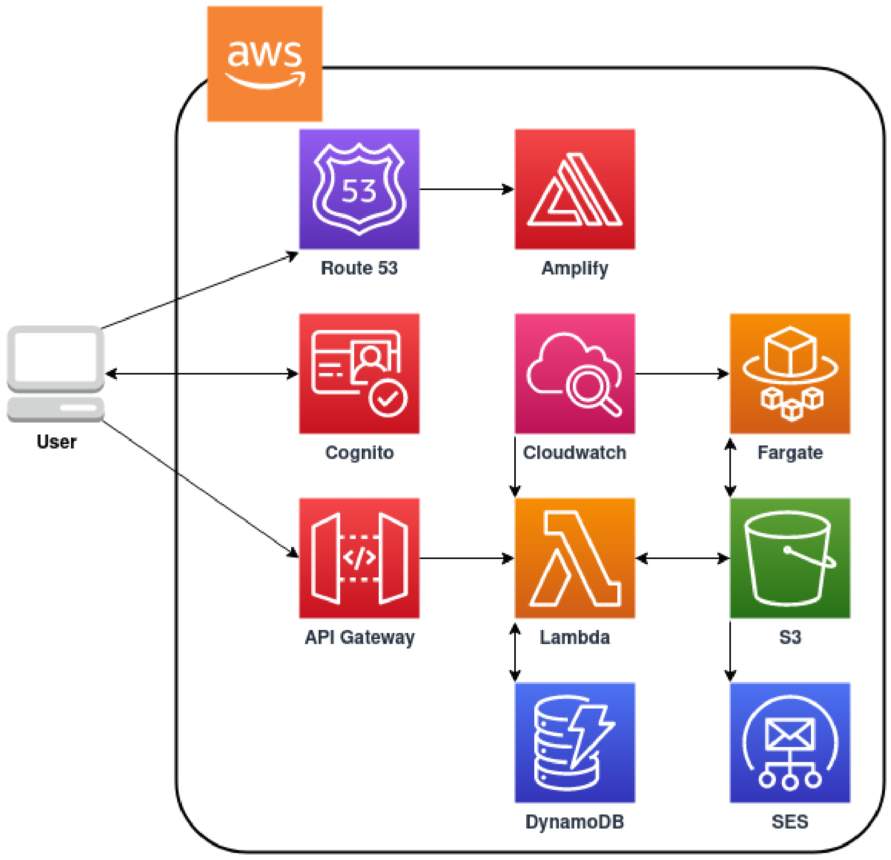


Figure 6.2: AWS Serverless Architecture for Quick Roster.

6.2.1 Creating the Scheduling Service

To export the scheduling service to Fargate, I must create each of the Elastic Container Service objects shown in figure 6.3. Fargate is a serverless compute engine for containers that is cheaper and more scalable than a standalone server for sporadic tasks. The benefits of using Fargate over a typical server is that Fargate can be run on-demand which eliminates the need to provision servers and it can scale to run a practically unlimited number of instances in parallel.

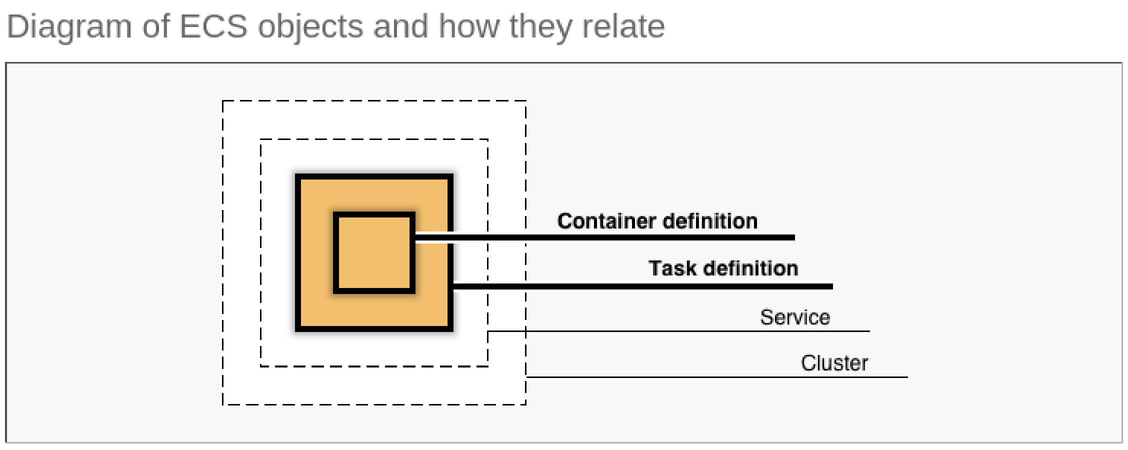


Figure 6.3: Diagram taken from the Fargate setup wizard that illustrates how Fargate objects relate.

I start by uploading the solver container built in the previous section to a Docker Hub[1] repository. This allows Fargate to discover and install the container. Next, I create a new Fargate task and give it access to the docker hub repository. I also specify the environment variables required by the scheduler container such as the input and output directory; configure the required memory and CPU cores, and set the container to automatically log information to CloudWatch. CloudWatch is AWS's monitoring and logging service which is capable of recording and alarming a large variety of metrics.

The only step required in setting up the task definition is to give the task an IAM role so the task has permission to read and write the schedules and solutions. IAM is AWS's credential management service; it can provide services with additional permissions by assigning a service a role. In this case, I create a role that allows schedules to read from an S3 input bucket and write to an output bucket.

There is no need to provision a service to manage the task because the task execution will be entirely event-driven.

Last, I configure the Fargate cluster. The cluster is used to manage the running of the tasks. I apply an auto-scaling rule to the cluster that simply limits the number of running tasks to prevent the service from being abused. If the web application proved to be popular, the auto-scaling service can be more intricately defined.

6.2.2 Automating the Scheduling Service

I automate the scheduling service by leveraging S3 event triggers. S3 is AWS's cloud storage offering. The benefit of using an S3 bucket is it can trigger an event when a new file is uploaded.

I create two S3 buckets for schedule files and rosters. I configure the input bucket to only accept JSONs and to trigger the 'solve-schedule' lambda when a new file is uploaded. Similarly, I configure the output bucket to trigger the 'notify-user' lambda.

AWS Lambda is a serverless compute service that can run code functions. When the 'solve-

[1] www.hub.docker.com

schedule' lambda is triggered, it starts a new Fargate schedule task and passes in environment variables that inform the scheduler where to read the input schedule from and where to write the output to. When the 'notify-user' lambda is triggered it uses the AWS Simple Email Service to email the user to notify them that a roster has been produced.

6.2.3 User Management

To prevent the service from being abused and to easily keep track of data, users must create an account. Rather than implement a user management system from scratch, I use AWS Cognito. Cognito provides a range of user management tools. This project uses it to allow users to easily sign-up and sign in to the web application.

To set up Cognito, I create a default user pool for the application. A user pool allows one to control all aspects of the user sign-up process such as password quality, multi-factor authentication, specify what details are needed to sign-up, etc. Once the user pool is set up, I record the pool credentials which will be needed later to connect the front end to the user pool.

6.2.4 Front End Infrastructure

I opt to use AWS Amplify to host the web application. Amplify is an inexpensive and scalable service for hosting web applications. To set up Amplify, I simply provide the service with access to the front end repository and configure it to auto-build the website every time I commit to the repository.

By default, Amplify comes with its own domain, though I would like to use a custom domain[2] for the service. Before I add my domain, I move the domain's DNS server to AWS Route 53. The benefit of doing this is that AWS services closely integrate with Route 53. Once Route 53 is set up, I add my domain to Amplify by simply selecting it from a drop-down menu.

6.2.5 Advanced Changes

At this stage, the back end is completely functional. However, the only method to communicate with the back end is through uploading files. To allow more advanced interactions, I create a REST API using AWS's API Gateway service which is capable of passing requests to individual services like Lambda and DynamoDB. I configure the gateway to only accept requests from authenticated users. There are no active APIs for the service, but it is in a position where the API can be expanded upon in the future to allow users to modify their accounts and to connect a GUI schedule builder to the back end.

Finally, I use DynamodDB to record user data. I create a simple NoSQL table that maps a user email to a set of preferences. At this time, the table just records if the user has granted permission for their schedules to be used to improve the service.

[2] `www.quickroster.me`. The site will not be made public until the embedded credentials are secured. To try the site before then, follow the readme in the front end repository to easily launch the front end on your local machine.

```
<template>
    <div class="container">
        <h1>Schedule Upload</h1>
        <UploadArea/>
    </div>
</template>

<script>
import UserInfoStore from '@/app/user-info-store';
import UploadArea from '@/components/UploadArea';

export default {
    components: {
        UploadArea
    },
    data: function() {
        return{
            userInfo: UserInfoStore.state.cognitoInfo
        }
    }
}
</script>
<style> </style>
```

Figure 6.4: A sample Vue file that describes an upload page.

6.3 Front end

To make the solver accessible to the widest possible audience, this project is offering the service as a web application. To build a powerful, sophisticated front end would take more time than this project can afford. Instead, the focus is to provide a simple, robust, prototype front end that can be easily extended in the future to include advanced features.

To construct the front end, I consider using the Vue, React, and Angular JavaScript frameworks. These frameworks use a model–view–viewmodel (MVVM) software architecture that separates the user interface (the view) from the back end logic (the model); the view model allows data objects from the model to be exposed to the view. These frameworks are easy to connect to server-side infrastructure; there is a wealth of resources and documentation available for implementing typical web app features using these frameworks, and they have access to a wide range of libraries.

I opt to implement the front end in Vue because it is lightweight and simpler than the other competing frameworks. The structure of a Vue file is similar to a standard web page. A sample 'view' of a web page for uploading files is presented in figure 6.4. The template block contains HTML which describes the structure of the web page. The upload area tag is an example of a Vue component. A component is a standalone Vue object that can be inserted into a web page. The script block contains JavaScript and Vue code. This script imports a data structure that tracks logged-in users and the upload area component so that it can be injected into the template. The style block can be used to apply CSS to the page.

To get started, I use the Vue CLI to create a blank project. I import the Bootstrap[3] framework which provides a high quality default element style and I import Sass[4] so I can create powerful

[3] www.getbootstrap.com
[4] https://sass-lang.com/

CSS rules. I design the site in adherence to the Material Design guidelines[5]. I create the following pages: a landing page; a 'manage' page where authenticated users can access the scheduling service; an upload page where users can upload schedule files, and a 'files' page where users can see previously uploaded schedules and download completed rosters.

I install the vue-router[6] library which allows Vue applications to be developed as a single page application. I implement a custom vue-router to control the flow of the application. The benefit of using the router is I can prevent unauthenticated users from accessing certain areas of the service. To determine if a user is authenticated, the router checks if the user session has a valid JSON Web Token. If the user lacks a valid token, they are redirected to Congito's sign-up and sign-in page. If the user successfully logs in, a JavaScript user-store object caches their token.

To use the scheduling features of the web app, an authenticated user must upload a schedule JSON file in the upload area. To upload their schedule, the user's Vue session is temporarily granted credentials which allows it to upload the schedule to the S3 input bucket, in a sub-directory that corresponds to the user's email. Similarly, when a user visits the files page to view their schedules and rosters, the Vue session is temporary granted credentials to retrieve their files from S3.

To make the web application public, the front end code is uploaded to a private repository and the Amplify service is given access to build the site.

6.4 Data Preparation

In the Data Considerations chapter, I identified two sources of data: artificially generated schedules and the scheduling benchmarks multi-activity data set. This project uses these data sources to produce solved rosters which are then split into training, validation, and test data sets. The data must be preprocessed and labeled before it can be used. Preparing the data involves creating a generator to produce a range of artificial schedules, transforming benchmark instances into a more usable format, and creating a method for the solver to load these problems.

6.4.1 Preparing the Data

To generate artificial data, I create a data generator object in Python that implements the generation scheme described in the previous chapter. The generator takes the previously described problem parameters, uses the SciPy[7] library to produce random variables, and outputs a schedule object. The schedule object has an encoding function that exports the schedule as a 'schedule' JSON file. The structure of a schedule file is presented in figure 6.5.

To utilise the benchmark problems, I create a parser to load the schedule txt file like one presented in figure 6.6, convert it to a Python object, and makes the following transformations: each task in the instance is interpreted as a track; each worker's maximum total minutes is interpreted as a worker's maximum shift limit; the mid-interval value of the minimum and maximum coverage is interpreted as the number of staff required for a shift. The instance doesn't specify worker availability so they are assigned a random block of time-off such that a worker has at least eleven hours off every twenty-four hours. After parsing the instance, it is exported as a schedule JSON file for later use.

[5] www.material.io/design
[6] https://router.vuejs.org
[7] www.github.com/scipy/scipy/

```
{
    "uuid": ... ,
    "shift_data": {
        "day_names": [...],
        "shift_names": [...],
        "shift_staffing_level_requirements": [day][track][shift]
           [skill],
        "track_names": [...]
    },
    "skill_data": {
        "skill_hierarchy": {skill_id -> [ [alt_skill, penalty],
           ... ]},
        "skill_id_to_name_map": {skill_id: "skill_name"}
    },
    "worker_data": {
        "worker_availability": [worker][day][track][shift][skill
           ],
        "worker_id_to_skill_map": {id -> [skills]},
        "worker_max_shifts": [...],
        "worker_names": [...]
    }
}
```

Figure 6.5: A sample schedule JSON. Schedule files contain all the information needed by the solver to produce a roster.

```
SECTION_HORIZON
7

SECTION_TASKS
3

SECTION_STAFF
# ID, MinTotalMinutes, MaxTotalMinutes
1,2040,2400
2,960,2400
3,2220,2400
...

SECTION_COVER
# Day, Time, TaskID, Min, Max
1,06:00-06:15,1,1,1
1,06:15-06:30,1,1,1
1,06:30-06:45,1,1,1
...
```

Figure 6.6: A sample multi-activity problem file.

In order for the MIP solver to produce a roster, the schedule files need to be parsed into a schedule object. This requires a decoder. I created a schedule decoder in Python that can deserialize the JSON files, convert them to Python objects, substitute in constants like penalty values, and yield a schedule object which can be used by the solver.

Before I label the data, I generate a set of 235 schedule with the following parameters:

- Number of workers $\sim U(7, 40)$
- Number of days $\sim 1 + 3 \cdot \text{Lognormal}(1, 1)$
- Number of shifts $\sim \mathcal{N}(7,\ 1.5)$
- Number of tracks $\sim 1 + \text{Lognormal}(1, 1)$
- Number of skills $\sim U(1, 4)$
- Mean availability $\sim 1 - exp(0.1)$
- Mean staffing requirement $\sim \mathcal{N}(0.5,\ 0.15)$

The distribution of these parameters is chosen by reviewing plots of the probability density function and selecting the parameters which provide a varied and realistic spread of values. Additionally, fifteen of the easy benchmark instances are chosen at random to be included in the collection. The easy benchmark instances can be computed within a few hours as opposed to a few days. I'm only including a small number of these instances; as previously mentioned, they are extremely sparse in terms of shift requirements, and some schedules span weeks in duration which is not a realistic duration for a volunteer event. This results in a collection of 250 schedules which is then randomly divided using a 60:20:20 split into a training set of 150 schedules, a validation set of 50 schedules, and a test set of 50 schedules.

6.4.2 Labeling the Data

I have created three data sets that need to be labeled. First, each of the schedules is solved by the scheduler which produces an optimal roster for each of the schedules. A roster contains all the information of a schedule in addition to a matrix indicating if a volunteer w is assigned to shift (d, t, s, k). The Gurobi solver is configured to return all optimal solutions it encounters so there are multiple results for each of the schedules. It is important to keep in mind that this is not an exhaustive set of optimal solutions.

Next, each of the rosters is processed by the roster labeler object. The labeler relates each volunteer w and shift (d, t, s, k) to a target variable that denotes if that volunteer is assigned to that shift. At the same time, the labeler computes the values of each of the features described in the next section. The features and target values for each roster are exported to a CSV file.

Finally, the labeled rosters are concatenated together to produce two labeled CSV files for the training and evaluation data sets.

6.5 Machine Learning

The exact solver can produce excellent rosters. However, for large instances, the running time can be significant. For a schedule with more than thirty volunteers, it can take from a few minutes to several hours to compute the solution. Although this project is cloud-based and has access to practically unlimited amounts of computing power, AWS bills computing tasks by the second. It is in the interest of the project to optimize the solver's performance so it can solver larger problems, compute results more quickly, and minimize the cost of the service.

As discussed in the related works, combinatorial problems like the NSP can be optimized with a machine learning pruning technique whereby a model is trained to identify decision variables that are unlikely to be included in the optimal solution. The decision variables can then be pruned which allows the MIP solver to compute the solution faster.

The goal is to create a model that can significantly prune instances while maintaining the integrity of the solution. The following metrics are used to evaluate a model's ability to achieve this goal:

1. **False negative rate** is the percentage of assignments that belong in the optimal solution but are misclassified to be not assigned. A high false negative rate indicates that volunteers in an optimal solution are being incorrectly pruned. There isn't a one-to-one relationship between the false negative rate and the objective function score because the labeled data only contains a subset of the optimal solutions. This means some misclassifications may not degrade the solution quality.

2. **Prune percentage** is the percentage of decision variables that can be pruned from the problem. This indicates how much quicker the solver will become. One would expect that significantly pruned problems can be solved much quicker compared to the original problem.

3. **Change in objective function score** is the percentage difference between the objective score of the exact solution and the predicted solution. This metric quantifies the change in solution quality.

To support the rapid prototyping and modifications to the machine learning process, I created a simple file-based machine learning pipeline which is presented in figure 6.7. The pipeline is controlled by a Python script that uses flags to indicate which stages to perform and skip. The pipeline constants like file directories are defined in a constants object so they can be easily modified. The pipeline is supported by a series of helper scripts that bootstrap different operations.

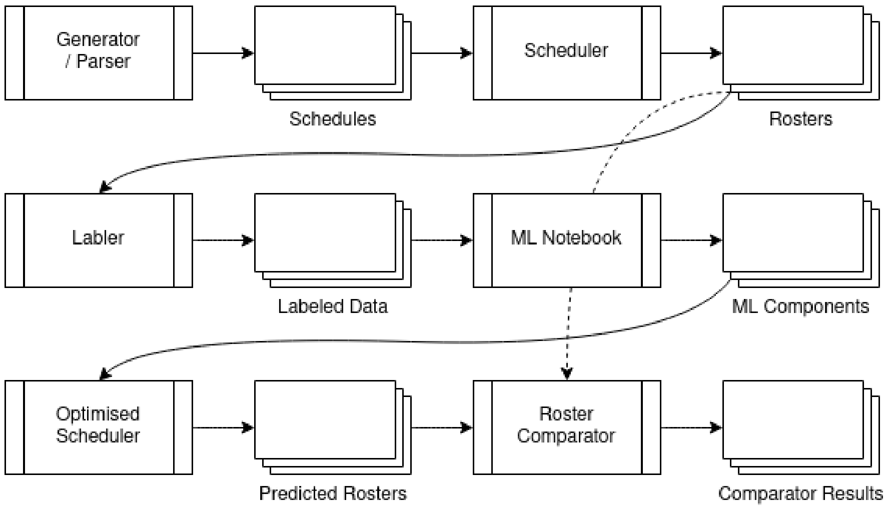


Figure 6.7: File-based machine learning pipeline pipeline for creating rosters, labelling data, experimenting with machine learning, and performing experiments. The flow of the pipeline is controlled by a Python script.

6.5.1 Feature Engineering

There isn't a standard feature set for the NSP so I devised a wide range of features that might be useful for determining the likelihood of a volunteer being assigned or not. I identified 33 potential features that fall under one of these five categories:

- Problem dimension e.g number of workers, number of days.
- Volunteer availability e.g worker is available, worker max availability.
- Schedule requirements e.g staffing requirements, availability sparsity.
- Penalty e.g worker skill penalty, worker penalty vs average penalty for available workers of that shift, does the worker have the minimum penalty.
- Meta-features such as the linear programming relaxed (LPR) value of the decision variable.

The time to compute each feature value for a schedule is negligible. The most expensive feature is the LPR value which can usually be calculated in sub-second time. The LPR is computed by solving the schedule as a linear programming problem and recording the value of the decision variables. The difference between a linear programming problem and a MIP is that variables in a linear program are all continuous while MIP variables can be a mix of integer and continuous values. In the context of the NSP, this means that workers can be partially assigned to shifts which is nonsensical but as we see later, this proves to be a valuable feature. The only change needed to solve the scheduled as linear programming problem is to relax the integer variables of the MIP formulation to continuous variables using Gurobi's relax function. Otherwise, the problem can be solved normally. I want to highlight that the relax function does not relax binary variables as one might expect; this transformation must be implemented by the programmer.

After creating an initial feature set, feature selection is performed to reduce the dimensionality of the feature set and to remove poor quality features. At each stage in the feature selection process, a stochastic gradient descent classifier and a random forest classifier are trained using the updated training data and the classifiers are evaluated using the validation data to verify that their false negative rate and prune percentage scores are not degraded.

First, the highly correlated features are removed. Each feature's Pearson correlation coefficient is computed using the Pandas[8] `corr()` function and displayed on a Seaborn[9] heat map. If a pair of features have a correlation $\geq |0.9|$, one of them is removed depending on which feature seems least useful in terms of information gain and random forest importance.

Next, the mutual information between each feature and target variable is computed using Sklean's[10] mutual information classifier. Features with an information gain of less than one percent are removed without affecting the evaluation metrics. Similarly, the Gini importance of each feature is computed using Sklearn's random forest classifier. Features with importance less than one percent can be removed without consequence.

At this point, the feature set has been reduced from 33 features to 16. Finally, to reduce the feature set even further, I use sequential forward selection to greedily identify which subset of features results in a monotonic decrease in the false negative rate while maintaining a reasonable prune percentage. I choose to use forward search over an exhaustive technique like sequential backward selection because it is much quicker to perform: the sequential backward selection has

[8]Open source data analysis and manipulation library.
[9]Python data visualization library based on matplotlib.
[10]Python machine learning library

to train many more iterations of the model which is particularly time-consuming as the random forest classifier takes many minutes to fit this data set.

This process reduces the feature set to just eleven features. A list of these eleven features along with their Gini importance and their information gain is presented in table 6.1. A list of all 33 features and a brief description of each feature is included in the appendix. A heat map showing the correlation of the remaining eleven features is also included in the appendix.

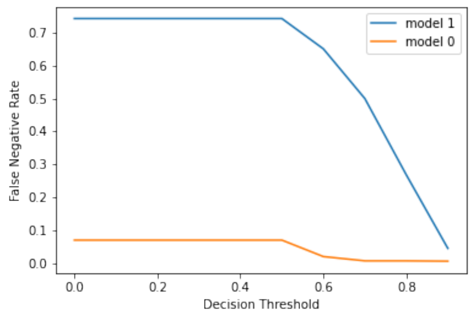
Feature Name	Gini Importance	Information Gain
Linear programming value of the decision variable	0.507	0.216
Worker skill penalty equals the minimum penalty	0.128	0.208
Worker skill penalty	0.114	0.169
Difference between current worker's skill penalty and the average available worker's skill penalty	0.081	0.090
Worker maximum availability	0.051	0.008
$\mathcal{X}^2$ of the staffing requirements for the current day	0.030	0.024
$\mathcal{X}^2$ of the staffing requirements for the current shift	0.030	0.020
Sparsity of staff availability	0.025	0.028
Worker availability Z-score	0.015	0.026
Difference between current available and average availability	0.013	0.009
Worker's available shifts is less than the worker's maximum shift limit	0.006	0.123

Table 6.1: A list of the eleven best features, their Gini importance, and their information gain.

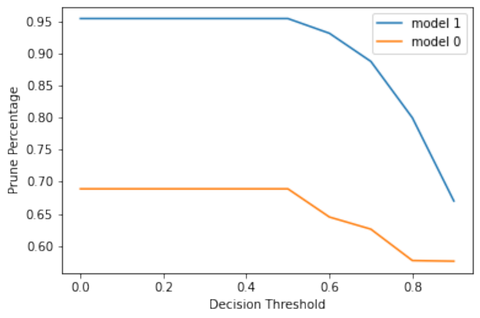
6.5.2 Training

To take advantage of Sklearn's wide range of models and tools, a Python Jupyter notebook is used to facilitate the model training. The training and validation data sets are loaded using a Pandas data frame. I opt to standardize the data using Sklearn's standard scaler which converts feature values to Z-scores by subtracting the mean and scaling the data to unit variance.

The training data is extremely unbalanced. It contains 1,466,873 values, 137,552 of which are assignments and 1,329,321 are non-assignments (91:9 split). I remove all data points where the volunteer is unavailable because the scheduler already prunes unavailable volunteers so there is no benefit in a model learning this behaviour. This reduces the non-assignments to 1,189,129 (9:1 split). To account for this natural class imbalance, each model is trained with the class weight parameter set to 'balanced'. It is evident in figure 6.8 that using the 'balanced' class rate can greatly reduce the false negative rate at an acceptable cost to the prune percentage.



(a) Effect on the false negative rate when using a balanced class weight. Model 0 uses a balanced class weight, model 1 uses the default class weight.



(b) Effect on the prune percentage when using a balanced class weight. Model 0 uses a balanced class weight, model 1 uses the default class weight.

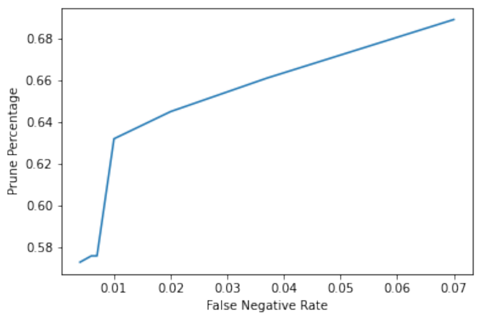
Figure 6.8: Demonstration of the effect of using the 'balanced' class weight parameter on a stochastic gradient classifier with a 'log' loss.

I explore the suitability of using the following classifiers for this task: stochastic gradient descent (SGD) with log loss, Gaussian naive Bayes, Bernoulli naive Bayes, decision tree, random forest (RF), and k-nearest neighbour. To make a comparison, I train each model with its default parameters and the 'balanced' class weight. I noted in preliminary experiments that models with a medium to high false negative significantly degraded the objective function score. Because of this observation, I'm most interested in models that minimize the false negative rate while maintaining a moderate to high prune percentage.

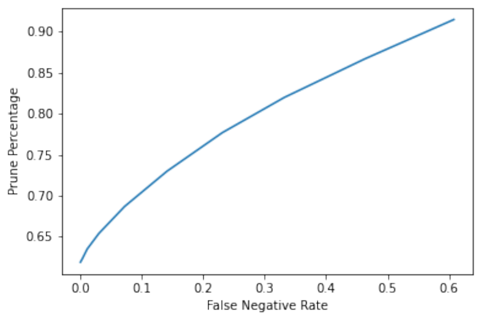
To evaluate each model's performance, I plot the decision threshold vs prune rate, decision threshold vs false negative rate, false negative rate vs prune percentage and use these plots for comparison. I found the Gaussian naive Bayes to be the worse performing model because surprisingly, varying the decision threshold did not change its prune rate nor its false negative rate. The most suitable models are the SGD and the RF classifiers. They both offer low false negative rates while maintaining a relatively high prune rate. Their performance is compared in figure 6.9. Clearly, there is a trade-off between the false negative rate and the pruning rate. The RF is capable of the greatest pruning rate but ultimately, I decide to use the SGD model because it boasts an exceptionally low false negative rate while maintaining a moderate prune percentage.

Finally, I hyper-parameter tune the model to achieve optimum performance. Typically the hyper-parameters are found using a grid search algorithm that tries to find the parameters that maximises a score function. I haven't defined a suitable scoring function that balances the false negative rate and the pruning rate so I opt to manually tune the model. I achieve this by training a series of models with different values for the same parameter and plot the performance of each model against the model with the best performing parameters. I found the optimal model parameters for the SGD are a modified Huber loss function with an elastic net penalty, a maximum iteration limit of 1000, and a balanced class weight. The most influential parameter is the class weight. The optimal model and the data scaler are exported using the Joblib[11] library's dump function. Writing the model and the scaler to a Joblib file means they can be efficiently loaded into Python at a future time without needing to be refitted.

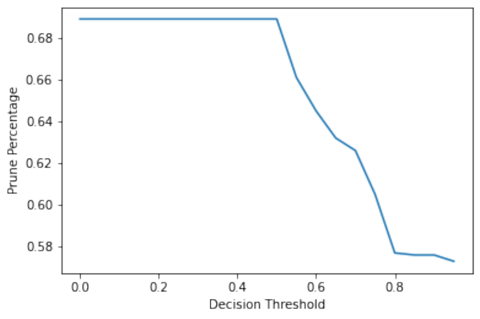
[11] github.com/joblib/joblib



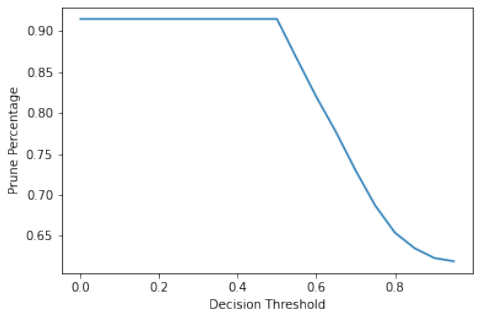
(a) SGD false negative rate vs prune percentage.



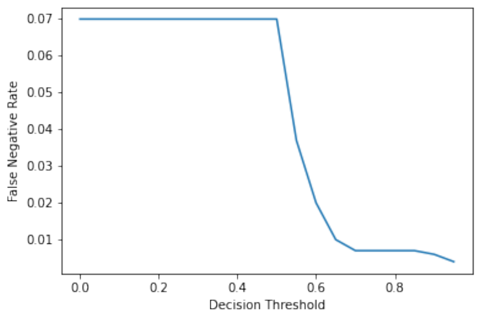
(b) RF false negative rate vs prune percentage.



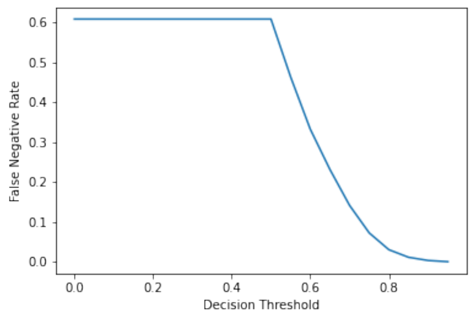
(c) SGD false negative rate vs prune percentage.



(d) RF false negative rate vs prune percentage.

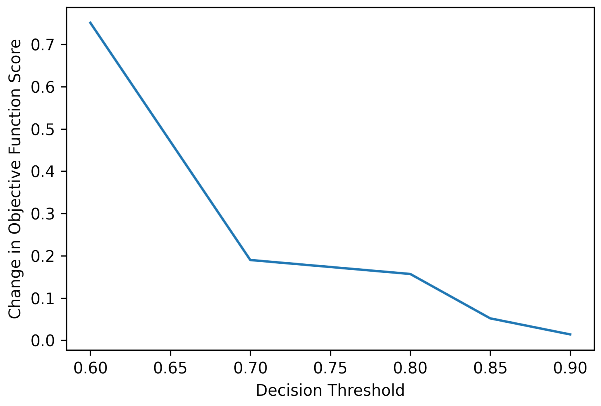


(e) SGD false negative rate vs prune percentage.

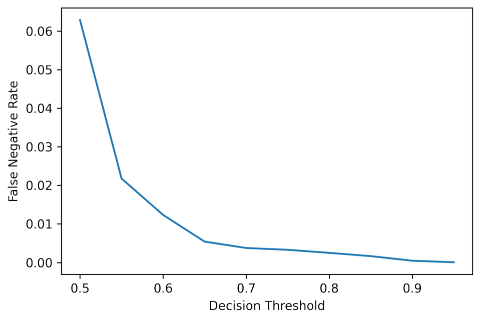


(f) RF false negative rate vs prune percentage.

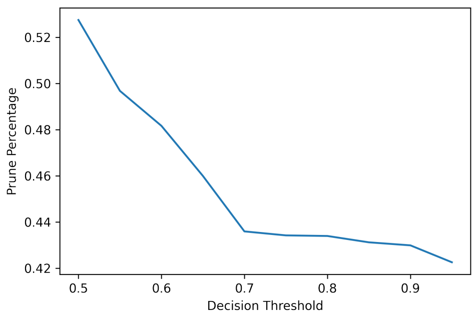
Figure 6.9: Comparison of a SGD model with a log loss vs a RF model. Both models are trained using their default parameters and a 'balanced' class weight. The plot data is computed by varying each model's decision threshold from 0 to 1 and recording the resulting prune percentage and false negative rate.



(a) The effect of varying the decision threshold on the average objective function score.



(b) The effect of varying the decision threshold on the average false negative rate.



(c) The effect of varying the decision threshold on the average prune percentage.

Figure 6.10: The performance of the tuned model when applied to individual schedules in the validation set. The decision threshold is varied between 0.6 and 0.9 for figure (a) and is varied from 0.5 to 0.95 in steps of 0.05 for (b) and (c).

6.5.3 Validation

Now that the model has been trained and tuned, I want to validate that it works well on individual rosters because up to now the model has been tested on a concatenated version of the validation data i.e the validation data has been treated like one massive schedule. Additionally, I can use the validation results to determine the optimal decision threshold.

To perform the validation, I create an optimized scheduler that uses the previously created model to prune input schedules. Each roster in the validation set is then solved with the optimized scheduler multiple times with various decision thresholds. The resultant changes in objective function score, false negative rate, and the prune percentage after running the solver are recorded. Last, I group the scores by decision threshold and calculated the average score for each threshold.

The results of the validation are presented in figure 6.10. The overall performance is reasonable: using a decision threshold greater than 0.85 yields an acceptable level of deterioration of the objective function score and prune rate above 40%. It's interesting to note that the average prune rate of the individual schedules is much lower than the 60% prune rate of the amalgamated schedules. The optimal decision threshold is 0.85; thresholds lower than this yield an unacceptable increase in the objective function score.

Chapter 7: Evaluation

The objectives of this project are to create a MIP solver that is suitable for volunteer scheduling, develop a web application so that others can use the service, and optimize the service with supervised search space pruning.

7.1 MIP Formulation of the NSP Algorithm Evaluation

The MIP formulation of the NSP algorithm has gone through many iterations and each iteration is evaluated before proceeding to the next stage as changing the formulation late into the project would be detrimental. The formulation is evaluated on its ability to create rosters that have good coverage, that assign volunteers with the most suitable skills to shifts, that have fair workloads, and that can be used for a wide range of volunteer scheduling scenarios.

The formulation is evaluated in multiple steps. First, the solver must solve the Netsoc schedule presented in figure 2.1. The solver can produce a perfect roster for this schedule: there is complete coverage, no skill penalty, and each volunteer has a balanced number of shifts.

Next, I generate and solve a selection of small rosters with only a few workers and shifts. I check each of the rosters to check that the roster is practical, usable, and of good quality. I also attempt to manually improve the schedule by swapping shifts around and re-arranging them. I'm confident the solver produces high-quality small schedules that can't be manually improved.

Finally, I generate and solve a selection of large rosters. Because these rosters are quite large, I only want to inspect them to ensure that the workload is balanced and that no shift is excessively relaxed or assigned too many non-optimal volunteers. I noticed that an early iteration of the formulation greedily assigned workers to shifts when the schedule was sparse and this resulted in unbalanced workloads. This is corrected in the final formulation by allowing the skill penalty to deviate slightly when performing the workload balancing optimization.

Overall, I'm confident that this MIP formulation provides high-quality rosters for this variant of the NSP.

7.2 Supervised Search Space Pruning Model Evaluation

The objective of the supervised search space pruning model is to reduce the average running time of the solver without noticeably degrading the quality of the solution. Two experiments are conducted to evaluate the performance of the model. The first experiment is concerned with evaluating the

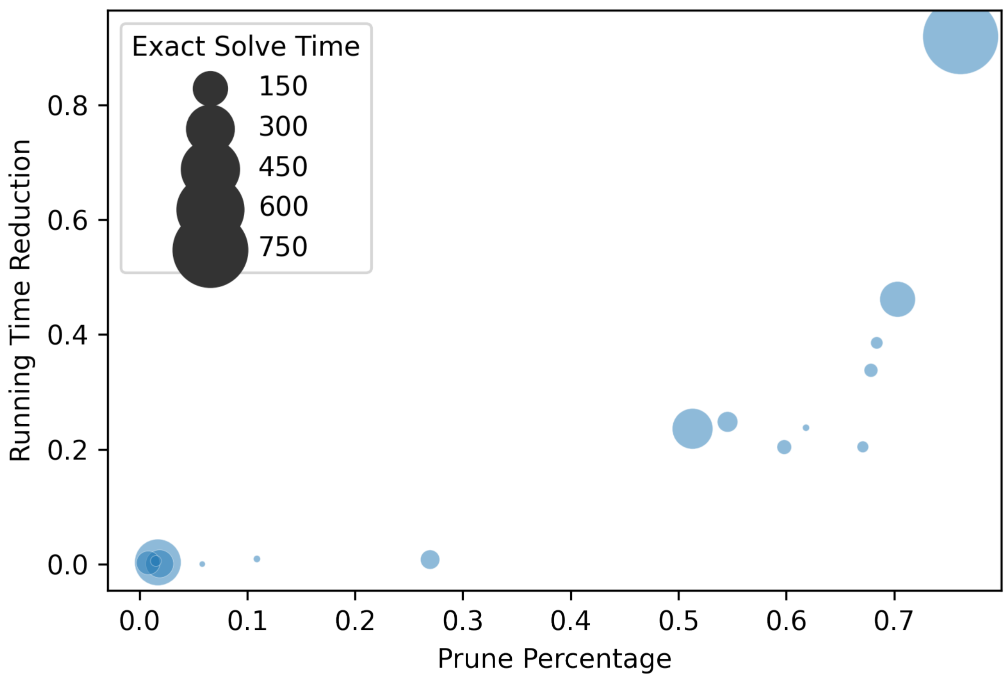


Figure 7.1: The performance of the optimized solver with an 85% decision threshold is displayed in terms of prune percentage and percentage time reduction. The size of each bubble indicates the time taken in minutes for the standard solver to compute the solution. Only instances from the test set that take more than five minute to solve are included in the plot.

model's overall performance; the second experiment is much smaller and is concerned with the model's performance on challenging instances.

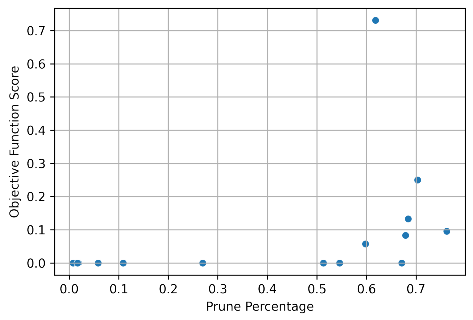
Experiment 1

This experiment is conducted on the previously prepared test set of 50 typical schedules. It takes 32.4 hours to produce all 50 rosters using the standard solved. When using the optimized solver, which pre-prunes the schedule using the search space pruning model, the solve time is cut by 47%, to 17.1 hours. The schedules are pruned by an average of 43%. The longest-running task saw its solve time reduced from 13 hours to just one hour, a reduction of 92%.

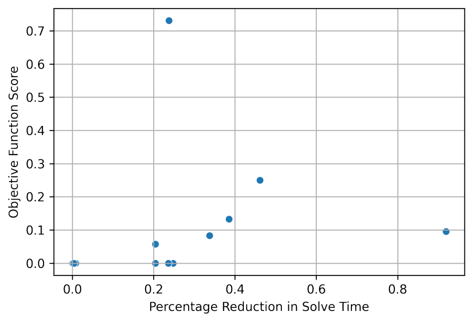
Of the 50 instances, the standard solver can produce rosters for 34 of the instances within five minutes. The remaining 16 instances take an average of two hours and one minute to solve. Using the optimized solver, the average solve time is reduced to one hour and four minutes. The performance of the optimized solver on these medium- to long-running problems is displayed in figure 7.1.

From figure 7.1, one can see there is an exponential relationship between the level of pruning and the running time reduction: as the prune percentage increases, the running time reduction increasing exponentially. Also, the level of pruning appears to have an almost bi-modal distribution whereby nine of the instances are significantly pruned while the other seven saw little to no pruning when using an 85% decision threshold.

Even though the pruning was quite high for many of the instances, the time reduction is moderate, especially for small and simple instances. One reason for this could be that the performance of these small and simple instances varies greatly depending on the solving technique is used by Gurobi.



(a) The relationship between the prune percentage and the change in objective function score.



(b) The relationship between the prune percentage and the change in objective function score.

Figure 7.2: The change in solution quality when using the optimized solver. Objective function score is the percentage difference between the score of the pruned solution and the optimal solution. Only instances from the test set that take more than five minute to solve are included in the plot.

Mathematical optimizers like Gurobi use a variety of techniques and heuristics like branch and bound, simplex algorithm, and greedy heuristics to solve integer programming problems [32]. It's possible that the standard schedule and the pruned schedule are solved with different techniques or that the techniques used don't particularly benefit for moderate pruning.

This running time is achieved without noticeably affecting the solution quality; the average objective value of a pruned solution is increased by only 4%. Of the 50 instances, 38 are solved by the optimized solver with no change to the objective function value.

Figure 7.2 illustrates the change in solution quality when using the optimized solver on medium-to long-running instances. Most instances see little change in their objective function score. There is one instance that has an optimal object score of one but the score of the predicted solution is 5 which explains the unusually high 70% change in objective function score.

It is evident from figure 7.2.a that there the objective function score degrades as a greater percentage of the problem is pruned away. By attempting to prune almost all the prune-able variables, there is an increased probability of removing a variable that belongs in the optimal solution. This is a consequence of using a fixed decision threshold. By implementing a dynamic threshold, the pruning could be more controlled.

There isn't a well-defined relationship between the solve time and the change in objective function score; it appears that instances with a greater reduction in solve time have a greater change in objective function score but more data is needed to support this observation.

Experiment 2

In experiment one, I observed that the instances that took the longest to solve often saw the greatest reduction in running time, especially dense instances. A dense instance is a scheduling problem with a high average shift staffing requirement; generally, they tend to be more challenging to solve than sparse instances which have a low average shift staffing requirement.

In this experiment, I want to analyse the optimized solver's performance on challenging long-running instances. To explore this more, I conduct a small experiment. I generate fifteen instances using the same scheme described in the previous chapter except I change the distribution of the mean shift staffing requirement from $\mathcal{N}(0.5, 0.15)$ to $\mathcal{N}(0.8, 0.05)$. Empirically, I have noticed that instances with a high mean shift requirement take much longer to solve so this should produced challenging instances.

Using the standard solver, six of the fifteen instances proved challenging to solve. It takes the standard solver 13.6 hours to solve these six instances. The average running time of these six instances is 2 hours and 16 minutes. Next, I resolve the six instances with the optimized solver. It takes only 3.4 hours to roster the schedules which represents a 75% reduction in running time. This has the effect of reducing the average solve time from 2 hours and sixteen minutes to just 34 minutes. This means the optimized solver can achieve four times the throughput of challenging instances compared to the standard schedule.

The average running time reduction is 58% with the longest-running tasks receiving the greatest running time reduction. The longest-running instance which ran for 7.5 hours could be solved in one hour which represents an 86% reduction in running time. The average prune rate for these instances is 61% and the change in objective function score was 9%. An important caveat to this experiment is that the sample size is quite small and further testing should be performed with a larger sample size.

Summary

Overall, the search space pruning model is a success. Experiment 1 shows that in typical cases, the optimized solver reduces the running time of schedules by almost fifty percent which allows the system to achieve double the throughput. Experiment 2 shows the optimized solver performs best on long-running instances. A possible explanation for this could be that Gurobi's powerful variety of techniques and heuristics are efficient at solving sparse instances which would suggest why these instances see a lower running-time reduction compared to the challenging instances which can see running time reductions of up to 92%.

Chapter 8: Conclusions and Future Work

8.1 Conclusion

Throughout the course of this project I have,

- Identified a variant of the nurse scheduling problem suitable for scheduling volunteers;
- Formulated the algorithm as a mixed-integer linear programming problem and implemented it using the Gurobi mathematical optimizer;
- Created a scalable serverless web application with a simple Vue front end and a robust AWS back end;
- Generated a realistic volunteer scheduling data set in the absence of a genuine data set;
- Optimized the solver using a search space pruning model to cut the average solve time in half at almost no cost to the objective function score.

Ultimately, this project has successfully achieved its goals of creating a scheduling algorithm that caters to the needs of volunteer organisations and creating a scalable, serverless web application that utilises this algorithm to roster volunteers.

8.2 Future Work

Though this project has fulfilled its core and advanced requirements, there are still improvements that can be made to the MIP formulation, the web application, and the supervised search space pruning model. Additionally, the demonstrated success of the supervised search space pruning model opens up more avenues of exploration.

8.2.1 MIP Formulation

This project uses a robust algorithm for scheduling volunteers but there are three improvements I would like to make.

First, I would like volunteers to have the ability to express greater availability preferences such as "not available", "available if needs be", and "available". This is a popular feature in Doodle which would be beneficial for users of this application.

I would also like to offer the ability to set a minimum staffing level. In some cases, it may be unacceptable to slack a shift below a certain threshold. This could be prevented by adding a new hard constraint that prevents a shift from being slacked below its minimum staffing level.

Last, I would like to implement a dynamic shift slacking penalty. In particularly challenging schedules, the solver may need to significantly slack shifts across a track but instead of slacking each shift a small bit, it is possible for the solver to greedily slack an entire shift. If a dynamic shift penalty is implemented such that it is cheap to lightly slack a shift and expensive to excessively slack it, this would balance how shifts are slacked.

8.2.2 Web App Improvements

At present, the AWS components are created, managed, and provisioned through the web console. Although using the console is acceptable for this project considering its time frame, if the application wishes to see real-world use, it should be configured programmatically. It would be straightforward to model the web application using the Serverless[1] framework now that the application has been constructed end-to-end and I have determined how the services need to be configured to interconnect. The Serverless framework, which is different from the design paradigm, allows AWS infrastructure to be described in a configuration file and launched using the Serverless framework. The benefits of using this framework are that changes to the infrastructure can be tracked through git and if the configuration files are made open-source, others can use them to launch similar services.

Currently, the schedules are created by the organiser only. This can be improved by allowing the organiser to share a schedule code with the volunteers so that they can input their availability and skills. This would allow volunteers to self-schedule themselves. If this is implemented, the organiser should be offered additional controls that would allow them to tweak volunteers' skills to improve the schedule feasibility.

The web application is controlled by a simple Vue front end. I would like to see this front end improved to offer a greater user experience and to match the standard of its competitors. One example would be to improve the schedule creation view to the standard of Doodle, which could be achieved with packages like vuedraggable[2].

I feel the user experience could be improved further with the creation of a mobile app. Since this project uses Vue, the web application can be easily exported to Android and IOS using Vue Native. Vue Native is a framework for creating cross-platform native mobile apps using JavaScript and Vue. Most of the front end codebase can be reused. The most significant change would be to modify the views so that they're mobile and touchscreen-friendly. For frequent users of Quick Roster, this could greatly improve their user experience.

Another useful feature would be calendar integration. I would like to add the ability for users to export their individual roster to Google Calendar. This would allow them to view their shifts alongside their other calendar events. A minor benefit of this is that users would not have to visit the site to view their roster; this would reduce site traffic which in turn reduces the running cost of the web application because web pages would not need to be served and there would be fewer calls to: Cognito and S3.

Finally, Cognito supports a range of single sign-on (SSO) options like signing in through Google, Facebook, Microsoft, etc. I would like to offer users the ability to sign in with a range of SSOs. This would result in a frictionless sign-up process for the web application.

[1] `www.serverless.com`

[2] `www.github.com/SortableJS/Vue.Draggable`

$$\begin{aligned}
\text{Upper bound of Pruning} &= 80\% \\
\text{SafePrunePercentage(Max Prune)} &= 52\% \\
\text{P(being pruned)} &= [0, .9, .2, .5, .3, .4, .4, .2, .9, .2] \\
\text{Sorted(P(being pruned))} &= [0, .1, .2, .2, .3, .4, .4, .5, .9, .9] \\
\text{52th Percentile} &= 0.4
\end{aligned}$$

Figure 8.1: An example of dynamic pruning. In this example the optimal decision threshold is 0.4. Pruning above the upper bound is guaranteed to incur a slack penalty. The safe prune percentage in this example is a function of the the maximum prune; it estimates than 52% of assignments can be safely pruned. The decision threshold is then determined by sorting the probability of assignment and finding the probability that corresponds to the 52th percentile; this probability is the decision threshold that yields a 52% prune.

8.2.3 Supervised Search Space Pruning

Dynamic Classification Threshold

When evaluating the optimized pruner in experiment 1, I remarked that almost half of the long-running schedules received little to no pruning at an 85% decision threshold. If the decision threshold for these lightly pruned instances could be lowered, it would improve the overall performance of the optimized solver. However, decreasing the decision threshold for all instances would be detrimental to the average solution quality. One possible technique to remedy this problem is to implement a dynamic classification threshold.

One approach to implementing a dynamic threshold is to map different classification thresholds to various classes of instance. For example, one could map sparse instances to a low decision threshold and dense instances to a greater threshold. This would require a lot of experimentation to decide how to map instances to decision thresholds.

A simpler approach could be to decide the optimal pruning rate for an instance and find the decision threshold which yields this level of pruning. The challenge with this approach is determining what level of pruning can be safely performed without affecting the objective value. Figure 8.1 contains an illustrative example of how dynamic pruning could be implemented.

Multi-stage Sparsification

A dynamic classification threshold isn't the only method to eke out more performance from the pruning model. Grassia et al. use multiple pruning stages to achieve greater pruning [2]. In each stage, the researchers train a new classification model for elements that were not pruned by earlier classification models. The benefit of this multi-stage sparsification is that each model can prune elements that the previous model found difficult to classify. Applying this technique could yield an even greater pruning performance.

Generalising Supervised Search Space Pruning to the NSP

The volunteer scheduling problem is different, but in many ways more straightforward than the NSP. The assignment of volunteers is ultimately dictated by their availability and good quality rosters can be produced thanks to volunteers' flexibility. Typical NSPs must incorporate labour laws and contractual rights into their formulation. The complexity is evident in the formulation of the ANDROM which has 23 constraints that are defined by work regulation [14]. These additional constraints add extra complexity which can make schedules difficult to compute in a timely fashion.

I believe there is an opportunity to apply supervised search space pruning to these challenging variants of the NSP to great effect. Specifically, there a MIP formulation and a set of benchmark instances available for a popular variant of the NSP available on Scheduling Benchmarks[3]. I believe with additional feature engineering and model exploration, there is an opportunity to attain a remarkable performance improvement for this problem.

Specialised Models and Analysis Scheduling Difficulty

A popular technique for improving supervised models is to train specialised models for use on certain instance types e.g dense, sparse, etc. It could be possible to improve the performance of the optimized scheduler by equipping it with a model trained for sparse and challenging schedules.

Previously, I made the observation that challenging instances tend to have a high average shift staffing requirement. However, this is just an observation. It would be valuable to analyse the correlation between each of the previously devised features and the problem difficulty. This analysis would enable specialised models to be trained to prune sparse and challenging instances.

Creation of a Genuine Data Set

As previously mentioned in the Data Considerations chapter, this project resorts to creating artificial data sets to train the machine learning models. Users of Quick Roster who successfully create a roster are asked if an anonymous copy of their roster can be retained to improve the quality of the service. If the service proves to be popular, these schedules could be used to optimize the models for genuine scheduling problems. Further to this, if users allow these anonymous schedules to be made public, valid rosters could be published to data repositories like Kaggle[4]. Of course, this is an extremely niche data set but it could be interesting to those experimenting with solving the NSP.

[3] `www.schedulingbenchmarks.org`
[4] `www.kaggle.com`

Acknowledgements

I would like to thank my supervisor Dr. Deepak Ajwani for sharing his invaluable expertise, insight, and guidance with me.

Bibliography

1. Burke, E. K., De Causmaecker, P., Berghe, G. V. & Van Landeghem, H. The State of the Art of Nurse Rostering. *Journal of Scheduling* **7,** 441–499. ISSN: 1094-6136. http://link.springer.com/10.1023/B:JOSH.0000046076.75950.0b (2020) (Nov. 2004).
2. Lauri, J., Dutta, S., Grassia, M. & Ajwani, D. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *arXiv:2001.01230 [cs].* arXiv: 2001.01230. http://arxiv.org/abs/2001.01230 (2020) (Jan. 5, 2020).
3. Office, C. S. *QNHS Volunteering and Wellbeing* https://www.cso.ie/en/releasesandpublications/er/q-vwb/qnhsvolunteeringandwellbeingq32013/. (accessed: 03.12.2020).
4. *Handbook of scheduling: algorithms, models, and performance analysis* (ed Leung, J. Y.-T.) OCLC: ocm55519835 (Chapman & Hall/CRC, Boca Raton, 2004). 1 p. ISBN: 978-1-58488-397-5.
5. De Causmaecker, P. & Vanden Berghe, G. A categorisation of nurse rostering problems. *Journal of Scheduling* **14,** 3–16. ISSN: 1094-6136, 1099-1425. http://link.springer.com/10.1007/s10951-010-0211-z (2020) (Feb. 2011).
6. Miller, H. E., Pierskalla, W. P. & Rath, G. J. Nurse scheduling using mathematical programming. *Operations Research* **24,** 857–870 (1976).
7. Osogami, T. & Imai, H. *Classification of various neighborhood operations for the nurse scheduling problem* in *International Symposium on Algorithms and Computation* (2000), 72–83.
8. Bard, J. F. & Purnomo, H. W. A column generation-based approach to solve the preference scheduling problem for nurses with downgrading. *Socio-Economic Planning Sciences* **39,** 193–213. ISSN: 00380121. https://linkinghub.elsevier.com/retrieve/pii/S0038012104000126 (2020) (Sept. 2005).
9. Burke, E., Cowling, P., De Causmaecker, P. & Berghe, G. V. A memetic approach to the nurse rostering problem. *Applied intelligence* **15,** 199–214 (2001).
10. Özcan, E. in *Computer and Information Sciences - ISCIS 2005* (eds Yolum, p., Güngör, T., Gürgen, F. & Özturan, C.)red. by Hutchison, D. *et al.* Series Title: Lecture Notes in Computer Science, 482–492 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005). ISBN: 978-3-540-29414-6 978-3-540-32085-2. http://link.springer.com/10.1007/11569596_51 (2020).
11. Kawanaka, H., Yamamoto, K., Yoshikawa, T., Shinogi, T. & Tsuruoka, S. *Genetic algorithm with the constraints for nurse scheduling problem* in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)* **2** (2001), 1123–1130.
12. Dowsland, K. & Thompson, J. Solving a nurse scheduling problem with knapsacks, networks and tabu search, 9.
13. Registered Irish Charities - Social and Economic Impact Report 2018, 32 (2018).
14. Burke, E. K., Curtois, T., Qu, R. & Vanden-Berghe, G. Problem Model for Nurse Rostering Benchmark Instances, 29.
15. Chen, Y., Liu, A., Sciannella, E. & Zhang, A. A Comparison of Approaches to the Nurse Scheduling Problem, 8.
16. Lin, X., Hou, Z., Ren, H. & Pan, F. Approximate Mixed-Integer Programming Solution with Machine Learning Technique and Linear Programming Relaxation (Nov. 2019).
17. MIT. *Learning-Augmented Algorithms* https://www.eecs.mit.edu/academics-admissions/academic-information/subject-updates-spring-2019/6890. (accessed: 04.12.2020).

18. Mitzenmacher, M. A Model for Learned Bloom Filters, and Optimizing by Sandwiching. *arXiv:1901.00902 [cs, stat]*. arXiv: 1901.00902. http://arxiv.org/abs/1901.00902 (2020) (Jan. 3, 2019).

19. Vassilvitskii, S. What Can ML Do For Algorithms? *Speech to speech translation,* 78.

20. Balcan, M.-F., Dick, T., Sandholm, T. & Vitercik, E. Learning to Branch. *arXiv:1803.10150 [cs]*. arXiv: 1803.10150. http://arxiv.org/abs/1803.10150 (2020) (May 16, 2018).

21. Kraska, T., Beutel, A., Chi, E. H., Dean, J. & Polyzotis, N. The Case for Learned Index Structures, 16 (2018).

22. Purohit, M., Svitkina, Z. & Kumar, R. Improving Online Algorithms via ML Predictions, 10.

23. Kelleher, J. D., Mac Namee, B. & D'Arcy, A. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies* 595 pp. ISBN: 978-0-262-02944-5 (The MIT Press, Cambridge, Massachusetts, 2015).

24. Vinyals, O., Fortunato, M. & Jaitly, N. Pointer Networks. *arXiv:1506.03134 [cs, stat]*. arXiv: 1506.03134. http://arxiv.org/abs/1506.03134 (2020) (Jan. 2, 2017).

25. *Supervised and Unsupervised Learning for Data Science* (eds Berry, M. W., Mohamed, A. & Yap, B. W.) (Springer International Publishing, Cham, 2020). ISBN: 978-3-030-22474-5 978-3-030-22475-2. http://link.springer.com/10.1007/978-3-030-22475-2 (2020).

26. Probst, M., Rothlauf, F. & Grahl, J. Scalability of using Restricted Boltzmann Machines for Combinatorial Optimization. *arXiv:1411.7542 [cs]*. arXiv: 1411.7542. http://arxiv.org/abs/1411.7542 (2020) (Nov. 27, 2014).

27. Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research* **4,** 237–285 (1996).

28. Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. & Song, L. Learning Combinatorial Optimization Algorithms over Graphs. *arXiv:1704.01665 [cs, stat]*. arXiv: 1704.01665. http://arxiv.org/abs/1704.01665 (2020) (Feb. 21, 2018).

29. Armbrust, M. *et al.* Above the Clouds: A Berkeley View of Cloud Computing, 25.

30. Jonas, E. *et al.* Cloud Programming Simplified: A Berkeley View on Serverless Computing, 35.

31. Vanhoucke, M. & Maenhout, B. Characterisation and Generation of Nurse Scheduling Problem Instances, 31.

32. Önal, H. First-best, second-best, and heuristic solutions in conservation reserve site selection. *Biological Conservation* **115,** 55–62. ISSN: 00063207. https://linkinghub.elsevier.com/retrieve/pii/S0006320703000934 (2021) (Jan. 2004).

33. Group, N. N. *Why You Only Need to Test with 5 Users* Mar. 2000. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

34. Lewis, J. R. IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction* **7,** 57–78. ISSN: 1044-7318, 1532-7590. http://www.tandfonline.com/doi/abs/10.1080/10447319509526110 (2020) (Jan. 1995).

Chapter 9: Appendix

9.1 Link to the GitLab Repository

The code for this project can be found on the internal UCD Gitlab repository: `https://csgitlab.ucd.ie/creavt/final-year-project-submission`

9.2 Full Feature Set

1. *Number of workers*.
2. *Number of days*.
3. *Number of shifts*.
4. *Number of tracks*.
5. *Number of skills*.
6. *Staffing required*
7. *Sparsity of staff requirements* describes what percentage of staff are required for a shift on average.
8. ***Sparsity of staff availability*** describes what percentage of staff are available for a shift on average.
9. *Average number of skills of each worker*.
10. *Median number of skills of each worker*.
11. *Chi-square value of the number of skills of each worker*.
12. *Worker is available* is a boolean that indicates if the worker is available for the current shift.
13. ***Worker skill penalty*** is the worker's skill penalty if they are assigned to the current shift.
14. ***Worker maximum availability*** is the most amount of shifts the worker can be assigned to.
15. ***Chi-square value of the staffing requirements for the current day***
16. ***Chi-square value of the staffing requirements for the current shift***
17. *Difference between average skill penalty and worker penalty*
18. ***Difference between current worker's skill penalty and the average avail worker's skill penalty***
19. ***Worker skill penalty equals minimum penalty*** is a boolean that indicates if the worker can be assigned to the current shift with a minimum penalty.

20. *Worker skill penalty Z-score* indicates the number of standard deviations the current worker's skill penalty is from the average.

21. *Skill rarity* is the percentage of workers with this skill.

22. *Skill demand* is the percentage of assignments that require this skill.

23. *Skill scarcity* is the percentage of shifts that workers with that skill are available to do.

24. *Difference between skill staffing and average skill staffing*

25. *Skill staffing Z-score* is the number of deviations between the number of staff with a specific skill that is required from the average.

26. *Difference between available workers and staffing required across shifts*

27. ***Difference between current available and average availability*** is the difference between the average number of workers available for the current shift and the average number of available workers for all shifts.

28. ***Worker availability Z-score*** is the number of deviations the current number of available workers is from the average availability.

29. *Maximum possible coverage* indicates what percentage of the total amount of shifts could be filled.

30. *Total workers required to fill the roster* indicates the number of workers required to completely staff the roster.

31. *Average shifts required is less than maximum* is a boolean that indicates if the average number of shifts per worker is than their maximum shift limit.

32. ***Worker's available shifts is less than the worker's maximum shift limit*** is a boolean that indicates if a worker is available for fewer shifts than their maximum shift limit.

33. ***Linear programming value of the target*** is found by solving the problem as a linear programming problem as opposed to a MIP problem. Linear programming solutions can be found within seconds. The value of the target variable can be used to indicate what the MIP value of the variable will be.

9.3 Supplementary Figures

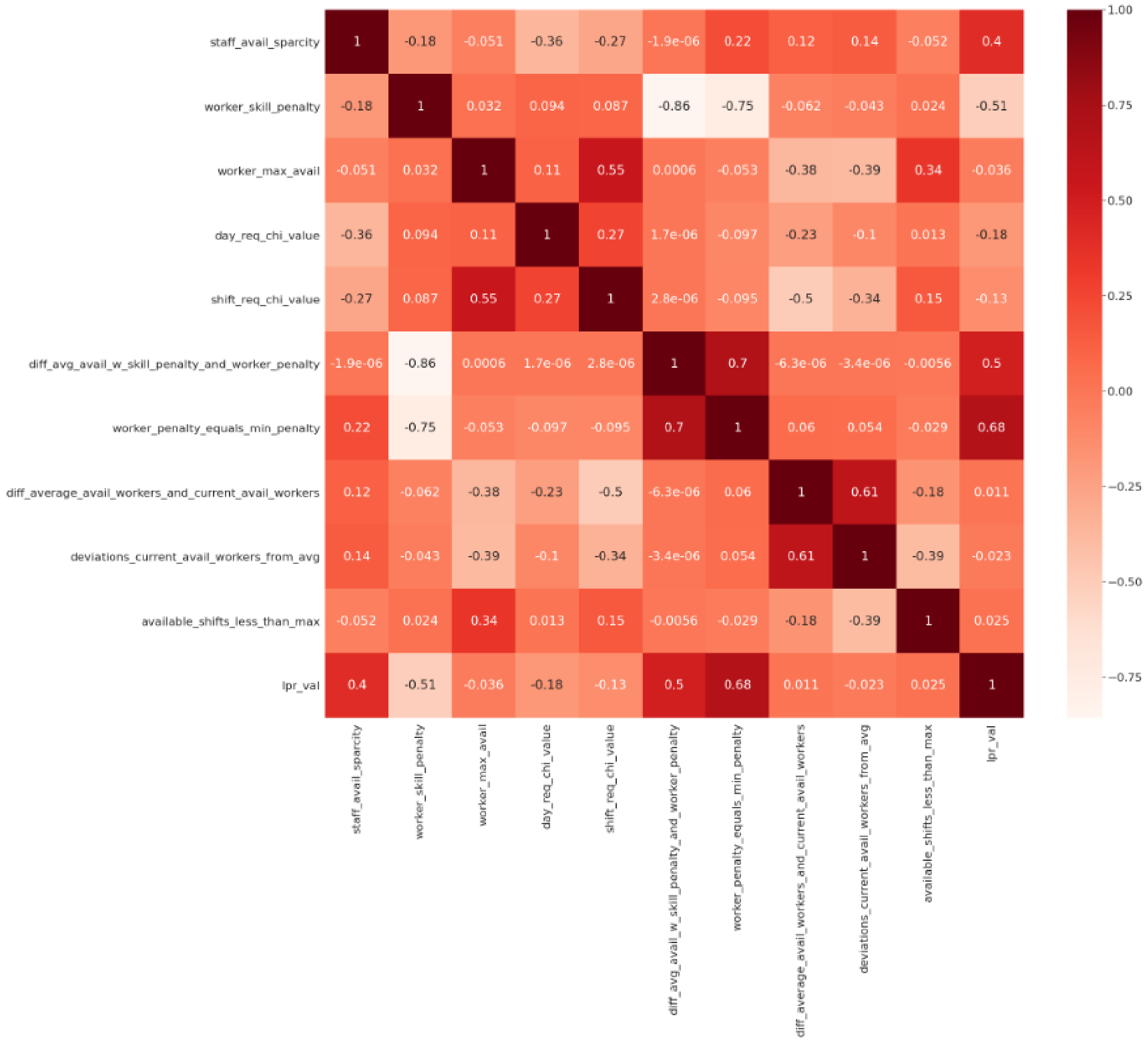
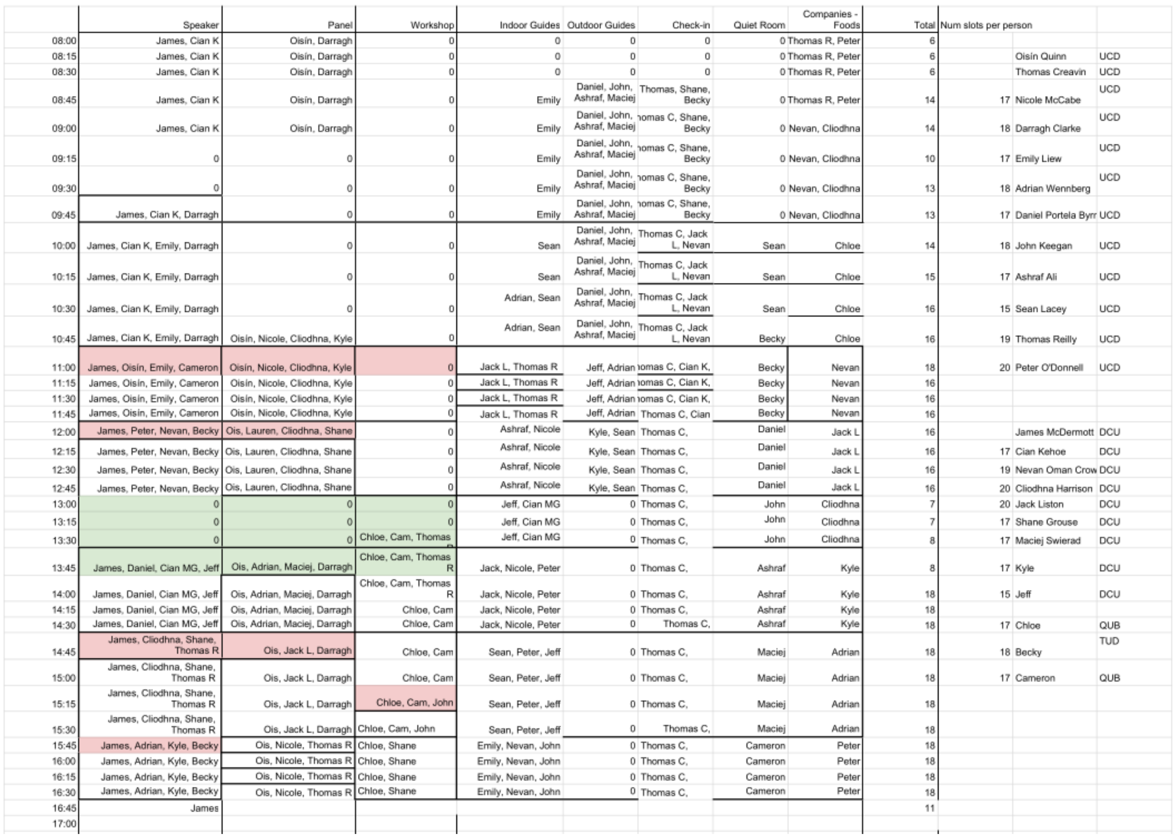
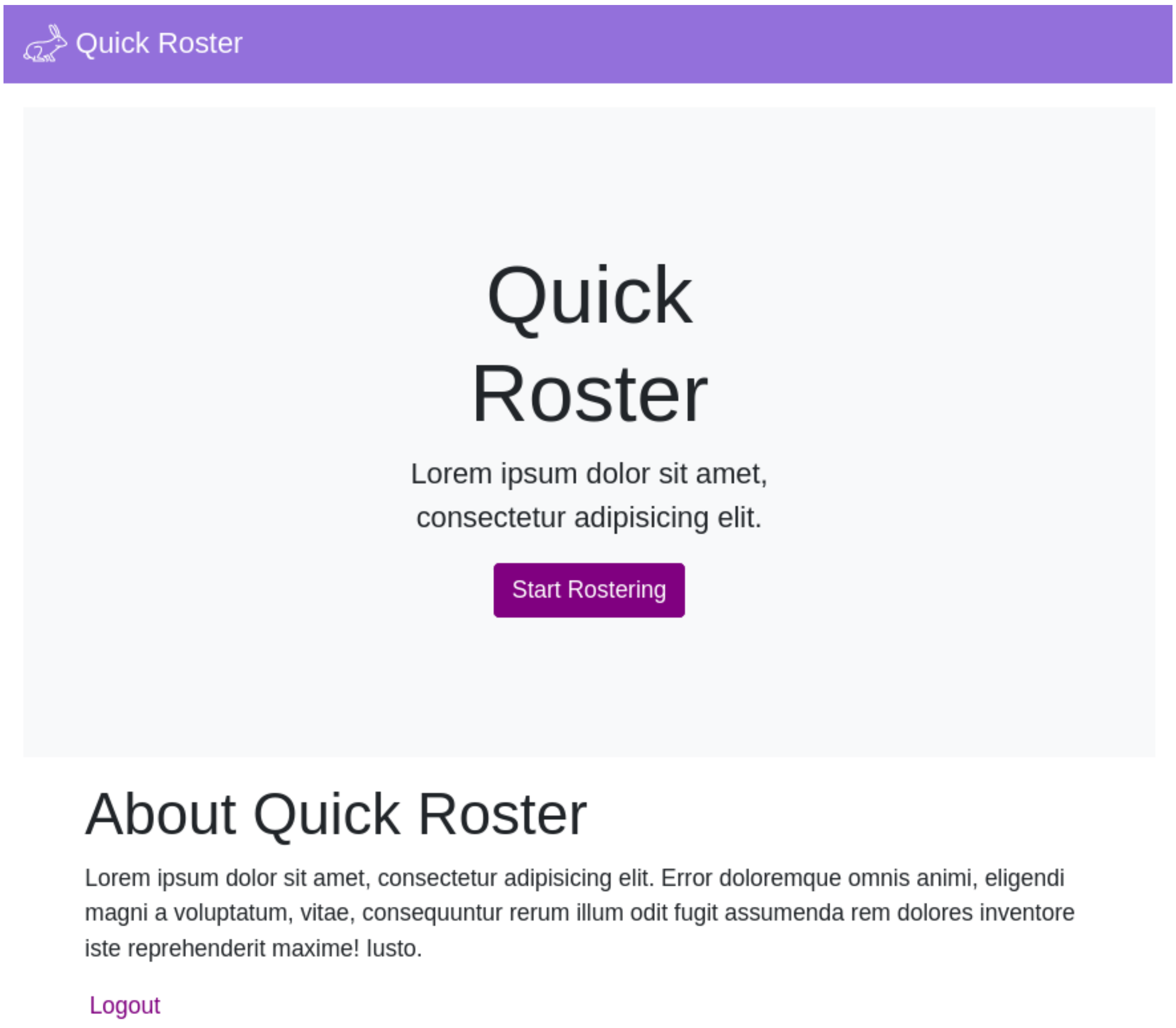


Figure 9.1: Heatmap of the ultimate features' Pearson correlation coefficients.



	Speaker	Panel	Workshop	Indoor Guides	Outdoor Guides	Check-in	Quiet Room	Companies - Foods	Total	Num slots per person		
08:00	James, Cian K	Oisín, Darragh	0	0	0	0	0	Thomas R, Peter	6			
08:15	James, Cian K	Oisín, Darragh	0	0	0	0	0	Thomas R, Peter	6		Oisin Quinn	UCD
08:30	James, Cian K	Oisín, Darragh	0	0	0	0	0	Thomas R, Peter	6		Thomas Creavin	UCD
08:45	James, Cian K	Oisín, Darragh	0	Emily	Daniel, John, Ashraf, Maciej	Thomas, Shane, Becky	0	Thomas R, Peter	14	17	Nicole McCabe	UCD
09:00	James, Cian K	Oisín, Darragh	0	Emily	Daniel, John, Ashraf, Maciej	homas C, Shane, Becky	0	Nevan, Cliodhna	14	18	Darragh Clarke	UCD
09:15	0	0	0	Emily	Daniel, John, Ashraf, Maciej	homas C, Shane, Becky	0	Nevan, Cliodhna	10	17	Emily Liew	UCD
09:30	0	0	0	Emily	Daniel, John, Ashraf, Maciej	homas C, Shane, Becky	0	Nevan, Cliodhna	13	18	Adrian Wennberg	UCD
09:45	James, Cian K, Darragh	0	0	Emily	Daniel, John, Ashraf, Maciej	homas C, Shane, Becky	0	Nevan, Cliodhna	13	17	Daniel Portela Byrr	UCD
10:00	James, Cian K, Emily, Darragh	0	0	Sean	Daniel, John, Ashraf, Maciej	Thomas C, Jack L, Nevan	Sean	Chloe	14	18	John Keegan	UCD
10:15	James, Cian K, Emily, Darragh	0	0	Sean	Daniel, John, Ashraf, Maciej	Thomas C, Jack L, Nevan	Sean	Chloe	15	17	Ashraf Ali	UCD
10:30	James, Cian K, Emily, Darragh	0	0	Adrian, Sean	Daniel, John, Ashraf, Maciej	Thomas C, Jack L, Nevan	Sean	Chloe	16	15	Sean Lacey	UCD
10:45	James, Cian K, Emily, Darragh	Oisín, Nicole, Cliodhna, Kyle	0	Adrian, Sean	Daniel, John, Ashraf, Maciej	Thomas C, Jack L, Nevan	Becky	Chloe	16	19	Thomas Reilly	UCD
11:00	James, Oisín, Emily, Cameron	Oisín, Nicole, Cliodhna, Kyle	0	Jack L, Thomas R	Jeff, Adrian	homas C, Cian K,	Becky	Nevan	18	20	Peter O'Donnell	UCD
11:15	James, Oisín, Emily, Cameron	Oisín, Nicole, Cliodhna, Kyle	0	Jack L, Thomas R	Jeff, Adrian	homas C, Cian K,	Becky	Nevan	16			
11:30	James, Oisín, Emily, Cameron	Oisín, Nicole, Cliodhna, Kyle	0	Jack L, Thomas R	Jeff, Adrian	homas C, Cian K,	Becky	Nevan	16			
11:45	James, Oisín, Emily, Cameron	Oisín, Nicole, Cliodhna, Kyle	0	Jack L, Thomas R	Jeff, Adrian	Thomas C, Cian	Becky	Nevan	16			
12:00	James, Peter, Nevan, Becky	Ois, Lauren, Cliodhna, Shane	0	Ashraf, Nicole	Kyle, Sean	Thomas C,	Daniel	Jack L	16		James McDermott	DCU
12:15	James, Peter, Nevan, Becky	Ois, Lauren, Cliodhna, Shane	0	Ashraf, Nicole	Kyle, Sean	Thomas C,	Daniel	Jack L	16	17	Cian Kehoe	DCU
12:30	James, Peter, Nevan, Becky	Ois, Lauren, Cliodhna, Shane	0	Ashraf, Nicole	Kyle, Sean	Thomas C,	Daniel	Jack L	16	19	Nevan Oman Crow	DCU
12:45	James, Peter, Nevan, Becky	Ois, Lauren, Cliodhna, Shane	0	Ashraf, Nicole	Kyle, Sean	Thomas C,	Daniel	Jack L	16	20	Cliodhna Harrison	DCU
13:00	0	0	0	Jeff, Cian MG	0	Thomas C,	John	Cliodhna	7	20	Jack Liston	DCU
13:15	0	0	0	Jeff, Cian MG	0	Thomas C,	John	Cliodhna	7	17	Shane Grouse	DCU
13:30	0	0	Chloe, Cam, Thomas	Jeff, Cian MG	0	Thomas C,	John	Cliodhna	8	17	Maciej Swierad	DCU
13:45	James, Daniel, Cian MG, Jeff	Ois, Adrian, Maciej, Darragh	Chloe, Cam, Thomas R	Jack, Nicole, Peter	0	Thomas C,	Ashraf	Kyle	8	17	Kyle	DCU
14:00	James, Daniel, Cian MG, Jeff	Ois, Adrian, Maciej, Darragh	Chloe, Cam, Thomas R	Jack, Nicole, Peter	0	Thomas C,	Ashraf	Kyle	18	15	Jeff	DCU
14:15	James, Daniel, Cian MG, Jeff	Ois, Adrian, Maciej, Darragh	Chloe, Cam	Jack, Nicole, Peter	0	Thomas C,	Ashraf	Kyle	18			
14:30	James, Daniel, Cian MG, Jeff	Ois, Adrian, Maciej, Darragh	Chloe, Cam	Jack, Nicole, Peter	0	Thomas C,	Ashraf	Kyle	18	17	Chloe	QUB
14:45	James, Cliodhna, Shane, Thomas R	Ois, Jack L, Darragh	Chloe, Cam	Sean, Peter, Jeff	0	Thomas C,	Maciej	Adrian	18	18	Becky	TUD
15:00	James, Cliodhna, Shane, Thomas R	Ois, Jack L, Darragh	Chloe, Cam	Sean, Peter, Jeff	0	Thomas C,	Maciej	Adrian	18	17	Cameron	QUB
15:15	James, Cliodhna, Shane, Thomas R	Ois, Jack L, Darragh	Chloe, Cam, John	Sean, Peter, Jeff	0	Thomas C,	Maciej	Adrian	18			
15:30	James, Cliodhna, Shane, Thomas R	Ois, Jack L, Darragh	Chloe, Cam, John	Sean, Peter, Jeff	0	Thomas C,	Maciej	Adrian	18			
15:45	James, Adrian, Kyle, Becky	Ois, Nicole, Thomas R	Chloe, Shane	Emily, Nevan, John	0	Thomas C,	Cameron	Peter	18			
16:00	James, Adrian, Kyle, Becky	Ois, Nicole, Thomas R	Chloe, Shane	Emily, Nevan, John	0	Thomas C,	Cameron	Peter	18			
16:15	James, Adrian, Kyle, Becky	Ois, Nicole, Thomas R	Chloe, Shane	Emily, Nevan, John	0	Thomas C,	Cameron	Peter	18			
16:30	James, Adrian, Kyle, Becky	Ois, Nicole, Thomas R	Chloe, Shane	Emily, Nevan, John	0	Thomas C,	Cameron	Peter	18			
16:45	James								11			
17:00												

Figure 9.2: Roster for the SISTEM 2020 Conference



Quick Roster

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

About Quick Roster

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Error doloremque omnis animi, eligendi magni a voluptatum, vitae, consequuntur rerum illum odit fugit assumenda rem dolores inventore iste reprehenderit maxime! Iusto.

Figure 9.3: Screenshot of the Quick Roster landing page.

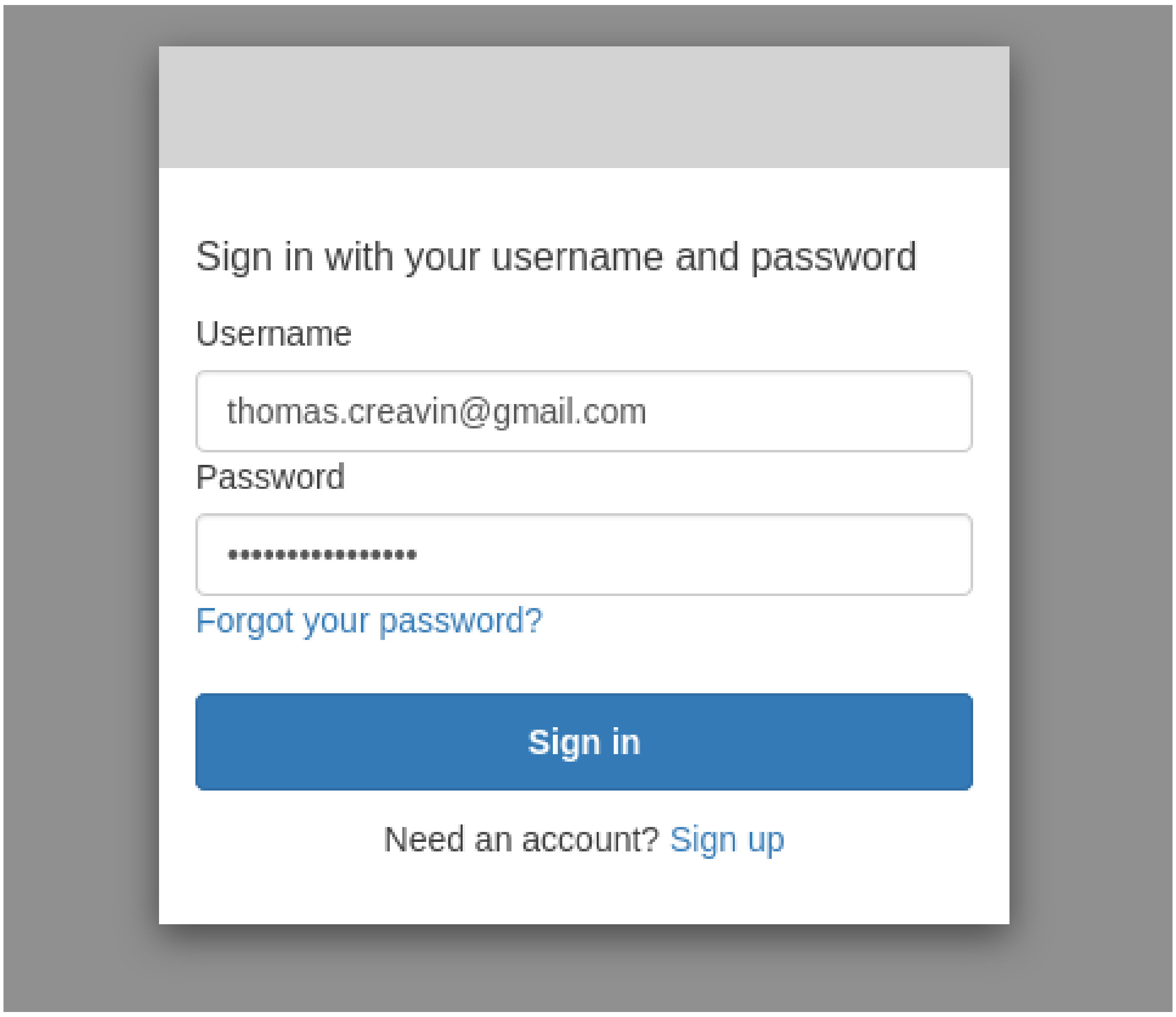
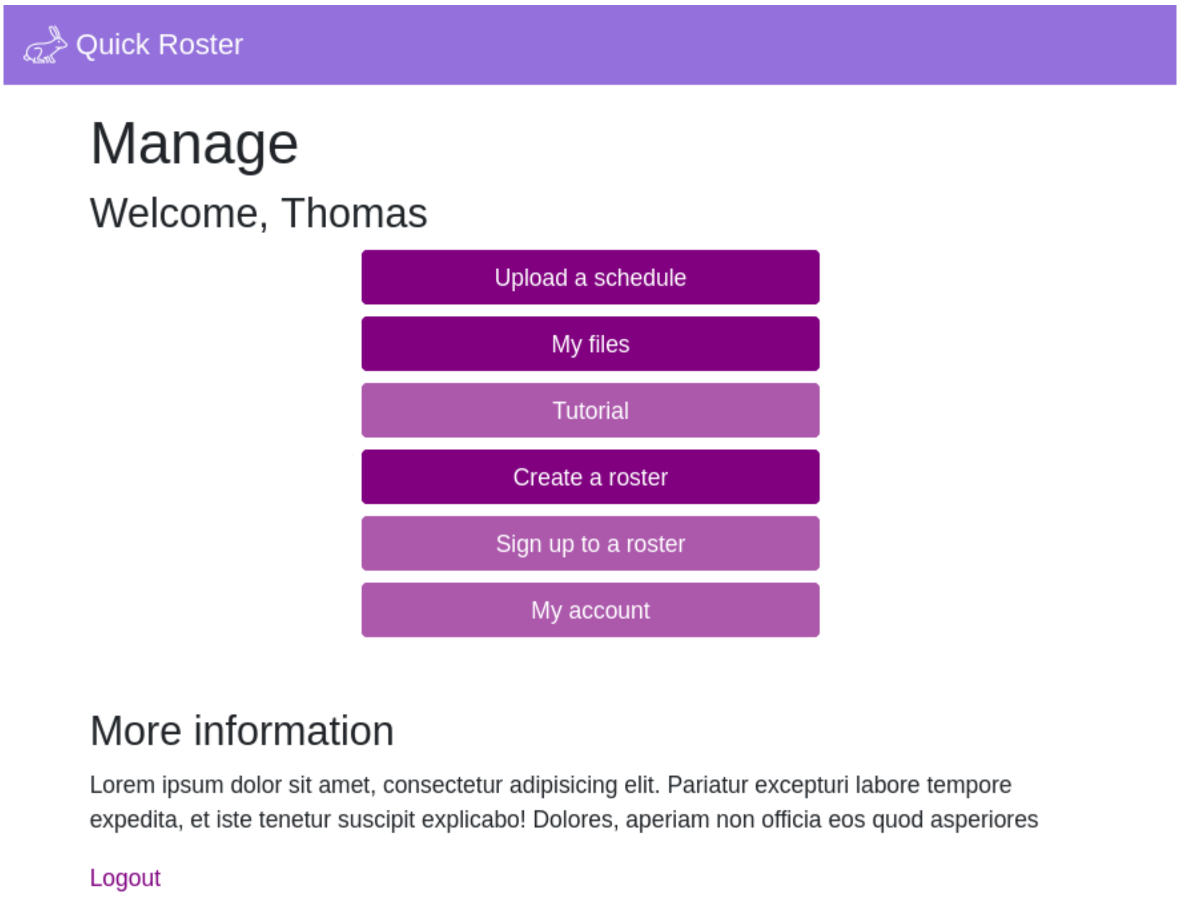


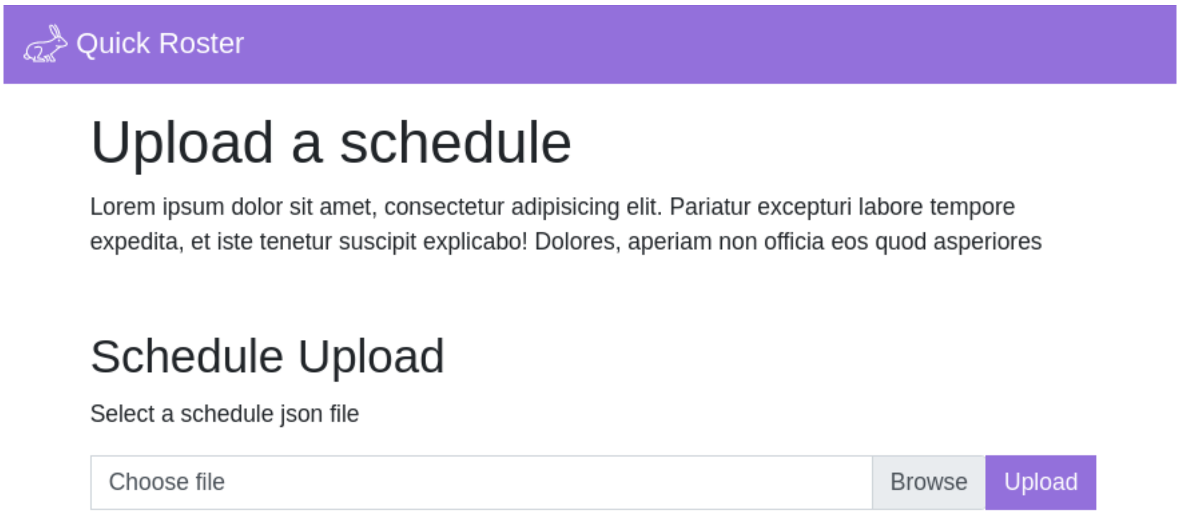
Figure 9.4: Screenshot of the Quick Roster sign-in page.



More information

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Pariatur excepturi labore tempore expedita, et iste tenetur suscipit explicabo! Dolores, aperiam non officia eos quod asperiores

Figure 9.5: Screenshot of the Quick Roster manage page.



Upload a schedule

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Pariatur excepturi labore tempore expedita, et iste tenetur suscipit explicabo! Dolores, aperiam non officia eos quod asperiores

Schedule Upload

Select a schedule json file

Figure 9.6: Screenshot of the Quick Roster upload page.

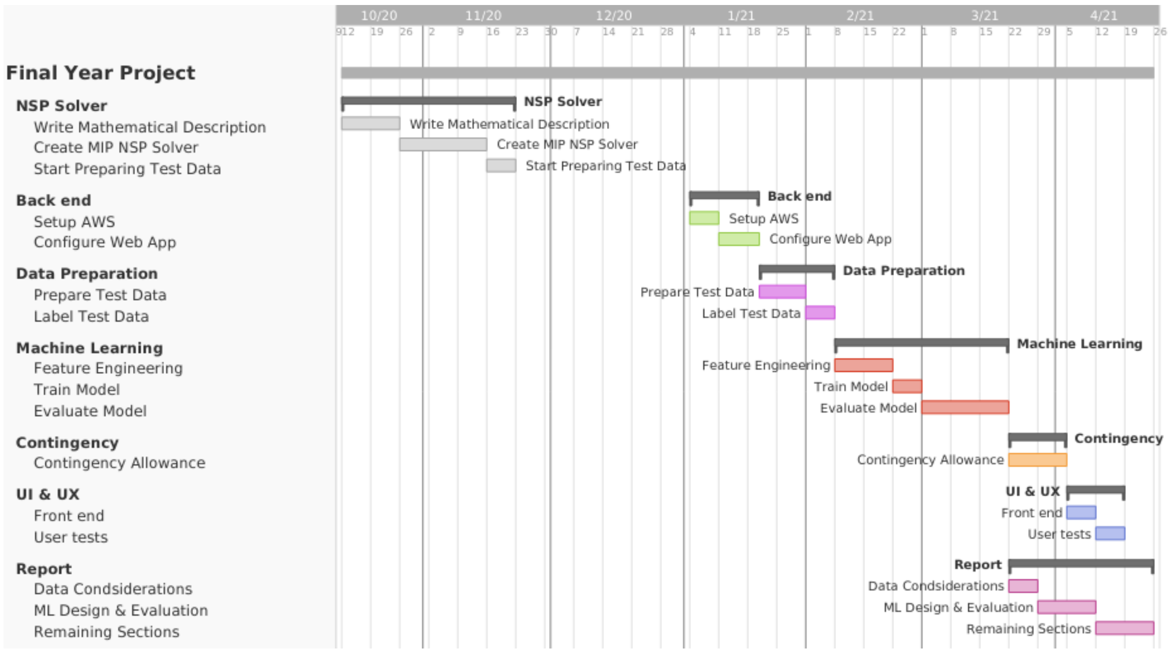
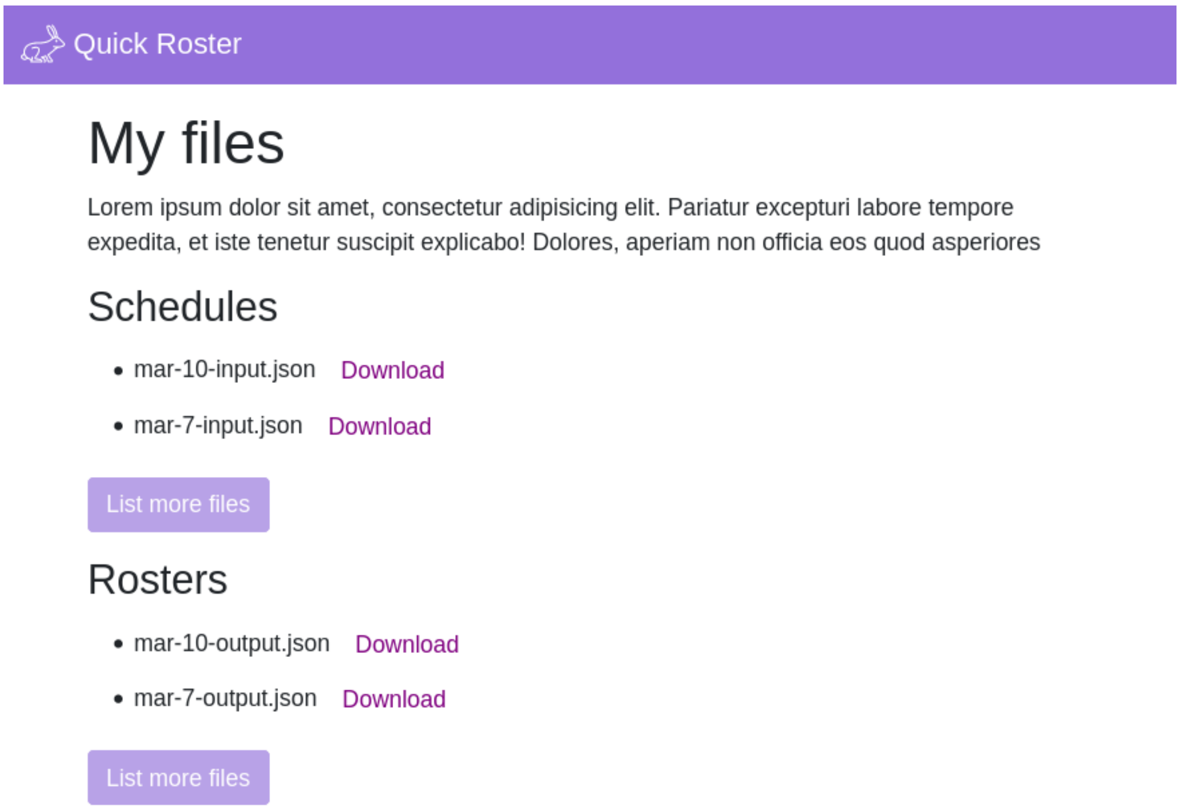


Figure 9.8: Timeline of the project work plan



My files

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Pariatur excepturi labore tempore expedita, et iste tenetur suscipit explicabo! Dolores, aperiam non officia eos quod asperiores

Schedules

- mar-10-input.json Download
- mar-7-input.json Download

List more files

Rosters

- mar-10-output.json Download
- mar-7-output.json Download

List more files

Figure 9.7: Screenshot of the Quick Roster files page.

9.4 Project Workplan

9.4.1 Completed Work

The following tasks have been completed in trimester one:

1. **Define the Nurse Scheduling Problem Variant**
   For this project, I want to address a variant of the NSP that is suitable for volunteer scheduling, is advanced enough to benefit from machine learning combinatorial optimization, and is solvable in this project's time frame. I carefully reviewed the most popular NSP variants and learned which variants result in an NP-hard problem. I expressed the chosen variant using De Causmaecker and Vanden Berghe's NSP categorization notation.

2. **Review Nurse Scheduling Problem Solving Techniques**
   I compared and contrasted many of the NSP solving techniques. I have opted to use an integer programming approach as it was straightforward to create and proved to produce optimal solutions. It's only effective for solving small instances of the problem but I plan to remedy this by augmenting it with machine learning.

3. **Define a Mathematical Description of the Integer Programming Problem**
   Once I identified the variant of the NSP I wish to solve, I created a mathematical description of the problem. This description is used to produce a mixed-integer programming formulation of the problem. The mathematical description is included at the end of the appendix. Although the implementation of the mathematical description has diverged from the original description (because of incremental improvements), it is still a valuable resource for understanding the problem this paper is addressing.

4. **Experiment with Integer Programming Solvers**
   I implemented early versions of the problem using the Gurobi and FICO Xpress solvers. I opted to use Gurobi as I preferred the Gurobi SDK and documentation to that of Xpress. I expressed the mathematical description in terms of a Gurobi MIP formulation and I iterated over this formulation to produce a competent NSP solver.

5. **Research Cloud-native architectures**
   I explored AWS, Azure, and Google Cloud Compute for deploying the program. AWS is the most cost-effective and simplest method to deploy to the cloud. I also have experience using AWS which minimises the learning curve. After I selected AWS, I designed an AWS architecture diagram for the system which is presented in figure **??** in the appendix.

6. **Procure Training Data**
   I need sample nurse scheduling problem data to build a training set and to benchmark the program. Many of the sample data sets used in the papers I have read no longer exist. I was able to identify a suitable set of test data available from Scheduling Benchmarks[1]. The data is complete with 255 instances and their optimal score for each instance.

   I have created a near-complete decoder that can ingest the sample data and transform it to work with the solver. Additionally, I have constructed a synthetic data generator but it would need more improvement to produce realistic data.

[1]http://www.schedulingbenchmarks.org/

9.4.2 Planned Work

1. **Back End (2.5 Weeks)**
   Ideally, I will begin to develop the back end two weeks before trimester two begins. I will use Ruby on Jets[2] to produce a serverless AWS back end. Ruby on Jets creates most of the AWS resources automatically using AWS CloudFormation. I will spend the majority of this stage integrating the solver into a Fargate container and configuring the Lambdas to pass data to and from Fargate. The remainder of the time is used to configure the web application.

   The back end is made up of seven components:

   - AWS Cognito allows administrators and users to easily access the roster maker.
   - Route 53 offers some DNS niceties and routing to static resources.
   - S3 stores static site resources and complete schedules.
   - API Gateway directs requests to the appropriate Lambda.
   - Lambdas serve site content and run the application.
   - The NSP solver is integrated into a Fargate container. Fargate containers are more apt than Lambdas for long-running, resource-intensive tasks.
   - CloudWatch logs the application and records metrics. It can be used to scale the application under high load.

   I acknowledge working over the Winter break is aspirational. If I am unable to allocate the time to work on the back end before the term, I will comprise the web application. A cloud back end is not an essential component of the project. In lieu of a web application, I can produce a local application by wrapping the Gurobi solver in a GUI.

2. **Data Preparation (Two Weeks)**
   I will spend the first week is testing out the NSP sample data decoder I built in trimester one, generating solutions to the data, and manually verifying solutions for small instances of the problem. Additionally, the Gurobi solver may require minor revisions.

   The second week is spent converting the optimal schedules into labelled data which can be used to train machine learning models. The labeled data must be structured so that a machine learning model can learn which volunteer features result in an assignment to a particular type of shift.

3. **Machine Learning (Six Weeks)**
   Two weeks are allocated to feature engineering. I will perform feature identification to create a collection of features that could be used to predict whether a volunteer is scheduled for a particular shift or not. If I have many features, I will perform feature selection techniques like wrapper, and filter selection to find a subset of the most influential features.

   One week will be spent training and tuning the model. This involves experimenting with machine learning classifiers such as k-nearest neighbours, decision trees, naive Bayes, etc to find the classifier with the best performance. Once I have chosen a classifier, I will optimize the model further by performing parameter tuning.

   A further three weeks are required to evaluate the model. Models will be evaluated using three metrics. First, the query time, that is, the time to compute the problem's features. Ultimately I would like the machine learning model to speed up the overall time to schedule volunteers; for this, the machine learning model must be quick at computing the problem's features.

   Second, I want to know what effect the model will have on the quality of the solutions. To evaluate this, I will measure the loss in the objective function score i.e the degradation of

[2]https://rubyonjets.com/

the solution quality. Additionally, I will manually verify that the solutions to small instances of the problem are adequate.

Last, I want to compare the accuracy of the predicted schedule to the optimal schedule. A scheduling problem can have a number of optimal solutions so the accuracy can vary greatly. But I still believe this could be an interesting evaluation metric to explore.

4. **Contingency (Two Weeks)**
   Two weeks are set aside as a contingency.

5. **User Interface & User Experience (Two Weeks)**
   I have allowed one week to create the user interface. I have decided a beautiful user interface and an intuitive user experience can not be achieved in the time frame of this project. Instead, I will use Bootstrap[3] to produce a sensible but plain front end that allows organisers to create schedules and volunteers to manually enter their availability.

   I have allocated a week to conduct two user tests. First, up to five users [33] will be invited to test the application and provide their input. I will correct any bugs or basic design flaws identified in this test. The second user test will use a larger testing pool. Each user will use the application and complete a questionnaire.

   I intend to use IBM's Computer System Usability Questionnaire (CSUQ) [34] to gather user feedback. The CSUQ is designed to help software practitioners measure users' satisfaction with the usability of computer systems. CSUQ consists of nineteen statements where testers indicate how strongly they agree or disagree with each statement. I may append additional questions asking testers' opinions on specific elements of the application.

6. **Report (Five Weeks)**
   Towards the end of the trimester, I can begin the final report. My personal preference is to write the report iteratively over five weeks as opposed to blocking off two-three weeks to write the report. I can write more easily this way. The report will be written concurrently with project development. The final week is entirely dedicated to report writing.

[3]https://getbootstrap.com/