2022

Backlog Burner: An Adventure Into Automated Scheduling

Anjolaoluwa J. Olubusi
The College of Wooster, aolubusi22@wooster.edu

Backlog Burner: An Adventure Into Automated Scheduling

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Bachelor of Arts in Computer Science and
Mathematics in the
Department of Mathematics & Computational Sciences at
The College of Wooster

by
Anjolaoluwa Olubusi
The College of Wooster
2022

Advised by:

Dr. Thomas Montelione

THE COLLEGE OF
WOOSTER

Abstract

The focus of this independent study is to explain the nurse scheduling problem (NSP) and use it as a basis to create an automated scheduling program. The nurse scheduling problem is an operational research problem that sets to find an optimal hospital schedule that fulfills the needs of the hospital and the personal requests of the nurses. The majority of solutions for the nurse scheduling problem are often designed within a hospital setting. The objective of this independent study is to use the solutions of the nurse scheduling problem to develop an automated scheduling program for the College of Wooster student population.

Acknowledgments

I would like to acknowledge Dr. Thomas Montelione and Dr. Colby Long for their advice. In addition, I would like to extend my gratitude to my family and friends for the encouragement and wisdom bestowed during the creation of this project.

Contents

List of Figures

List of Tables

List of Listings

CHAPTER 1

Introduction

The average layman is plagued with the troubles of poor work-life balance. Due to this, researchers from all academic fields have attempted to find the most optimal work schedule that permits a good work-life balance. The problem of finding the schedule that permits a good work-life balance is best described through the nurse scheduling problem. The nurse scheduling problem (NSP) is an operational research problem that sets out to find an optimal hospital schedule that fulfills the needs of the hospital and the personal requests of the nurses [23].

Most solutions for the nurse scheduling problem are designed within a hospital setting. The objective of this project is to develop software that can create a schedule centered on maintaining work-life balance while accounting for hard and soft constraints. The software of this project is a web application. Currently, automatic scheduling applications are focused within a corporate climate. The software of this independent study is designed for use within a college climate. The college climate was chosen due to the unique scheduling habits of college students. Most schedules for working people have from morning to late afternoon devoted to their profession. Due to this, most hobbies start after work.

This isn't the case with college students. Most college students don't have a sizeable amount of commitments within the working day. The commitments are usually an hour to an hour and thirty minutes long for classes. Furthermore, there

is a need for automatic scheduling since these students have not lived on their own before. Thus, they will fall prey to improper scheduling. Within this project, hard constraints are defined as occasions that occur at a specified time. These events could include: class meetings, professor office hours or a campus job. Soft constraints are events that do not have a specified starting and ending time. These events could include watching a Netflix movie, reading a book, and working out. The internal algorithm of the scheduling software will derive from solutions to the nurse scheduling problem. This document will first review three types of solutions to the nurse scheduling problem: the integer programming models, the ant colony optimization models, and the genetic algorithm models.

The integer programming model (IPM) uses integer programming to determine the most optimal schedule for the hospital. Integer programming is a special class of mathematical models that solves optimization problems where some or all the variables of the problem are integers [5]. For instance, the problem of determining the most optimal package placement in a truck would be solved using an integer programming model. This is due to the fact that is it impossible to have a fractional number of items within the truck. The nurse scheduling problem does not have formalized form. Thus, there are numerous avenues one could go to model and solve the nurse scheduling problem. A selection of these models are discussed in Chapter 2.

The ant colony optimization model combines the ant colony optimization algorithm with another heuristic to calculate its solution to the nurse scheduling problem. Ant Colony Optimization (ACO) is a class of optimization algorithms that mimics ant foraging to determine the most optimal approximate solution to a problem [3]. The Ant Colony Optimization originated in the early 1990s [3]. Specifically, the Ant Colony Optimization was proposed in 1992 as Macro Dorigo's PhD thesis. Ant foraging is the food finding process of ants. Ants use pheromones to

denote a path between their home and a food source. Due to their chemical nature, pheromones evaporate quickly. As the pheromones evaporate, the pheromone density of that path decreases. Ants prefer a shorter path with denser pheromones. Therefore, the path an ant, or a collective of ants, follows is generally the shortest path towards a food source. The ant colony optimization algorithm acts similarly. The algorithm creates mini agents that move across a graph that represents the problem being solved. These mini agents are a computerized version of ants. The ant colony optimization is often combined with a local searching algorithm, like hill climbing, in order to improve the quality of the solution [1]. The ant colony optimization model is further elaborated in Chapter 3.

The genetic algorithm model uses genetic algorithms to solve the nurse scheduling problem. Genetic algorithms are a set of algorithms that imitates the evolutionary process and the idea of the survival of the fittest aspects of evolution [1]. The algorithms begin with an initial population of solutions. These solutions are randomly generated. First, each of these solutions are tested using a fitness function. Then, the algorithm combines the solutions to form new solutions. In addition, the genetic algorithm randomly modifies the older solutions. Some of the best solutions of each generation are kept whilst the others are replaced by the newly formed solutions. The process is repeated until stopping criteria are met. Similar to the act colony optimization algorithm, the genetic algorithm is often combined with a local search algorithm. Local search algorithms modify the current solution until a more optimal solution is found, or a specified time limit has been breached [1]. The local search algorithm purifies the solution produced by the genetic algorithm. The genetic algorithm model is further elaborated in Chapter 4.

This document will describe the internal algorithm used in the scheduling software. The scheduling software, named Backlog Burner, combines the ant colony optimization algorithm and genetic algorithm to derive the user's new schedule.

The combination of the ant colony optimization algorithm and genetic algorithm is called the Genetic Ant Colony Optimization Algorithm (GACO) [24]. The GACO algorithm is further discussed in Chapter 5.

A proportion of the College of Wooster student population were asked to test Backlog Burner. The results of this testing are discussed in Chapter 6.

CHAPTER 2

Integer Programming Models

Integer programming models are special classes of mathematical models that solves optimization problems where some or all the variables of the problem are integers [5]. Specifically, integer programming models are special versions of linear programming models where the variables are constrained to be integers. Linear programming models are mathematical models that solve linear optimization problems. This chapter will examine the concepts of linear programming, integer programming, analyzing nurse scheduling integer programming models and detail the applicability of these models in the real world. Section 2.1 details the concepts of linear programming, integer programming and the methods by which integer programming models are solved. Section 2.2 explains integer programming models designed to solve the nurse scheduling problem. Furthermore, section 2.2 details the applicability of the integer programming models for the nurse scheduling problem.

2.1 What Is Integer Programming?

2.1.1 Definition

The standard form for linear programming models is:

$$\text{maximize} \quad z = c_1x_1 + c_2x_2 + \dots c_nx_n \tag{2.1}$$

$$\text{subject to} \quad a_{11}x_1 + a_{12}x_2 + \dots a_{1n}x_n = b_1 \tag{2.2}$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \tag{2.3}$$

$$\vdots \tag{2.4}$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \tag{2.5}$$

Figure 2.1: Standard format of a linear programming model [5]

$x_1, \dots, x_n$ are non-negative variables. Each x_i represents a variable of the problem. $c_1, \dots, c_n$ represents the cost of each particular variable. Specifically, c_i represents the cost of the variable x_i. Equation 2.1 represents the objective function. The objective function is the mathematical form of the problem's criteria [5]. The objective function measures the effectiveness of the solution [5]. Using matrix notion, we can represent the objective function, $f(x)$, as $f(x) = cx$ where

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{ and } c = (c_1, c_2, \dots, c_n) \tag{2.6}$$

x is the decision matrix and c is the cost matrix. Equation 2.2 to 2.5 represents the constraints of the problem. The constraints of the modeled problem are represented through linear equations. There are situations where the constraints are not simply linear. For example, $x_1 \leq 50$ and $x_2 \geq 30$. When these situations arise, new variables are added to the constraints so that the inequality becomes a linear equation. There are two types of added variables, surplus variables and slack variables. Surplus

variables are used to convert a greater than or equal to inequality constraint into a linear equation [5]. Referring to the example above, $x_2 \geq 30$ becomes $x_2 - s_1 = 30$ where s_2 is a surplus variable. Slack variables are used to covert less than or equal to inequalities into linear constraints. Here, $x_1 \leq 50$ becomes $x_1 - s_2 = 50$. The objective function does not change with the addition of surplus and slack variables since the surplus and slack variables do not add any value to the objective function. a_{ij} represent the weight of each variable x_j for each constraint i. The coefficient matrix is:

$$A = \begin{bmatrix} a_{11} \dots a_{1n} \\ \dots \\ a_{m1} \dots a_{mn} \end{bmatrix} \tag{2.7}$$

b_i represents the limit of each constraint i. The requirement matrix is:

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{2.8}$$

With this, we can rewrite the standard form of integer programming models as:

$$\text{maximize} \quad z = cx \tag{2.9}$$

$$\text{subject to} \quad Ax = b \tag{2.10}$$

$$x \geq 0 \tag{2.11}$$

$$b \geq 0 \tag{2.12}$$

Figure 2.2: Minimized format of an integer programming model [5]

2.1.2 Solving Linear Programming Models

In the standard form of linear programming models, there are m constraint equations and n variables to solve. The n variables include the decision variables and any artificial variables that are added to the model. With this, there are $n - m$ variables that can be set to any value whilst solving the integer programming model. A solution to the model is found by setting $n - m$ variables to zero and solving for the other m variables [5]. The $n - m$ variables which are set to zero are called non-basic variables. The m variables which are not set to zero are called basic variables. The solution that arises from solving for the m variables is called a basic solution. A basic solution does not always have to fulfill the constraints of the problem. A solution that does not fulfill the requirements of the problem is called an infeasible solution. A solution that fulfills the requirements of the problem is called a basic feasible solution. The solution of these model are often found through the Simplex Method. Developed in 1947 by George B. Dantzig, the Simplex Method is an iterative algorithm [5]. The inner workings of the Simplex method can be represented by the following pseudocode:

1. Start with an initial feasible solution
2. While solution is not optimal
3. Adjust Solution
4. Check if solution is optimal

Figure 2.3: The Simplex Method

The initial feasible solution is often the extreme points. The concept of extreme points can be properly explained through the following example. Suppose there is a mathematical model described as the following:

$$\text{maximize} \quad z = 8x_1 + 5x_2 \tag{2.13}$$

$$\text{subject to} \quad x_1 \leq 150 \tag{2.14}$$

$$x_2 \leq 250 \tag{2.15}$$

$$2x_1 + x_2 \leq 500 \tag{2.16}$$

$$x_1, x_2 \geq 0 \tag{2.17}$$

Figure 2.4: An Example Mathematical Model [5]

If Figure 2.4 is converted into the standard form, the model becomes:

$$\text{maximize} \quad z = 8x_1 + 5x_2 + 0s_1 + 0s_2 + 0s_3 \tag{2.18}$$

$$\text{subject to} \quad x_1 + s_1 = 150 \tag{2.19}$$

$$x_2 + s_2 = 250 \tag{2.20}$$

$$2x_1 + x_2 + s_3 = 500 \tag{2.21}$$

Figure 2.5: Figure 2.5 In The Standard Form [5]

The constraints of this model describe half planes in the two dimensions [5]. The intersection of these half planes are the extreme points.

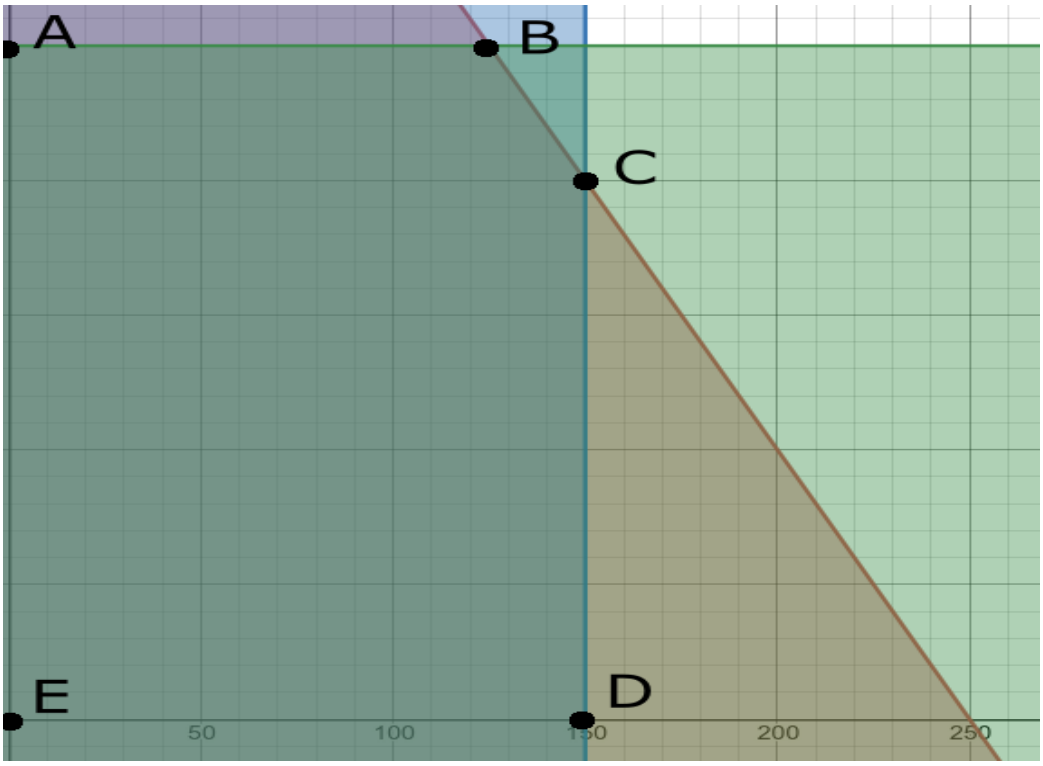

Figure 2.6: Graphical Visualization Of The Figure 2.4 Constraints

Using Figure 2.6, the points A, B, C, D, E are the extreme points. With linear programming problems, the unique solution to the problem, assuming there are no multiple solution to the problem, occurs at these extreme points.

Using the example from Figure 2.4, there are 3 constraints equations and 5 variables. Thus, there are 2 non-basic variables. Generally, the decision variables are set to 0. However, there are cases where a combination of decision and surplus variables are considered to be the non-basic variables. For this example, x_1 and x_2 going to be 0. Therefore, our initial feasible solution is: $s_1 = 150$, $s_2 = 250$, and $s_3 = 500$.

The Simplex Method relies on a simple principle:

Principle. *Assume that we have a starting solution. If there is a better solution, that is not our starting solution, then there is an adjacent solution that is better than our current basic solution.*

Given the initial feasible solution, the transition to the adjacent solution is made through a pivot operation [5]. A pivot operation is a set of row operations applied to the current form our model. Whilst applying pivot operations, the model is often represented through the tableau form. The tableau form is a special form of the model that decreases the difficulty of changing variable coefficients. In the tableau form, the objective function changes. To recall, the objective function of Figure 2.5 is $z = 8x_1 + 5x_2$. To convert a model from the standard form into the tableau form, the objective function is modified such that all variables are on the same side of the equation and that the coefficient of z is 1. For Figure 2.5, the converted objective function is $z - 8x_1 - 5x_2 - 0s_1 - 0s_2 - 0s_3 = 0$.

Basis	z	x_1	x_2	s_1	s_2	s_3	**Solution**
Z	1	-8	-5	0	0	0	0
s_1	0	1	0	1	0	0	150
s_2	0	0	1	0	1	0	250
s_3	0	2	1	0	0	1	500

Table 2.1: Figure 2.5 in Tableau Form

The first column shows the current basic variables. The last column shows the values of the basic variables. With the tableau form, the Simplex Method follows these steps:

1. First, examine the objective function row. This row is generally the topmost row in the tableau form. Choose a non-basic variable to enter into the basis that has the most negative coefficient within the objective function row. The chosen variable is called the entering variable. For our linear model, the entering variable is x_1 since its coefficient, in the objective function, is -8. The column of that chosen variable is called the pivot column.

2. If all the values of the pivot column are less than or equal to zero, then there is no solution. Otherwise, transition to step 3.

3. For each element in the solution column, b_i, we divide the element by the corresponding value in the pivot column, a_{ik}, if the corresponding value is larger than zero. This selected row is called the pivot row. Once, the ratio is calculated, the row with the smallest ratio is selected. Let θ_i represent the ith ratio where $\theta_i = b_i/a_{ik}$. Thus, the computed ratios are:

$$\theta_1 = \frac{150}{1} = 150$$
$$\theta_3 = \frac{500}{2} = 250$$

   Here we do not calculate the ratio of the second row since the value of the second row is zero. Given that θ_1 is the smallest ratio value, s_1 is leaving the basis section.

4. To determine the adjacent solution, we divide each element in the pivot row by the pivot element. For Figure 2.5, our tableau becomes:

Basis	z	x_1	x_2	s_1	s_2	s_3	**Solution**
Z	1	-8	-5	0	0	0	0
s_1	0	1	0	1	0	0	150
s_2	0	0	1	0	1	0	250
s_3	0	2	1	0	0	1	500

   Use row operations to make sure that all other elements are zero except the pivot element. At the end our tableau is:

Basis	z	x_1	x_2	s_1	s_2	s_3	**Solution**
Z	1	0	-5	8	0	0	1200
x_1	0	1	0	1	0	0	150
s_2	0	0	1	0	1	0	250
s_3	0	0	1	-2	0	1	500

5. If all items within the objective function row are greater than or equal to zero, then we have reached the most optimal solution. If not, then the algorithm transitions into step 1. Since there are still negative coefficients in the objective function, step 1 is repeated. The final form of the tableau is:

Basis	z	x_1	x_2	s_1	s_2	s_3	**Solution**
Z	1	0	0	0	1	4	2250
x_1	0	1	0	0	-1/2	1/2	125
s_1	0	0	0	1	1/2	-1/2	25
x_2	0	0	1	0	1	0	250

2.2 The Models

2.2.1 Branch-and-Bound Model

Erbu Yilmaz's branch-and-bound model solves the nurse scheduling problem by first dividing the nurses' full schedule into week long chunks [23]. Then, Yilmaz's model solves the nurse scheduling problem with this smaller schedule size. Integer programming models that divide the main problem into smaller sub problems are called branch-and-bound models.

In this model, each nurse can do 3 types of shifts: 8:00 a.m to 4:00 p.m, 4:00 p.m to 12:00 a.m and 12:00 a.m to 8:00 a.m. Unlike other mathematical solutions

for the nurse scheduling problem, this model creates the nurse labor schedule by minimizing the nurses' total idle waiting time. Furthermore, this mathematical model considers: the maximum total weekly working time for each of the nurses, the re-assignability of nurses, and the total number of possible working hours for each shift [23]. The maximum total working time a week for a nurse, the total number of nurses in a hospital, and maximum and minimum numbers of nurses that worked a shift are specified by the user. The model's notation uses the following variables:

Model Definitions

1. Let TN be the total number of nurses in a hospital.
2. Let WH be the maximum total working time in a week for a nurse.
3. Let i be the nurse index where $i = 1, \cdots, TN$.
4. Let x_{ij} be equal to 1 if nurse i works shift j of a week. x_{ij} equals to 0 otherwise.
5. Let $N_{j(min)}$ be the minimum number of nurses worked shift j.
6. Let $N_{j(max)}$ be the maximum number of nurses worked shift j.

Model Notation

The model is noted as followed:

$$\text{Min} \quad (WH * TN) - 8 \sum_i \sum_j x_{ij} \tag{2.22}$$

$$\text{subject to} \quad 8\left(\sum_j x_{ij}\right) \leq WH \quad \forall i \tag{2.23}$$

$$\sum_{j=1}^{3} x_{ij} \leq 1, \sum_{j=2}^{4} x_{ij} \leq 1, \cdots, \sum_{j=19}^{21} x_{ij} \leq 1 \forall j \tag{2.24}$$

$$N_{j(min)} \leq \sum_i x_{ij} \leq N_{j(max)} \forall j \tag{2.25}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \tag{2.26}$$

Figure 2.7: Yilmaz's Model For The Nurse Scheduling Problem [23]

Equation 2.22 is the objective function of this model. Yilmaz' objective function minimizes nurses' total idle waiting time during a week planning horizon. ($WH*TN$) represents the maximum total working hours for all nurses within a hospital. $8 \sum_i \sum_j x_{ij}$ represents the amount of allocated working hours for all nurses within a hospital. $\sum_i \sum_j x_{ij}$ is the total number of shifts for all nurses.

Equation 2.23 constraints this model such that any solution does not exceed the maximum working hours per week for each nurse.

Equation 2.24 states that each nurse has a minimum of 2 resting shifts before being assigned to another shifts. For instance: some constraints on nurse i are:

$$x_{i1} + x_{i2} + x_{i3} \leq 1$$

This mean that if shift 1 is assigned to nurse i then shift 2 and shift 3 can not be assigned to nurse i.

Equation 2.25 ensures that the number of nurses working each shift is within a user specified range for that particular shift.

Equation 2.26 restricts the decision variable such that it can only have the values of 0 and 1.

Application Of Model

Whilst this model has not been applied into the real-world, it has been tested against an illustrative example. In the example given in Yilmaz' paper, there are a total number of 10 nurses. Each nurse has a maximum total working hours of 40. The table below shows the weekly labor constraint of the nurses:

	Shifts of day-1 j=1,2,3	Shifts of day-2 j=4,5,6	Shifts of day-3 j=7,8,9	Shifts of day-4 j=10,11,12	Shifts of day-5 j=13,14,15	Shifts of day-6 j=16,17,18	Shifts of day-7 j=19,20,21
8:00 a.m to 4:00 p.m	3	3	4	4	3	1	1
4:00 p.m to 12:00 a.m	2	3	2	2	2	1	1
12:00 a.m to 8:00 a.m	1	1	1	1	1	1	1

Table 2.2: Minimum number of nurses for each shift in the example [23]

	Shifts of day-1 j=1,2,3	Shifts of day-2 j=4,5,6	Shifts of day-3 j=7,8,9	Shifts of day-4 j=10,11,12	Shifts of day-5 j=13,14,15	Shifts of day-6 j=16,17,18	Shifts of day-7 j=19,20,21
8:00 a.m to 4:00 p.m	5	4	5	5	4	2	1
4:00 p.m to 12:00 a.m	3	4	3	3	4	1	1
12:00 a.m to 8:00 a.m	2	2	2	2	2	1	1

Table 2.3: Maximum number of nurses for each shift [23]

Using LINGO, a popular optimization software [23][5], the optimal solution is calculated within a very short time span. In Yilmaz' paper, the optimal schedule is calculated in 112 iterations.

Although this model can quickly create an optimal schedule, it is very limited. Yilmaz's model does take full-time nurses and part-time nurses into consideration [23]. Furthermore, this model does not consider the situation that a nurse unexpectedly can not work a shift [23].

2.2.2 Preference Scheduling

The Brad Purnomo model uses an algorithm, described in "Solving The Model", to solve the nurse scheduling problem. The unique aspect of this model is that it incorporates the nurses' personal preferences in the decision-making [2]. Similar to Yilmaz's model, this model divides the entire schedule into chunks, usually 28 or 46 day chunks, and uses user-submitted data [2]. Specifically, the user-submitted data is: the nurses' desired schedule and the coverage requirements of the hospital [2]. Instead of having 3 shifts types, the preferences model has 5 shift types. In Brad Purnomo's model, these shift types are:

1. Day (7 a.m.–3 p.m.)
2. Evening (3 p.m.–11 p.m.)
3. Night (11:00 p.m.– 7 a.m.)
4. AM (7 a.m.–7 p.m.)
5. PM (7 p.m.–7 a.m.)

To reduce the amount of iterations it would take to calculate a solution, the working day is divided into a set of continuous working periods [2]. These periods

have uneven length so that each shift contains one or more periods. Unlike Yilmaz's model, this model uses periods to express demand instead of shifts [2]. This reduces the amount of rows within the tableau form of the model. The model uses an algorithm, described in the "Solving The Model" section, to create the optimal nurse schedule.

Model Definitions

The Brad Purnomo model uses these notations for its indices and sets:

i	index for nurses; $i \in N$
j	index for schedules; $j \in N$
p	scheduling period (portion of a day); $p \in P$
d	day of the week; $d \in D$
N	set of nurses to be scheduled
S_i	set of schedules considered for nurse i
D	set of days in the planning horizon
P	set of periods in a day

Table 2.4: Indices and sets used in the Brad Purnomo Model [2]

The Brad Purnomo model uses these notations for its parameters and user-submitted data [2]:

c_{ij}	penalty "cost" of assigning schedule j to nurse i
a_{ijdp}	1 if the schedule j for nurse i contains period p on day d, 0 otherwise
LD_{dp}	lower bound on demand for nurses on day d in period p
UD_{dp}	upper bound on demand for nurses on day d in period p
M	large number representing the cost of either an outside nurse or under-coverage in a period

Table 2.5: Parameters and user-submitted data used in the Brad Purnomo Model [2]

The Brad Purnomo model uses these notations for the decision variable:

x_{ij}	1 if nurse i is assigned to schedule j, 0 otherwise
y_{dp}	number of outside nurses used on day d in period p
s_{dp}	lower bound on demand for nurses on day d in period p

Table 2.6: Decision variables used in the Brad Purnomo Model [2]

The Brad Purnomo model uses a variety of hard and soft constraints. For the purposes of this paper, the selection of hard constraints will be shown. These hard constraints are:

Rule	Hard Constraint
Illegal work patterns-there must be an eight-hour break between shifts	yes

Number of different shift types a nurse can work in 2-week period when contract calls for single shift type	≤ 2
Consecutive working days	≤ 6 days
Consecutive days off 12-hour shift (5 working days)	<6
Number of working weekends	2 weekends in 4 weeks
Rotational Schedules: Day and Night Shift	Must work hired shift for 50% of assigned schedule
Number of transitions for rotational nurse during each stretch of work	≤ 6 in 2 weeks
Personal requests	*MinReq* requests guaranteed over the planning horizon
Overtime: Full-time/part-time	Pattern allowed (8-hour shift) Day → AM, Night → PM, Evening → AM/PM
Overtime: Full-time (12-hour shifts)	Pattern allowed (8-hour shift) AM → Day, PM → Night. Maximum working hours allowed ≤ 88 per week for full-time. Maximum working hours allowed ≤ 32 per week for part-time.
Minimum hours a nurse must work in a 2-week period	yes

Table 2.7: Hard constraints of the Brad Purnomo model [2]

Model Formalization

$$\text{Minimize} \quad \sum_{i \in N} \sum_{j \in S_i} c_{ij} x_{ij} + M \sum_{d \in D} \sum_{p \in P} y_{dp} \tag{2.27}$$

$$\text{subject to} \quad -s_{dp} + \sum_{i \in N} \sum_{j \in S_i} a_{ijdp} x_{ij} + y_{dp} = LD_{dp} \quad \forall d \in D, p \in P, \tag{2.28}$$

$$\sum_{j \in S_i} x_{ij} = 1 \quad \forall j \tag{2.29}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i \in N, j \in S_i \tag{2.30}$$

$$0 \leq s_{dp} \leq UD_{dp} - LD_{dp}, y_{dp} \geq 0 \quad \forall d \in D, p \in P \tag{2.31}$$

Figure 2.8: The Brad Purnomo Model [2]

The objective function of this model, equation 2.27, minimizes the cost of matching providing the right number and type of nurses for all the $|P| \times |D|$ periods, and the cost of using outside nurses to cover schedule gaps.

Equation 2.30 limits the set of feasible solutions such that each solution maintains a specified number of nurses per period. This number of nurses are between LD_{dp} and UD_{dp}. This constraint also accounts for nurse shortages. s_{dp} is a surplus variable for the number of nurses. It is constrained in Equation 2.33.

Equation 2.32 and equation 2.33 constrict the model such that each nurse, within each unit, can only have one assigned schedule.

Solving The Model

The model determines the optimal schedule by:

1. The Brad Purnomo model first starts with the user submitted schedule, called

the base schedule, and a max iteration number. The model checks whether the base schedule satisfies demand, violates no hard constraints.

2. It creates a variable named α that stores the iteration number we are in.

3. Then it creates a subset of possible solutions by modifying one or two shifts in the user submitted schedule.

4. Each solution, within the subset, is added to the model and the Simplex Method used to solve the model described in Model Formalization.

5. If there is no solution found whilst α was less than the max iteration number, or the number of nurses that operate outside the hospital is 0 and α is smaller than the maximum iteration number, then the algorithm transitions to step 5. Otherwise, the final schedule is reported.

6. We modify the set of solutions we currently have by shifting one or two shifts. After that process is completed, we transition to step 3.

Application Of Model

When this model is tested against real data, the model solves the nurse scheduling problem in 5-6 iterations, depending on the number of nurses [2]. Here below is the summary of the tests against real data:

Number of nurses	Number of iterations
20	5
68	6
58	6

Table 2.8: Summary of results [2]

In comparison to Yilmaz's model, this model performs far better in terms of iterations. To recall, Yilmaz's model found its solution in 116 iterations for 10 nurses [23]. Whilst, this model found its solution within 5-6 iterations for 20-58 nurses [2].

CHAPTER 3

Ant Colony Optimization Models

To recall, the Ant Colony Optimization algorithm mimics an ant's ability to look for food by using mini-agents that move across a graph that represents the problem being solved. This chapter will dwell further on the Ant Colony Optimization algorithm itself and how it is used to solve the nurse scheduling problem.

3.1 What Is The Ant Colony Optimization Algorithm?

The inner workings of the Ant Colony Optimization can be represented by the following pseudocode.

1. while has not met ending conditions do:
2. CreateSolutionUsingAnts()
3. UpdatePheromone()
4. OtherActions() {optional}

Figure 3.1: Psuedo Code of Ant Colony Optimization Algorithm [3]

As shown above, the ACO is an iterative algorithm. The number of iterations is controlled by the amount of time it takes the ACO to reach the ending conditions.

The ending conditions signify that a solution has been found. Depending on the problem the Ant Colony Optimization algorithm is solving, the ending conditions will vary in specification. The next paragraphs will fully describe and analyze the *CreateSolutionUsingAnts*(), *UpdatePheromone*(), *OtherActions*() methods in detail.

The *CreateSolutionUsingAnts*() creates possible solutions using mini-agents. The mini agents, thematically named ants, originally start without any paths formed. The ants choose a path using the pheromone value of that path. In addition, the mini-agents can use another factor, called heuristic information, to aid their path-making decision. The heuristic information is an optional weighting function that assigns a certain value to each path of the graph [3]. The value the heuristic information assigns has some relation to the problem the Ant Colony Optimization is applied to. For example, the heuristic information is the inverse of the distance between the edges if the ACO is applied to the traveling salesman problem.

The mini agents follow a simple probability rule: "Follow the path with the higher pheromone probability". The pheromone probability is defined as:

$$\frac{[\tau_i]^\alpha * [\eta(P_i)]^\beta}{\sum_{j \in P}[\tau_j]^\alpha [\eta(P_j)]^\beta}$$

Figure 3.2: Formula For Pheromone Probability [3]

i is the chosen path, τ_i is the pheromone value of i, $\eta(P_i)$ is the heuristic value of i and P is the set of all possible chosen paths. α and β are weighting variables that determine the relationship between the pheromone values and the heuristic information. The larger one variable is, in comparison to the other variable, the more weight that variable has on deciding the value of the pheromone probability. Once a path has been chosen, the mini-agent adds the path to its path sequence. At the beginning of each iteration, this path sequence is empty.

The *UpdatePheromone*() evaporates the pheromones using the pheromone update rule. The pheromone update rule is the guideline used to change the pheromone values of all paths in the graph. The pheromone update rule consists of two areas. The first area is the pheromone evaporation. Pheromone evaporation decreases the pheromone values of all paths in the graph by a constant factor. This factor is called the evaporation rate, often denoted by ρ. While the evaporation rate is in the range of 0 and 1 inclusive, ρ can never be 0. Pheromone evaporation is important to the Ant Colony Optimization algorithm due to the fact it allows the algorithm to forget some paths and incentivizes the algorithm to explore new paths. In addition, it prevents the algorithm from choosing a suboptimal path as its solution[3]. The second area is changing the pheromone values of a possible solution or solutions. There is no set of rules to determine which solutions are selected to be updated. The two most popular solution selection rules are the IB-update rule and the BS-update rule. The IB-update rule states that the chosen solution must be the best solution of the iteration [3]. The BS-update rule states that the chosen solution must be the best solution since the beginning of the algorithm [3]. The pheromone value of the selected solution becomes:

$$(1 - \rho) \cdot \tau_i + \rho \cdot \cdot \sum w_i \cdot F(i) \tag{3.1}$$

Figure 3.3: Formula For Pheromone Value After Evaporation [7]

i is the selected solution or solutions. ρ is the evaporation rate, τ_i is the pheromone, w_i is an optional weighting variable very similar to the heuristic information and $F(s)$ is the quality function. The quality function, $F(i)$, is a function that determines how optimal a solution is. The more optimal the solution, the higher the value of $F(i)$ is.

OtherActions() are the set of actions that can not be performed by a single mini

agent. These actions may include, local search methods or the collection of certain information to determine if pheromone should be deposited to a certain path [3].

3.2 The Models

3.2.1 Sun Yat-sen Ant Colony Optimization Model

Similar to the Preference Scheduling, the Sun Yat-sen model uses user defined parameters and the nurses' preference to determine the best solution to the nurse scheduling problem [22]. In Sun Yat-sen's paper, the user defined parameters are: pheromone weights, evaporation rate, weighting variables, node switch probability and penalty coefficient. The Sun Yat-sen model first reads in the user parameters and scheduling data. Then, the model creates the graph the ant colony optimization algorithm will be applied on. Each node on the graph represents each shift pattern. Soon after, the ant colony optimization algorithm is applied to the constructed graph. The solution to the Sun Yat-sen model is the sequences of nodes derived by the ant colony optimization algorithm. This model has the following constraints:

1. Every nurse works exactly one shift pattern
2. The demand for nurse is fulfilled for every type of nurse and time of day.

Notation

The Sun Yat-sen model uses the following notations for its decision variable and parameters [22]:

x_{ij}	1 if nurse i works shift pattern j, 0 otherwise
m	Number of possible shift patterns
n	Number of nurses
p	Number of nurses types
a_{jk}	1 if shift pattern j covers day/night k, 0 otherwise
q_{ig}	1 if nurse i is of grade g or higher, 0 otherwise
c_{ij}	Preference cost of nurse i working shift pattern j
R_{kg}	Demand of nurses with grade g on day/night k
F_i	The set of feasible shift patterns for nurse i

Quality Function

The quality function for this model is:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \lambda\left(\sum_{k=1}^{14}\sum_{g=1}^{3}\{\max\{0, R_{kg} - \sum_{i\in G_g}\sum_{j\in F_i} a_{jk}x_{ij}\}\}\right)$$

Figure 3.4: Quality Function For Sun Yat-sen Model [22]

This quality function combines a simple objective function with a penalty function.

The simple objective function is $\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$. This function measures the total preference cost of all nurses.

Penalty functions penalize the quality function if the possible solution, represented by x_{ij}, violates the model's constraint. These functions transforms constrained optimization functions to unconstrained optimization function with the introduction of a penalty term. For the Sun Yat-sen model, this penalty function is $\lambda\left(\sum_{k=1}^{14}\sum_{g=1}^{3}\{\max\{0, R_{kg} - \sum_{i\in G_g}\sum_{j\in F_i} a_{jk}x_{ij}\}\}\right)$. λ is the penalty coefficient and

$\sum_{k=1}^{14}\sum_{g=1}^{3}\{\max\{0, R_{kg} - \sum_{i\in G_g}\sum_{j\in F_i} a_{jk}x_{ij}\}\}$ is a function that measures the harshness of not fulfilling demand.

Creating The Graph

Each element of F_i, where $i = 1, 2, \cdots, n$, is a node in the constructed graph [22]. Let there be an extra node, noted by v_0, that represent the starting position of the ant colony optimization algorithm. An edge u is defined as the pair (v_i, v_{i+1}) where $i = 0$ or $v_i \in F_i; i = 1, 2, \cdots, n-1$.

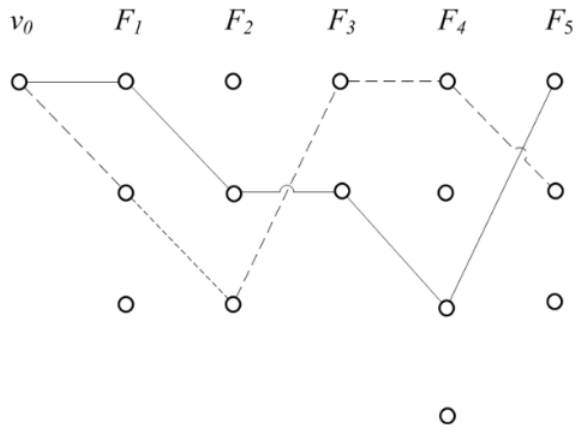

Figure 3.5: Visualization of Sun Yat-sen Model Graph [22]

Getting Our Solution

The Sun Yat-sen model uses the ant colony optimization algorithm to calculate the solution to the nurse scheduling problem. However, this model changes how the pheromones are managed and modifies how the pheromone probability is calculated.

Before the pheromone probability is calculated, the heuristic value each node is calculated. For this model, the heuristic value is $\eta_{ij}^s \cdot \eta_{ij}^d$ for each pair of nodes (v_i, v_j). η_{ij}^s is the cost of the nurse i working within schedule v_{ij}. This variable is calculated at the beginning of the ant Colony optimization algorithm. η_{ij}^s is formulated as

$(1 + c_{ij})^{-1}$. η_{ij}^{d} is the cost all the nurses, that are not nurse i, fulfilling the uncovered shifts in schedule v_{ij}. η_{ij}^{d} is formulated as $\sum_{g=1}^{3} w_g q_{ig} (\sum_{k=1}^{14} a_{jk} d_{kg})$. q_{ig} is the number representing the shortage of nurses, of grade g, during period k. w_g is a weighting parameter for the demand of higher grade nurses.

At the beginning of the ant colony optimization algorithm, the pheromone value, for all edges on the graph, is set to $(\phi_0 + 1)^{-1}$. ϕ_0 is the fitness of a solution created using the heuristic information. A local pheromone updating rule applied so that each pheromone arc, τ_{x_{i-1},x_i}, of the solution x is updated to be equal to $(1 - \epsilon)\tau_{x_{i-1},x_i} + \epsilon\tau_0$ where $\epsilon \in (0, 1)$. After all the ants creating their solution, a global pheromone rule is applied so that all the pheromones are updated to:

$$(1 - \rho)\tau + \rho(1 + \phi^{bsf})^{-1}, \quad if (v_w, v_l) \in x^{bsf} \tag{3.2}$$

$$\tau, \quad \text{otherwise} \tag{3.3}$$

Figure 3.6: Pheromone Update Rule For Sun Yat-sen Model [22]

ρ is a parameter and ϕ^{bsf} is the fitness of the best-so-far solution.

Application In The Real World

The Yat-Sen Model was tested against an integer programming model, an indirect genetic algorithm model, and a hill-climbing algorithm model [22]. When tested on 52 instances on benchmark data, the Sun Yat-sen model generally performs as well as or better than the integer models and genetic algorithms. In 47 out of 52 instances of the benchmark data, the Sun Yat-sen model performs as well as a simple integer model. The Sun Yat-sen model performs better than the hill-climbing genetic algorithm model in 51 out of 52 instances of the benchmark data. For 38

instances of the 52 instances of benchmark data, the Sun Yat-sen model performs better than indirect genetic algorithm model. In the remaining 14 instances, the Yat-sen model performs slightly better than the indirect genetic algorithm model. There are only 2 instances where the indirect genetic algorithm slightly performs better than the Yat-Sen model.

3.2.2 Hybrid Ant Colony Optimization Model

The Hybrid Ant Colony Optimization (HACO) model combines the ant colony optimization algorithm with the hill climbing technique to solve the nurse scheduling problem [16]. The hill climbing method is a local search technique used to find any better solutions near the currently best solution. Despite the differences between the ant colony optimization algorithm and the Simplex Method, this model applies a similar process of handling constraints. Within this model, the ant colony optimization algorithm is constrained by a set of hard constraints. Hard constraints are the set of constraints that must be obeyed for a solution to be considered valid. Furthermore, the solution derived through the ant colony optimization algorithm is evaluated through a set of soft constraints. Soft constraints are a set of constraints that are useful to fulfill but do not necessarily have to be obeyed. These constraints are further detailed in the "Constraints" subsection. This model was specifically developed for a Special Care Nursery ward in a Malaysian hospital [16]. Thus, the constraints of the nurses and the model are designed to suit the Malaysian Special Care Nursery ward. Similar to the integer programming models, each model can do 3 types of shifts. In Hybrid Ant Colony Optimization paper, these shift types are described to be:

Shift Type	Duration	Notation
Morning	07:00-14:00	M
Evening	14:00-21:00	E
Night	21:00 - 07:00	N

In addition, the nurses are given off days and holidays. These breaks are noted as:

Off-days	Notation
Weekly off day	WO
Public off day	PO
Annual leave	AO
Night off day	NO

Constraints

As previously mentioned, hard constraints are the set of constraints that must be obeyed for a solution to be considered valid. The hard constraints are [16]:

1. Nurses are to work six days a week with one day off each week. This is the Mandatory Working Days Constraint.

2. Each nurse must have consecutive working days between two and six. Thus, split off days or single working day are disallowed. This is the Work Stretch Constraint.

3. At least three nurses, of differing skill level, must be allocated for each shift. This is the Covering Constraint.

4. Each nurse must be assigned one shift per 24 hour-day. This is the Work Requirement Constraint.

5. *N* shifts are assigned in blocks of three shifts as directed by turns and rotations. This is the Pre-assigned Constraint.

6. There has to be two days off after the third *N* shift of the working block. This is the Pre- assigned Constraint.

7. For each nurse, a night shift can not be followed by a day shift. This is the Ordering Constraint.

8. For all the nurses: A morning shift should be followed by an evening shift. An evening shift should be followed a night shift. This is the Ordering Constraint.

Soft constraints are a set of constraints that are useful to fulfill but do not necessarily have to be obeyed. Here are the soft constraints [16]:

1. The schedule should have consecutive off days. This is the Days Off Arrangement Constraint.

2. The number of morning shifts should between two and four. The number of evening shifts should be between two and four. This is included within the Shift Arrangement Constraint.

3. Similarly, for a 5-day work stretch, the number of morning shifts should be between two and three. Furthermore, the number of evening shifts should be between two and three. This is included within the Shift Arrangement Constraint.

4. For a 4-day stretch, there should be two morning shifts and two evening shifts.

5. For 2-day and 3-day work stretches, there should be two morning shifts or two evening shifts and three morning shifts or three evening shifts respectively.

Each soft constraint has a value attached to it that represents the severity of violating that constraint. This value is called the penalty value.

Quality Function

The quality function of this model uses the following sets [16]:

1. Z - Total value of penalties

2. T - Total number of days on shift timetable

3. J - Total number of nurses in the ward

4. I - Day index

5. P - Shift arrangement

6. S - Stretch of working day

7. O - Shift Type

In addition, the quality function uses the following parameters [16]:

1. ϵ_{jo} is the penalty value for nurses j=1,. . . ,39 of nurse type o

2. ρ_{jp} is the penalty value for nurse j for shift arrangement p

3. φ_{js} is the penalty value for nurse j for working stretch s

4. ψ_{jl} is the penalty value for nurse j for non-working stretch l

Furthermore, the function uses the following decision variable [16]:

1. A_{ijo} is the assignment of nurse type o (0 = not assigned or 1 = assigned)

2. C_{ptj} is the number of shift arrangement for nurse j

3. D_{stj} is the number of working stretch for nurse j

4. E_{jv} is the number of non-working stretch v

The quality function for the Hybrid Ant Colony Optimization model is:

$$\text{Min} \quad Z = \sum_{i=1}^{I} \sum_{j=1}^{J} A_{ijo}\epsilon_{jo} + \sum_{j=1}^{J} \rho_{jp}C_{ptjd} + \sum_{j=1}^{J} \varphi_{js}D_{stjd} + \sum_{j=1}^{J} \Psi_{jl}E_{jv} \tag{3.4}$$

Figure 3.7: Quality Function For The Hybrid Ant Colony Optimization [16]

The quality function returns the smallest combinations of penalties.

Deriving A Solution

The model derives a solution by [16]:

1. Gathering the number of staff in the roster, the length of roster and the number of public off days. These parameters are user inputted.
2. The evaporation value of pheromone (noted by ρ), number of one group of ants (m), the weights of the pheromone and heuristic information α and β respectively, and the number of iterations are set.
3. The initial set of schedules are created.
4. Hill climbing is used to improve the initial schedule.
5. The ants move across the graph as instructed through the ant colony optimization algorithm.
6. Once the ants have completed their tour, all pheromone values are updated.
7. Update currently the best solution. Then, evaluate whether the stop condition has been reached.

Application In The Real World

When the Hybrid Ant Colony Optimization Model was applied to a hospital in Malaysia, the model performed better than the memetic model and can satisfy most of the nurses' personal preferences [16]. Referring to the Hybrid Ant Colony Optimization Model paper, it takes about 1 minute to create a schedule for the entire Special Nursery Ward using the hybrid ant colony model. In comparison, it takes 12 minutes to create a schedule for the entire Special Nursery Ward using the memetic model.

CHAPTER 4

Genetic Algorithm Models

4.1 What Are Genetic Algorithms?

To recall, genetic algorithms are a set of algorithms that imitates the evolutionary process and the idea of the survival of the fittest aspects of evolution [1]. Genetic algorithms have existed since the 1960s. Rather than solving a specific problem, genetic algorithms are designed to study how solutions are formed. Genetic algorithms were originally invented by John Holland [17]. They were further developed by Holland's colleagues and students. Holland's version of genetic algorithms described a method where populations of "chromosomes" would transition through several forms by a process similar to natural selection [17]. John Holland combined this method with the genetic operations of crossover, mutation and selection.

4.1.1 Definitions

These definitions come from the book *Introduction to Genetic Algorithms* by S. N. Sivanandam [17].

- Chromosomes represent possible solutions to the currently solved problems. The chromosomes are often bit strings. Each chromosome contain genes.

- Genes, within the context of genetic algorithm, is a single value or set of values of the language used to describe the chromosome. Genes encode a unique aspect of the solution. Within the context of bit string chromosomes, the genes will be a singular bit or a set of bits.

- Alleles are all possible values of genes. In the context of bit string chromosomes, the values of alleles are 0 or 1. Within each gene was an allele with the value of 0 or 1.

- The crossover operator switches genetic material between two chromosome parents. This operator begins by choosing a random gene, called the locus, within the chromosome parents. Once selected the crossover operator exchanges the genes before and after the locus to create two new offsprings. These offsprings will have aspects of both parents. The chromosome parents often have one chromosome. This operation has a probability p_c of occurring between two selected chromosomes. p_c is called the crossover rate.

- The mutation operation modifies the allele of a random gene within the selected chromosome. The probability that a random gene is mutated is p_m. p_m is called the mutation rate.

- The selection operation chooses the chromosomes that are allowed to reproduce. During the selection operation, a fitness function is used to determine which chromosomes are allowed to reproduce. Each chromosome has a probability p_s of being selected. The higher the fitness value of a particular chromosome the larger the value of p_s is. p_s is called the selection rate.

- The fitness function is a function that evaluates how well the chromosome solves the current problem. Fitness functions produce a value that signifies

the effectiveness of the tested chromosome. This value is called the fitness value.

4.1.2 Rubric For Genetic Algorithms

Borrowing from *Introduction to Genetic Algorithms*, here below is a simple rubric that most genetic algorithms follow:

1. First, the genetic algorithm begins to create a set of n chromosomes. The n chromosomes are randomly generated.
2. The following steps are repeated until a stopping critea is reached:
3. The fitness value for the entire chromosome population is determined.
4. Repeat until there are n new offsprings:
5. Select a pair of parent chromosomes.
6. With the crossover rate of p_c, apply the crossover operation to form two offsprings. If there is no crossover, create two new offsprings that are exactly copies of their parents.
7. With the mutation rate of p_m, apply the mutation operator to the new offsprings. Place the newly mutated offsprings into the new population.
8. If the total number of new offsprings is odd, randomly remove one of the newly created chromosomes.
9. Replace the old chromosomes with the newly created offsprings.
10. Go to step 3

Each iteration of step 4 is called a generation. Each iteration of step 2 is called a run.

4.2 The Models

4.2.1 Cooperative Genetic Algorithm Model

Similar to the Yilmaz's model, the Cooperative Genetic Algorithm Model has three shifts types [8]. Referring the Cooperative Genetic Algorithm Model paper, these shift types are defined as:

Shift Type	Notation
d	day shift (8:00 - 16:00)
n	night shift (16:00 - 00:00)
l	late-night shift (0:00 - 08:00)

A nurse's day off are denoted by *h*. Each day off and shift is a byte within the bit string chromosome [8].

Hard and Soft Constraints

Within this model, the maximum number of nurses for of nurses is denoted by *MAXN*. The maximum number of days is denoted by *MAXD*. When applying this model within a simulation, *MAXN* is 15 and *MAXD* is 30 [8]. Referencing the Cooperative Genetic Algorithm Model paper, the hard constraints of this model are:

1. Each nurse can work only one shift in a day
2. The number of day shift nurses must be larger than the required number of nurses for day shifts
3. The number of night shift nurses must be equal to the required number of nurses for night shifts
4. The number of late shift nurses must be equal to the required number of nurses for late shifts

The soft constraints are the individual schedule preferences of each nurse.

Fitness Function

The fitness function of this model, denoted by F_i for nurse i, is compromised of three components [8]. Using the Cooperative Genetic Algorithm Model paper as a reference, these components are:

Notation	Description
F_{ip}	Fitness of the night shift pattern in respect to its order and length
F_{ic}	The number of consecutive night shifts in respect to its length
F_{id}	The interval between night shifts

Each component of the fitness function quantifies the soft constraints. If the soft constraints are violated, then the corresponding fitness factor is penalized.

F_i is formally defined as:

$$F_i = \alpha F_{ip} + \beta F_{ic} + \gamma F_{id} \tag{4.1}$$

Figure 4.1: Fitness Function For Cooperative Genetic Algorithm Model [8]

α, β, γ are weights that quantify the specific critea for the hospital. These variables are user specified parameters.

Objective Function

The objective function for the entire genome population is:

$$\text{Maximize} \quad avg = \frac{1}{N} \sum_{i=1}^{MAXN} F_i \tag{4.2}$$

$$\text{Maximize} \quad dev = \sqrt{\frac{1}{MAXN} \sum_{i=1}^{MAXN} F_i^2 - \frac{1}{MAXN^2} \left(\sum_{i=1}^{MAXN} F_i \right)^2} \tag{4.3}$$

$$\textbf{Subject To} \quad \text{The Hard Constraints} \tag{4.4}$$

Figure 4.2: Objective Function For Cooperative Genetic Algorithm Model [8]

The most desirable solution is the solution where $avg = 0$, $dev = 0$ and $F_i = 0$. Thus, the closer avg, dev and F_i are to 0 the more desirable the solution is.

Finding A Solution

Referring to the Cooperative Genetic Algorithm Model paper, the Cooperative Genetic Algorithm Model finds a solution to the nurse scheduling problem as follows:

1. The working history of the last 15 days and the weight parameters are submitted.

2. The initial feasible schedule is created. When creating the initial feasible schedule, each chromosome in the population represents a nurse's schedule. Each allele is either a shift type or a day off.

3. To create a schedule, two chromosomes are randomly selected. Two position within their bit strings are selected. The crossover operator is applied to produce two new child bit strings.

4. The average fitness and standard deviation of the population, with the new children, is calculated. If the average fitness of the newer population is larger

than the average fitness of the older population and the standard deviation of the newer population's fitness is less than the older population's fitness, then the new solutions will not be removed.

5. We repeat Step 3 and 4 until one generation has been created.

6. Steps 3 to 5 is repeated until each the nurse's fitness is 0 or the time spent iterating exceeds *MAXTIME*.

Application In The Real World

The Cooperative Genetic Algorithm Model is not tested against real world data. However, the model is tested in a simulated environment. With the parameters of:

1. *MAXD* is 30
2. The working history of the previous 15 days
3. *MAXN* is 15
4. *MAXTIME* is 15 minutes
5. $\alpha = \beta = \gamma = 1.0$

With these parameters, the model derived a solution in 10 trials [8].

4.2.2 Indirect Genetic Algorithm Model

The Indirect Genetic Algorithm Model solves the nurse scheduling problem through the use of a two-step process [1]. The first step is using the genetic algorithm to create a set of lists. Each item in these lists are nurses. The chromosomes, within the Indirect Genetic Algorithm Model, are a list of nurses whose schedules need to be created. The second step is passing that ordering of nurses to a decoder. Within the

genetic algorithm space, decoders translate chromosomes into a different format. Whilst translating, decoders can still modify the selected chromosome. There are no specific guidelines on the limitations of decoders. Referring to the Indirect Genetic Algorithm paper, all genetic decoders follow the following rubric:

1. Each decoded solution must have a corresponding chromosome solution.
2. Each chromosome solution must have a corresponding decoded solution.
3. All decoded solutions should be represented by the same number of chromosome solutions.
4. The transformation between chromosome solution and decoded solution should be fast.
5. Small changes in the chromosome solution should result in small changes in the decoded solution.

The Indirect Genetic Algorithm Model violates rubric rule 1 & 3. These specific rubrics are violated so that the decoder does not tend to lower quality solutions [1]. The genetic algorithm does not directly influence what solution will be chosen. It indirectly solves the nurse scheduling problem by providing the foundation on which the solutions are created. The decoder chooses the schedule with the highest score [1].

Notations

Referencing the Indirect Genetic Algorithm Model paper, this model uses the following variable notations:

- $d_{ks} = 1$ if there are still nurses needed on day k of grade s otherwise $d_{ks} = 0$
- $a_{jk} = 1$ if shift pattern j covers day k otherwise $a_{jk} = 0$

- w_s is the weight of covering an uncovered shift of grade s
- w_p is the weight of the nurse's p_{ij} value for the shift pattern
- p_{ij} is the preference cost for nurse i using schedule j
- w_{demand} is the demand for nurses.

The Decoder

For each nurse, the decoder first iterates over each possible schedule. It assigns each schedule with a specific score. This score is determined by the equation below:

$$w_p(100 - p_{ij}) + \sum_{s=1}^{3} w_s q_{is} \left(\sum_{k=1}^{14} a_{jk} d_{ks} \right) \tag{4.5}$$

Figure 4.3: Decoder Scoring Formula For The Indirect Genetic Algorithm Model [1]

This equation measures how well the schedule fulfills the nurse's preferences and allocates nursing resources to uncovered shifts [1].

Fitness Function

The Indirect Genetic Algorithm Model uses the following equation as its fitness function:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ij} + w_{demand} \sum_{k=1}^{14} \sum_{s=1}^{p} max \left[R_{ks} - \sum_{i=1}^{n} \sum_{j=1}^{m} q_{is} a_{jk} x_{ij}; 0 \right] \tag{4.6}$$

Figure 4.4: Fitness Function For The Indirect Genetic Algorithm Model [1]

The fitness function is used to determine the fitness of the decoded solutions.

Crossover Operators

The crossover operator for the Indirect Genetic Algorithm model acts similarly to the crossover operator explained in 4.1. However, the Indirect Genetic Algorithm model modifies the crossover operator. Referring to the Indirect Genetic Algorithm's academic paper, it first adds a binary layer mask. Each position in the binary layer mask represents a position in the parent chromosomes. If the i^{th} position in the binary layer mask is 1, then the first child inherits the i^{th} bit of the first parent whilst the second child inherits the i^{th} bit from the second parent. The operator acts contrary to this rule when the i^{th} value of the binary layer mask is 0. If the i^{th} position in the binary layer mask is 0, then the first child inherits the i^{th} bit of the second parent whilst the second child inherits the i^{th} bit from the first parent. In this crossover rate, the selection rate is the probability of a position in the layer mask has the value of 1 [1].

Application In The Real Word

Referring to Uwe and Dowsland's paper, the Indirect Genetic Algorithm model was applied to 52 sets of real hospital data. Each set of data contained the requirements of all shifts and nurse type combinations for one week, the list of nurses that are available for that week, and the preference cost and qualifications for each nurse [1]. The data was gathered from three wards over a period of several months [1]. Thus, the gathered data contains a variety of hospital situations. The Indirect Genetic Algorithm model was tested using a Pentium II PC [1]. The Pentium II PCs are very similar to the computers the average hospital used in 2004. The model found an optimal or near-optimal schedule in 51 out of the 52 data sets [1].

CHAPTER 5

Creating The Scheduler

The previous three chapters of this paper explained three groups of models for the nurse scheduling problem. Each of these groups has its own set of advantages and disadvantages.

The integer programming models, for the nurse scheduling problem, are incredibly simplistic. Furthermore, the methods used in the integer programming models derive an exact solution as opposed to finding an approximate solution [6]. Despite the stated advantages, integer programming models contain some disadvantages. Integer programming models are simplistic in order to reduce the complexity the solutions [4]. The simplistic nature of integer programming models reduces how applicable the model is to the real world. The solution created by these models are often too simple and specific to be applied [4].

Unlike integer programming models, genetic algorithms can be applied to a variety of use cases. Genetic algorithms generally derive their solutions within a small amount of iterations. However, the solution, generated by the genetic algorithm, can not be guaranteed to be the best solution possible.

Similar to genetic algorithms, the any colony optimization algorithm can be used to solve a wide array of problems. Unlike genetic algorithms, the ant colony optimization algorithm derives a solution that is globally optimal. However, this

solution, created by the ant colony optimization algorithm, requires many iterations to form.

It would be advantageous to combine the advantages of the ant colony optimization algorithm models, genetic algorithm models and integer programming models. Despite the advantageous nature of combining these models, the integer programming model can not be combined with the other models due to its use of the Simplex Method. While the features of the integer programming models can not be used, the advantages of these models can be gained. Using a similar structure to the Hybrid Ant Colony Optimization model, the genetic algorithm can be combined with the ant colony optimization algorithm. The combination of the genetic algorithm and the ant colony optimization algorithm is called the Genetic Ant Colony Optimization Algorithm (GACO) [24].

5.1 Our Model

5.1.1 Genetic Ant Colony Optimization Algorithm (GACO)

To recall, the genetic ant colony optimization algorithm combines the genetic algorithm and the ant colony optimization algorithm. Citing Wei-guo and Lu's Genetic Ant Colony Algorithm paper, the Genetic Ant Colony Optimization Algorithm is as follows [24]:

1. The genetic algorithm is used to generate a set of initial solutions.
2. Whilst the genetic algorithm iterates through its generations, the genetic algorithm checks if the newest generation has increased its fitness.
3. If the average fitness of the current generation divided by the average fitness of the previous generation is less than or equal to 3%, then the genetic algorithm is stopped.

4. The set of solutions generated by the genetic algorithm is passed into the ant colony optimization algorithm.

5. The ant colony optimization algorithm creates the ants with a bias to the initial solutions generated by the genetic algorithm.

6. Using the ant colony optimization algorithm, a globally optimum solution is found.

5.1.2 Definitions

The model used in Backlog Burner has the following notations:

- Within this model, the user's schedule is divided into n time slices. These time slices are 5 minutes in length.
- Let t_i represents the ith time slice. The value of t_i is 1 if it belongs to an event. The value of t_i is 0 otherwise.
- Let the collection of sequential time chunks be called an event. Let event e be defined as: $e_{a,b} = \{t_a, t_{a+1}, \cdots, t_b | a < b\}$. $|e_{a,b}| = 5(b - a)$.
- Let $f_{i,j}$ represent a set of non-allocated sequential time chunks between two events. $|f_{i,j}| = 5(j - i)$.
- Let F represent the set of all non-allocated chunks.
- Let FT contain the cardinality of all elements of F.

5.1.3 Objective Function

For the automated scheduling software, the genetic ant colony optimization algorithm is used with the following objective function:

$$\text{Maximize} \quad \inf FT \tag{5.1}$$

Figure 5.1: Objective Function For Backlog Burner's Model

5.1.4 Constraints

Hard Constraints

The Backlog Burner model uses the following hard constraints:

- Let the user's schedule be noted by s and the timings of a new event be noted by t. t must not collide with any of the events in s.

5.1.5 Genetic Algorithm Portion of GACO

Chromosomes

Within this model, the chromosomes are encoded as intervals of time slices. If an event is set to start from 00:00 to 00:30, then it will be encoded as (0, 6).

Genes

The genes are the values of the chromosome's interval. Referring to the example in the chromosome section, the gene for that particular chromosome is 0 and 6.

Alleles

The alleles are the possible values of genes. Within this model, the alleles are all the non-allocated time slices.

Fitness Function

The fitness function is equivalent to the objective function descried in 5.1.

Crossover Operation

The crossover operation takes a weighted average of the parents' starting element of their chromosomes. The weight of the first parent is chosen randomly and is between 0 and 1. The weight of the second parent is the complement of the weight of the first parent. Once the new starting element has been determined, the finishing element is calculated by adding the event's length to the new calculated starting element. The new interval is tested to see if it violates any of the hard constraints of the model. If the new interval violates the hard constraints, then the crossover operation is repeated. If the new interval does not violate the hard constraints, the new child is passed into the new stages of GACO.

Mutation

The mutation operation changes the interval of the chromosome by a random factor. The new interval will be within 2 hours of the original chromosome.

Selection

The parent chromosomes, used in the crossover operation, are selected by their fitness value. A chromosome is selected if its fitness value is larger than or equal to the average fitness of the population.

5.1.6 Ant Colony Optimization Portion of GACO

Graph

The graph used in the ant colony optimization portion of GACO is the set of all possible intervals for all the new events. Each node in the graph represents a particular interval. The graph is designed in layers. Each layer i of the graph

represents all possible intervals for new event *i*. The nodes in one layer is connected to the nodes in the next layer that does not violate the hard constraints of the model. The start node is connected to all the nodes of the first layer. The end node is connected to all the nodes of the last layer.

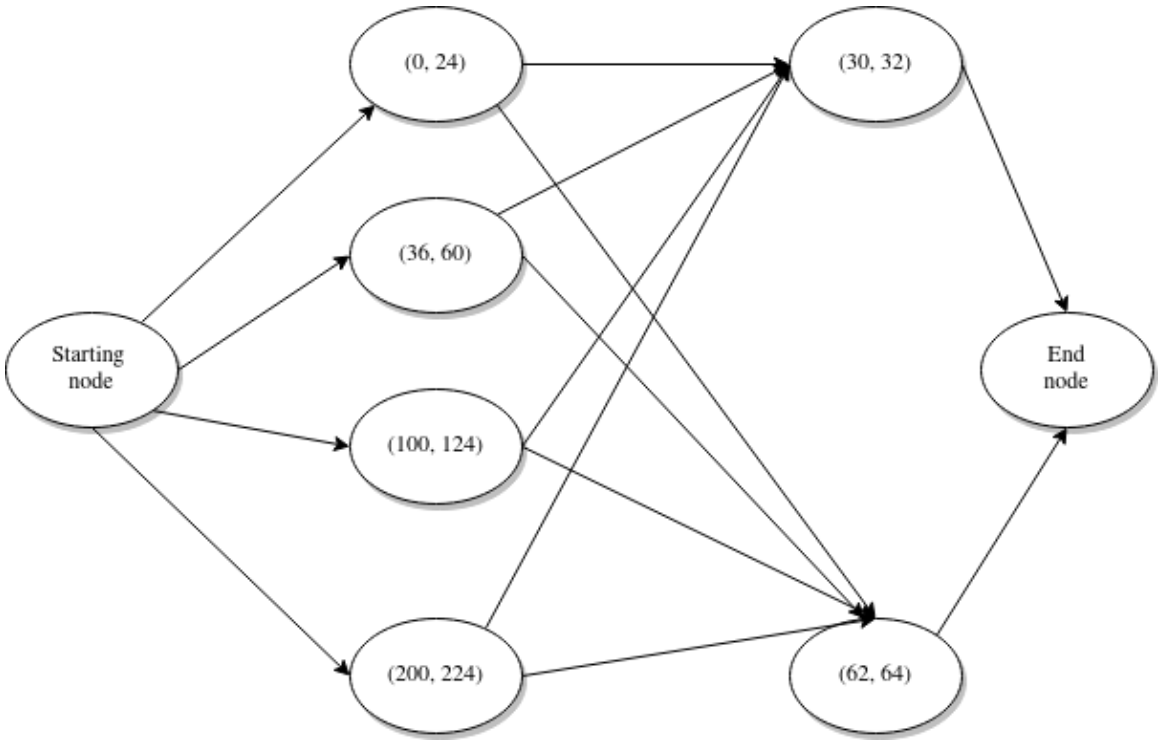

Figure 5.2: Visual representation of the ant colony optimization graph

When the GACO algorithm switches to the ant colony optimization portion, the pheromone values of the nodes used in the schedules created by the genetic algorithm are increased. This allows the ants to initially know the schedules created by the genetic algorithm.

Global Updates

The paths of the ants are the list of time intervals for the new events. These paths are evaluated by the objective function of this model. The pheromone values for the nodes of the paths with the higher objective function value are increased.

Weights

The ant colony optimization weights in this implementation of GACO are equal.

5.2 The Web App

The automated scheduling software being developed is a web app built in VueJS. VueJS is a frontend JavaScript framework designed to develop single page applications [18]. Frontend JavaScript frameworks are software designed to aid web development through the use of pre-written JavaScript code. JavaScript is a programming language commonly used in web development. Single page applications are web applications which use a single page to display all aspects of their web application [12]. The body of the single page application is updated when there is new information to display [12]. Single page applications are heavily preferred since the web server only has to send one page to the client rather than sending multiple pages to the client [12]. For this project, Vue.js was chosen due to its flexibility and ease of use.

5.2.1 Anatomy of A VueJS App

A basic VueJS app currently has these files:

- App.vue
- main.js
- HelloWorld.vue

The next three sections will discuss the contents and functions of these files.

main.js

Listing 5.1: Template main.js

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3
4 createApp(App).mount('#app')
```

All Vue applications begin with an application instance. This application instance is created using the *createApp* function [19]. Application instances are used to tie global variables [19]. This connection of global variables to VueJS applications allows the global variables to be used within VueJS applications [18]. The *createApp* function is a function created within VueJS [19]. Thus, the first line of Listing 5.1 imports the function from VueJS. Then, the fourth line, specifically the *createApp*(*App*) section, creates the application instance. The option passed into the createApp function sets the root component.

VueJS components are reusable pieces of JavaScript code [20]. The root component is the first component that is shown on the website [20]. Application instances are also used to mount root components. In Listing 5.1, the VueJS mounts the root component App.vue [20]. The second line of the code snippet imports the root component from the App.vue file. The fourth line of the code snippet, specifically the *.mount*('*#app*') section, mounts the root component.

App.vue

Listing 5.2: Template App.vue

```
1  <template>
2    <img alt="Vue logo" src="./assets/logo.png">
3    <HelloWorld msg="Welcome to Your Vue.js App"/>
4  </template>
5
6  <script>
7  import HelloWorld from './components/HelloWorld.vue'
8
9  export default {
10   name: 'App',
11   components: {
```

```
12      HelloWorld
13    }
14  }
15  </script>
16
17  <style>
18  #app {
19    font-family: Avenir, Helvetica, Arial, sans-serif;
20    -webkit-font-smoothing: antialiased;
21    -moz-osx-font-smoothing: grayscale;
22    text-align: center;
23    color: #2c3e50;
24    margin-top: 60px;
25  }
26  </style>
```

App.vue is the root component of the VueJS application. VueJS components are structured with a template section, script section and style section.

Template section Written in HTML, the template section provides a blueprint on how the component appears in the web application. In addition to HTML, VueJs provides additional tags to integrate VueJS features with standard HTML. In Listing 5.2, lines 1-4 provide a sample template section. Line 2 shows the VueJS logo that appears on the top of the page. Line 3 inserts the HelloWorld component. The HelloWorld component will be further discussed in the following subsection.

Script section The script section details how the component functions. In Listing 5.2 line 7, the HelloWorld component is imported into the root component. Lines 9-14 define the root component. Line 10 sets the name of the root component. Line 11 lists any components that are used in the root component. Currently, the App component does not have extra JavaScript functions. If it contained extra JavaScript script functions it would be contained in the methods list. The methods list is declared by:

Listing 5.3: Methods list for App.vue

```
1   export default {
```

```
2    name: 'App',
3    components: {
4      HelloWorld
5    },
6    methods: {
7    //Insert Functions Here
8      GetName(){
9        return "John"
10     }
11   }
12 }
```

In addition to functions, variables can also be attached to VueJS components. The attached variables are declared in the data list. The data list is declared by:

Listing 5.4: Data list for App.vue

```
1  export default {
2    name: 'App',
3    components: {
4      HelloWorld
5    },
6    data: {
7    //Insert Variables Here
8    username: "Person McName"
9    }
10 }
```

Style section The style section informs VueJS how the component will visually look. The style section uses CSS (Cascading Style Sheets) [13]. CSS is the language used to customize the look of web pages [13]. In Listing 5.2, Line 18 attaches the configurations, determined in line 19 to line 24, to the #app tag. Line 19 determines what font the component will use. The text are aligned to the center in line 22. In line 23, the color of the text displayed is set to a shade of gray. Line 24 adds a 60 pixel margin to the top. Given that App.vue is the root component, this style section will be applied to all components within the VueJS application.

HelloWorld.vue

Listing 5.5: Script section for HelloWorld.vue

```
1 <script>
2 export default {
3   name: 'HelloWorld',
4   props: {
5     msg: String
6   }
7 }
8 </script>
```

In the HelloWorld component, the script section is structured similarly to the root component. The only main difference is the props variable that is declared in line 4 [20]. Props are special attributes that you can attach to components [20]. The value of a prop variable can be set in the same line where the component is declared. For HelloWorld.vue, we only have one prop variable named *msg*.

Listing 5.6: Template section of the root component

```
1 <template>
2   <img alt="Vue logo" src="./assets/logo.png">
3   <HelloWorld msg="Welcome to Your Vue.js App"/>
4 </template>
```

Referring to the template of the root component, the value of *msg* is set to be "Welcome to Your Vue.js App".

Listing 5.7: Template section for HelloWorld.vue

```
1  <template>
2    <div class="hello">
3      <h1>{{ msg }}</h1>
4      <p>
5        For a guide and recipes on how to configure / customize
             this project,<br>
6        check out the
7        <a href="https://cli.vuejs.org" target="_blank" rel="
             noopener">vue-cli documentation</a>.
8      </p>
9      <h3>Installed CLI Plugins</h3>
10     <ul>
11       <li><a href="https://github.com/vuejs/vue-cli/tree/dev/
             packages/%40vue/cli-plugin-babel" target="_blank" rel=
             "noopener">babel</a></li>
```

```
12        <li><a href="https://github.com/vuejs/vue-cli/tree/dev/
              packages/%40vue/cli-plugin-eslint" target="_blank" rel
              ="noopener">eslint</a></li>
13      </ul>
14      <h3>Essential Links</h3>
15      <ul>
16        <li><a href="https://vuejs.org" target="_blank" rel="
              noopener">Core Docs</a></li>
17        <li><a href="https://forum.vuejs.org" target="_blank" rel
              ="noopener">Forum</a></li>
18        <li><a href="https://chat.vuejs.org" target="_blank" rel=
              "noopener">Community Chat</a></li>
19        <li><a href="https://twitter.com/vuejs" target="_blank"
              rel="noopener">Twitter</a></li>
20        <li><a href="https://news.vuejs.org" target="_blank" rel=
              "noopener">News</a></li>
21      </ul>
22      <h3>Ecosystem</h3>
23      <ul>
24        <li><a href="https://router.vuejs.org" target="_blank"
              rel="noopener">vue-router</a></li>
25        <li><a href="https://vuex.vuejs.org" target="_blank" rel=
              "noopener">vuex</a></li>
26        <li><a href="https://github.com/vuejs/vue-devtools#vue-
              devtools" target="_blank" rel="noopener">vue-devtools<
              /a></li>
27        <li><a href="https://vue-loader.vuejs.org" target="_blank
              " rel="noopener">vue-loader</a></li>
28        <li><a href="https://github.com/vuejs/awesome-vue" target
              ="_blank" rel="noopener">awesome-vue</a></li>
29      </ul>
30    </div>
31  </template>
```

This template is similar to normal HTML except for line 3. Line 3 dynamically inserts the value of *msg* as a header of size 1.

Listing 5.8: Style section for HelloWorld.vue

```
1  <style scoped>
2  h3 {
3    margin: 40px 0 0;
4  }
5  ul {
6    list-style-type: none;
7    padding: 0;
8  }
9  li {
10   display: inline-block;
11   margin: 0 10px;
12 }
```

```
13 a {
14   color: #42b983;
15 }
16 </style>
```

The style section for HelloWorld is scoped [18]. Thus, the configurations in this section is only applied to the HelloWorld component and not other components.

5.2.2 Web App Design

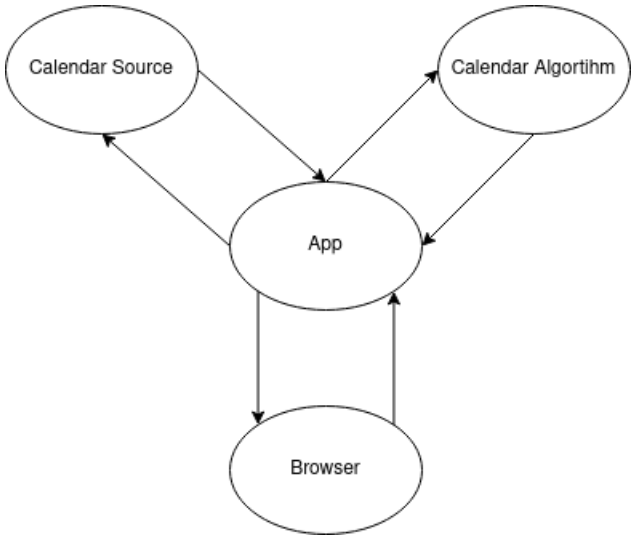

Figure 5.3: Visual Representation of Backlog Burner's Structure

Referring to Figure 5.2, the structure of the automated scheduler contains 3 areas: the app, the calendar source and the calendar algorithm.

The App

The App area of the diagram represents the VueJS area of Backlog Burner. This area directly interacts with the user. The primary function of this area to provide a simple interface for the user that can interact with the GACO algorithm and their calendar. To use the application, the user must:

1. Login with their Google account or their Outlook account

2. Add their list of hobbies

3. Click the submit button

4. Once the new schedule is received, the user must click the finalize button

Step 1 and steps 2-4 are encapsulated within two pages: the Login page and the Schedule page.

The app section achieves its goal through the use of a web application programming interface (API). A web API is a web program built to run complex code without having the user to implement the complex code [14]. The GACO algorithm is encased in a web API. Web APIs have a link where a web application can access the web API. This link is called an endpoint. To use a web API, an HTTP Request is sent to the specific endpoint of the web API. HTTP Requests are web requests for a certain action to be applied to a web resource [15]. These requests are often sent to web servers and APIs [15]. There are several types of HTTP requests [15]. This project only uses two categories of HTTP requests: GET and POST. GET requests ask for data from the web server or APIs [15]. POST requests send data to the web server or APIs [15].

The VueJS application is programmed differently from the VueJS application described in 5.2.2.

MAIN.JS The main.js file for Backlog Burner is:

Listing 5.9: The main.js for Backlog Burner

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import router from './router'
4 import GAuth from 'vue3-google-oauth2'
5 import VueCookies from 'vue3-cookies'
6 import * as msal from '@azure/msal-browser'
7
8 const app = createApp(App);
9
```

```
10 var msalConfigTemp = {
11     auth: {
12         clientId: "CLIENT ID",
13         authority: "https://login.microsoftonline.com/common",
14         redirectUri: process.env.VUE_APP_REDIRECT_URL,
15         postLogoutRedirectUri: process.env.VUE_APP_REDIRECT_URL
               ,
16         mainWindowRedirectUri: "localhost:8080/login"
17     }
18 }
19
20 const gAuthOptions = { clientId: 'CLIENTID', scope: 'profile
       https://www.googleapis.com/auth/calendar https://www.
       googleapis.com/auth/calendar.events', prompt: 'consent',
       fetch_basic_profile: false}
21 app.config.globalProperties.$msalClient = new msal.
       PublicClientApplication(msalConfigTemp);
22 app
23 .use(router)
24 .use(GAuth, gAuthOptions)
25 .use(VueCookies)
26 .mount('#app')
```

In comparison to 5.2.2, the main.js file has massively expanded. Lines 4 - 6 import *router*, *GAuth*, *VueCookies* and *msal*. *router* is a VueJS object that helps link web pages between a VueJS application. *GAuth* is a JavaScript library that allows our VueJS application to interact with Google Authentication. Similarly, *msal* is a JavaScript library that allows our VueJS application to interact with Microsoft's authentication system. The details of these interactions will be further discussed in the *Calendar Source* section. *VueCookies* is a JavaScript library that allows VueJS to store information within a user's cookies.

Lines 10-18 describe the configurations needed to interact with Microsoft's authentication system. This configuration is wrapped within an auth object. This auth object has a clientId, authority, redirectUri, postLogoutRedirectUri, and mainWindowRedirectUri variables. The clientId is the ID given to Backlog Burner when registering the application with Microsoft's Azure service. The authority variable tells Microsoft what type of user is asking for access. Referring to the

Microsoft authorization documentation, there are four general values for the authority variable:

- https://login.microsoftonline.com/common/ - This authority indicates that the user is either using a work/school account or personal account
- https://login.microsoftonline.com/organizations/ - This authority indicates that the user is solely using their work or school account
- https://login.microsoftonline.com/consumers/ - This authority indicates that the user is solely using their personal account
- https://login.microsoftonline.com/<tenant>/ - This authority indicates that the user is using their account of a specific organization. This organization is indicated by the tenant ID. This tenant ID replaces <tenant> in the url listed

In a similar function to Lines 10-18, Line 20 declares all the necessary information needed to authenticate a user with Google. To authenticate with Google, the application needs to provide the clientId, the resources requested and the type of sign in needed. The clientId is inserted in the clientId variable in gAuthOptions. The resources requested is stated in the scope variable. This application requests access to the profile information and the calendar data. The type of sign in is stated in the prompt variable. The *consent* value indicates that a pop-up window will appear that ask the user's consent to provide Backlog Burner with the user's data.

Line 21 creates a global msal object using the configuration set in lines 10 to 18. Msal object handle authenticating with Microsoft without forcing the user to deal with the intricacies of web authentication.

Lines 22-26 mount the App component to the application whilst setting the *router*, *GAuth* and *VueCookies* as globally accessible packages.

App.Vue Secondly, the App.Vue is less verbose. The HelloWorld component is not inserted. Instead, the Header component is inserted along with the router view component. The Header component is a simple component that displays the title "Backlog Burner" and the logout button. The router view tag shows the accompanying page to the specific link clicked.

Listing 5.10: App.Vue for Backlog Burner

```
<template>
  <div class="container">
    <Header />
    <router-view></router-view>
  </div>
</template>

<script>
import Header from './components/Header.vue'
import './assets/css/main.css'
export default {
  name: 'App',
  components: {
    Header
  },
  data() {
    return{

    }
  },
  methods: {

  },
  created() {
  },
  provide:{

  },
  globals:{
  }
}
</script>
```

Pages In VueJS, pages are created with views. Views, within VueJS, acts similarly to VueJS components but are intended to be used to display information to the screen. This project has three VueJS views: LoginView, ScheduleView, LogoutView.

LoginView handles logging and authenticating the user. Before LoginView is shown to the screen, the VueJS application checks to see if the user has been previously logged in. If the user has logged in before, the VueJS transitions to the Schedule View. Else, LoginView is shown to the user.

LoginView's template section is:

Listing 5.11: LoginView's template section

```
1 <template>
2 <h3>Login</h3>
3 <button @click="LoginMicrosoft">Login With Your Work/School
      Account</button>
4 <button @click="LoginGoogle" :disabled="!Vue3GoogleOauth.isInit
      || this.$cookies.isKey('accessToken')">Login With Your
      Gmail</button>
5 </template>
```

The LoginView contains two buttons for logging in the user. The *disabled* area of line 4 disables the Google login button if the user has already logged in or the *GAuth* object is not initialized yet. Currently, Backlog Burner only accepts users with a Google account or an Outlook account. Microsoft and Google use the bearer token authentication system. The bearer token system uses the following steps:

1. The web application sends an HTTP GET request for a request token [10]. This HTTP GET request includes the web application's ID, the user's log in information and any extra destination specific information.

2. Once the HTTP GET request has been received and processed, an authentication token is produced and sent to the web application [10]. This authentication token allows the web application to receive the user's data that is stored at the authenticator [10].

In Backlog Burner, the bearer token authentication is handled through the $msalClient and $GAuth global variables that were created in main.js.

The LogoutView handles logging out the user. A POST request is sent to the authenticator's logout endpoint. Once the POST request is received and processed, the web application removes any stored login information and transitions to the LoginView.

The ScheduleView handles displaying the user's current schedule and showing their new schedules. Before LoginView is shown to the screen, the VueJS application checks to see if the user has been previously logged in. If the user is not logged in, the web application redirects to LoginView. When the page has loaded, the web application sends a GET Request to web server of the account's calendar. If the user logged in with their Google account, then the web application sends a GET request to Google asking for the user's calendar data. The same process is applied for those who log in with their Microsoft accounts.

New Components Secondly, the HelloWorld component is removed. Instead of using the HelloWorld component, the AddCalendarEvent component is added. Attached to the ScheduleView, the AddCalendarEvent component handles modifying the calendar shown on the Schedule screen through the use of two pop up forms. AddCalendarEvent's template section is split into two divisions: the Add Event pop up form and the Add Hobby pop up form. The Add Event pop up form is used to add any missing hard constraints. The Add Hobby pop up form is used to the user's hobby into the VueJS application. In AddCalendarEvent's template, the Add Event form is declared as:

Listing 5.12: The Add Event pop form

```
1 <form @submit="onSubmit" class="add-form">
2   <div class="form-control">
3       <label>Name: </label>
4       <input type="text" v-model="nameOfMedia" name="name"
            placeholder="Enter name" />
5   </div>
6
```

```
    <vue-cal
    class="vuecal--date-picker"
    xsmall
    hide-view-selector
    :time="false"
    :transitions="false"
    active-view="month"
    :disable-views="['week']"
    @cell-focus="startDate = $event"
    >
    </vue-cal>

    <vue-cal
    class="vuecal--date-picker"
    xsmall
    hide-view-selector
    :time="false"
    :transitions="false"
    active-view="month"
    :disable-views="['week']"
    @cell-focus="endDate = $event"
    >
    </vue-cal>
    <input type="submit" value="Sumbit Task" />
</form>
```

The vue-cal tag creates a calendar in the form. The calendar serves to allow the user to pick the start and ends date of the task. AddCalendarEvent stores the values of the information entered into the form and adds then to the calendar once "Submit Task" is pressed. In AddCalendarEvent's template, the Add Hobby form is declared as:

Listing 5.13: Add Hobby Form

```
<form @submit="pushSC" class="add-form">
    <div class="form-control">
        <label>Name: </label>
        <input type="text" v-model="eventName" name="name"
            placeholder="Enter name" />
    </div>

    <div>
        <label>Length of Event: </label>
        <input type="number" v-model="lengthOfSC" name="
            lengthOfSC" />
    </div>

    <vue-cal
```

```
13          class="vuecal--date-picker"
14          xsmall
15          hide-view-selector
16          :time="false"
17          :transitions="false"
18          active-view="month"
19          :disable-views="['week', 'year', 'day']"
20          @cell-focus="selectedSCDate = $event"
21          >
22      </vue-cal>
23
24      <input type="submit" value="Sumbit Hobby" />
25  </form>
```

Similar to the Add Task form, AddCalendarEvent stores the values of the information entered into the form and adds them to the list of hobbies once "Submit Hobby" is pressed.

Using The GACO Algorithm

The specific details on how the GACO algorithm is implemented are discussed in Section 5.3. The Genetic Ant Colony Optimization algorithm is encapsulated by an HTTP API. This allows the user side of the software to use the GACO algorithm without being involved with the implementation of GACO. The VueJS application similarly sends a POST request to the HTTP API. This POST request contains a slice of the user's current schedule and some information about the tasks the user has submitted. Once this POST request is received and processed, the API returns a list of timings for the tasks the user has submitted.

5.3 Implementing GACO

As previously stated, the GACO algorithm is encapsulated within an HTTP API. This HTTP API and the implementation of GACO are written in the Rust programming language. This section is divided into five subsections. The first subsection explains the process of creating an HTTP API in Rust. The last four subsections are devoted to

describing how the GACO algorithm is implemented in Rust. Rust is a programming language designed for performance and memory safety [9].

5.3.1 Web API

This project uses Actix Web to handle HTTP requests and responses. Actix Web is a Rust web framework [21]. To attach Actix Web to a Rust project, Actix Web must be installed first. For the sake of brevity, this paper will not go through the process of installing external libraries. Given that Actix Web is installed, an Actix Web API is declared with: [21]:

```
1  use actix_web::{App, HttpServer};
2
3  #[actix_web::main]
4  async fn main() -> std::io::Result<()> {
5      HttpServer::new(|| {
6          App::new()
7      })
8      .bind("127.0.0.1:8080")?
9      .run()
10     .await
11 }
```

Lines 5-7 creates a HttpServer object. The HttpServer object represents an HTTP Server as a Rust struct. Line 8 tells Actix Web the desired IP address and port of the web API. Line 9 activates the Web Server. The web API is hosted at 127.0.0.1:8080. However, this web API can not be used. To add a Rust function to the web API, the Actix Web service function is used. Let us assume we want to add a function named *echo* that returns any value that was sent to the web API. *echo* is declared by [21]:

```
1  use actix_web::{post, HttpResponse, Responder};
2  1
3  #[post("/echo")]
4  async fn echo(req_body: String) -> impl Responder {
5      HttpResponse::Ok().body(req_body)
6  }
```

Line 3 binds the echo function to the "/echo" endpoint. Web endpoints are strings

used to identify specific functions within an API. These strings are attached to the end of the API's URL. Thus, information must be sent towards "127.0.0.1:8080/echo" to activate *echo*. Line 5 sends out an HTTP OK response with the response body containing the original request body. To attach the function to the API, the service function is applied to the HttpServer object [21]. The service function only takes the function object as a parameter. Thus, the API is: [21]:

```
1  use actix_web::{App, HttpServer};
2  use actix_web::{post, HttpResponse, Responder};
3
4  #[post("/echo")]
5  async fn echo(req_body: String) -> impl Responder {
6      println!("{:?}", req_body);
7      HttpResponse::Ok().body(req_body)
8  }
9
10 #[actix_web::main]
11 async fn main() -> std::io::Result<()> {
12     HttpServer::new(|| {
13         App::new()
14     })
15     .service(echo)
16     .bind("127.0.0.1:8080")?
17     .run()
18     .await
19 }
```

For this project, the API is declared using:

```
1  use actix_web::{App, HttpServer};
2  use actix_cors::Cors;
3  mod model;
4  mod ga;
5  mod aco;
6  extern crate chrono;
7
8  #[actix_web::main]
9  async fn main() -> std::io::Result<()>{
10     let url = match cfg!(debug_assertions){
11         true => "127.0.0.1",
12         false => "backlogburner.com"
13     };
14     println!("Hosting URL: {}", url);
15     HttpServer::new(|| {
16         let cors = Cors::permissive();
17         App::new()
18             .wrap(cors)
```

```
19                  .service(model::getNewSchedule)
20      })
21      .bind((url, 5000))?
22      .run()
23      .await
24  }
```

Lines 3-5 import the genetic algorithm and the ant colony optimization algorithm. Lines 10-13 determines which URL the API should be hosted on. If the API is run under a development environment, then the API will be hosted on "127.0.0.1". Otherwise, it will be hosted on backlogburner.com. In this project, the getNewSchedule function implements the GACO algorithm. Thus, line 19 attaches the getNewSchedule function to the web API.

5.3.2 Implementing GACO

The GACO algorithm is implemented below:

```
1  #[post("/model")]
2  async fn getNewSchedule(user_data: Json<UserData>) ->
       HttpResponse{
3      let model_data:Vec<(i128, i128)> = user_data.
           ConvertUserData();
4      let endValue: i128 = (user_data.EndOfCycle.timestamp() -
           user_data.monday.timestamp()) as i128/300;
5      let freeIntervals = ga::getListOfFreeTime(&model_data,
           endValue as f32);
6      let pool = ga::run(100, &model_data, &user_data.newEvent,
           endValue as f32);
7      let selectedSolution = aco::run(100, &user_data.newEvent, &
           freeIntervals, &pool, &model_data);
8      let newEvents = ConvertToScheduleData(&selectedSolution, &
           user_data);
9      HttpResponse::Ok()
10         .content_type("application/json")
11         .json(newEvents)
12 }
```

The getNewSchedule function uses the JSON form of the VueJS application data as a parameter variable. The function is designed to expect a HttpResponse object to be returned. Thus, lines 13-15 returns an OK HTTP response with the new events

attached to the response. The new events are converted into a JSON format. Line 3 converts the user data into a form that is more accessible to the GACO algorithm. The implementation details are discussed in 5.3.4. Line 5 calculates all the time blocks, within the ranges of the user data, that are not being used by any event. Line 6 employs the genetic algorithm. The details of this function and the genetic algorithm implementation are discussed in 5.3.5. The ga::run function returns a set of possible solutions. Line 7 applies the ant colony algorithm optimization section of the GACO algorithm. This line returns a single solution. Line 8 coverts the single solution from the model space into a more human-readable format. The implementation of this function is explained in 5.3.4.

5.3.3 Modeling Our Data

Whilst the Rust HTTP API implements the GACO algorithm, it also handles transforming the data from the VueJS application into information that the model can use. The user data is abstracted to the UserData struct.

Listing 5.14: The UserData struct

```
1 #[derive(Debug, Serialize, Deserialize, Clone)]
2 pub struct UserData{
3 pub listOfEvents: Vec<HardConstraint>,
4 pub monday: DateTime<Utc>,
5 pub EndOfCycle: DateTime<Utc>,
6 pub newEvent: Vec<RequestedEvent>
7 }
```

The UserData struct contains a list of events from the user's schedule. This list is named *listOfEvents*. The *monday* variable represents the starting time of this slice. The *EndOfCycle* variable represents the ending time of this slice.

Each element in *listOfEvents* is abstracted into a HardConstraint struct.

Listing 5.15: The HardConstraint struct

```
#[derive(Debug, Serialize, Deserialize, Clone)]
pub struct HardConstraint{
    pub class: String,
    pub end: DateTime<Utc>,
    pub source: String,
    pub start: DateTime<Utc>,
    pub title: String
}
```

The *class* variable represents the types of the event. The *start* and end variables represent the start and end times of the event. The *source* variable represents the origin of the event. The *title* variable represents the name of the event.

In the UserData struct, the newEvent variable represents the list of new events requested by the user. Each new event is abstracted into a RequestedEvent struct.

Listing 5.16: The RequestedEvent struct

```
#[derive(Debug, Deserialize, Serialize, Clone)]
pub struct RequestedEvent{
    pub class: String,
    pub length: f32,
    pub selectedDate: DateTime<Utc>,
    pub source: String,
    pub title: String
}
```

The *class* variable represents the types of the newly requested event. The *source* variable represents the origin of the newly requested event. The *title* variable represents the name of the newly requested event. The *length* variable represents the length of the newly requested event. The *selectedDate* variable represents the intended time slice of the user data.

The user data is transformed into the model data through the ConvertUserData function.

```
pub fn ConvertUserData(&self) -> Vec<(i128, i128)>{
    let mut newVec = Vec::<(i128, i128)>::new();
    for x in &self.listOfEvents{
        newVec.push((
                (x.start.timestamp() - self.monday.timestamp())
                    as i128/300,
```

```
6                       (x.end.timestamp() - self.monday.timestamp())
                            as i128/300
7               ));
8           }
9           return newVec;
10      }
```

Lines 5-6 calculates the number of seconds between the beginning of the slice to the starting and ending times of the event and divides this number by 300. The number that comes from this conversion is the number of 5 minute time blocks from the beginning of the schedule slice. The ConvertUserData function stores these number in a list and exports this list. The converted user data is stored in the variable called *model_data*.

When the GACO algorithm is finished, the solution created must be converted from the model space into a more accessible format for the VueJS application. This conversion is handled in the ConvertToScheduleData function.

Listing 5.17: The ConvertToScheduleData function

```
1   pub fn ConvertToScheduleData(newEventModelData: &Vec<Vec<(i128,
        i128)>>, userData: &UserData) -> Vec<Vec<HardConstraint>>{
2       let mut scheduleData = Vec::<Vec<HardConstraint>>::new();
3       for newEvents in newEventModelData{
4           let mut eventList = Vec::<HardConstraint>::new();
5           for i in 0..userData.newEvent.len(){
6               let mut new_hc = HardConstraint::new();
7               new_hc.start = DateTime::<Utc>::from_utc(
8                   NaiveDateTime::from_timestamp(
9                       ((newEvents[i].0 * 300) as f32 + (userData.
                            monday.timestamp() as f32)) as i64, 0 as
                             u32)
10                      , Utc);
11              new_hc.end = DateTime::<Utc>::from_utc(
12                  NaiveDateTime::from_timestamp(
13                      ((newEvents[i].1 * 300) as f32 + (userData.
                            monday.timestamp() as f32)) as i64, 0 as
                             u32)
14                      , Utc);
15              new_hc.class = "hc".to_string();
16              new_hc.title = userData.newEvent[i].title.clone();
17              new_hc.source = "A".to_string();
18              eventList.push(new_hc);
19          }
20          scheduleData.push(eventList);
```

```
21      }
22      return scheduleData;
23  }
```

Lines 7-14 multiplies the model space numbers by 300 and adds this number to the starting time of the slice. Lines 15-17 attaches necessary qualitative data to the newly converted event.

5.3.4 Genetic Algorithm

The Genome

Within the Rust API backend, each genome is abstracted to the following struct:

```
1  #[derive(Debug, Clone)]
2  pub struct GAPath{
3      pub fitness: f32,
4      pub schedule: Vec<(i128, i128)>,
5      pub newEventsIndex: Vec<usize>,
6      pub EndOfCycle: f32
7  }
```

The fitness variable describes the fitness of the genome. The schedule variable represents a possible new schedule for the user within the model space. The *newEventsIndex* variable represents a list of indices, for the schedule variable, that access the timings of the user submitted hobbies. *EndOfCycle* represents the model space value of the end of the user's schedule splice. This variable is used in calculating fitness.

Each genome is initialized by the following method:

Listing 5.18: Genome Intialization Method

```
1  pub fn init(&mut self, hardConstraints: &Vec<(i128, i128)>,
       listOfRequestedEvents: &Vec<model::RequestedEvent>,
       EndOfCycle: f32){
2      let mut temp : MyPheno = MyPheno{
3          EndOfCycle: EndOfCycle,
4          schedule: hardConstraints.clone(),
5          model_data: hardConstraints.clone(),
```

```
6           newEventsIndex: Vec::<usize>::new()
7       };
8       let ListOfFreeTime = ga::getListOfFreeTime(hardConstraints,
            EndOfCycle);
9       let mut rng = thread_rng();
10      let mut listOfNewEvents = Vec::<(i128, i128)>::new();
11      for x in listOfRequestedEvents{
12          let mut freeTime = ListOfFreeTime.choose(&mut rand::
                thread_rng()).unwrap();
13          while freeTime.1 - freeTime.0 < (x.length/5.0) as i128{
14              freeTime = ListOfFreeTime.choose(&mut rand::
                    thread_rng()).unwrap();
15          }
16          let mut freetime_length = freeTime.1 - freeTime.0;
17          let mut multiple: i128 = rng.gen_range(0 .. (
                freetime_length as f32/(x.length/5.0)) as i128);
18          let mut start = freeTime.0 + (multiple * (x.length/5.0)
                as i128);
19          let mut sample_schedule: (i128, i128) = (start, start +
                (x.length/5.0) as i128);
20          while ga::checkViolations(&sample_schedule,
                hardConstraints){
21              freeTime = ListOfFreeTime.choose(&mut rand::
                    thread_rng()).unwrap();
22              while freeTime.1 - freeTime.0 < (x.length/5.0) as
                    i128{
23                  freeTime = ListOfFreeTime.choose(&mut rand::
                        thread_rng()).unwrap();
24              }
25              freetime_length = freeTime.1 - freeTime.0;
26              multiple = rng.gen_range(0 .. (freetime_length as
                    f32/(x.length/5.0)) as i128);
27              start = freeTime.0 + (multiple * (x.length/5.0) as
                    i128);
28              sample_schedule = (start, start + (x.length/5.0) as
                    i128);
29          }
30          temp.schedule.push(sample_schedule);
31          listOfNewEvents.push(sample_schedule);
32      }
33      temp.schedule.sort_by(|a, b| a.partial_cmp(b).unwrap());
34      for x in listOfNewEvents{
35          let index = temp.schedule.iter().position(|&r| r == x).
                unwrap();
36          temp.newEventsIndex.push(index);
37      }
38      return temp.clone();
39  }
```

Lines 11-32 goes through each user submitted hobby and randomly selects a timing for each user submitted hobby. Let h represent a user hobby in $listOfRequestedEvents$.

Line 12 chooses a random range of non-allocated time blocks. Lines 13-15 check if the chosen range of non-allocated time blocks can contain hobby *h*. If the chosen range can not contain *h*, then a new range of non-allocated time blocks is chosen. Let the range of non-allocated time blocks be denoted by *r*. This process is continued until a range of time blocks that can contain the hobby *h* is chosen. Lines 16-19 finds a time for *h* within *r*. Lines 20-29 checks if the new timings for hobby *h* conflict with the user's schedule. If there is a conflict, new timings are determined for *h*. Otherwise, the timings are added to the schedule variable of the genome. Line 34-37 find the position of hobby *h* in the genome's schedule and store that index in a list.

Implementing The Genetic Algorithm

The genetic algorithm is implemented in the run function.

Listing 5.19: The run function for the ga crate

```
1  pub fn run(population: i32, hardConstraints: &Vec<(i128, i128)
       >, listOfRequestedEvents: &Vec<model::RequestedEvent>,
       EndOfCycle: f32) -> (Vec<MyPheno>, f32){
2      let mut pop: Vec<MyPheno> = (0..population).map(|i| ga::
           init2(hardConstraints, listOfRequestedEvents, EndOfCycle
           )).collect();
3      let mut s = Simulator::builder(&mut pop)
4          .set_selector(Box::new(UnstableMaximizeSelector::new
               (10)))
5          .set_max_iters(50)
6          .build();
7      let mut temp = StepResult::Success;
8      let mut counter = 0;
9      let mut fitnessTuple = (0.0, 0.0);
10     let mut fitnessSum = 0.0;
11     let mut popCopy = s.population();
12     fitnessSum = ga::calculateSumFitness(&popCopy);
13     fitnessTuple.0 = fitnessSum/(popCopy.len() as f32);
14     loop {
15         temp = s.checked_step();
16         fitnessTuple.1 = fitnessTuple.0;
17         match temp {
18             StepResult::Failure => {
19                 println!("Failure");
20                 break;
21             },
```

```
22                StepResult::Done => {
23                    break;
24                },
25                _ => {
26                    fitnessSum = ga::calculateSumFitness(&popCopy);
27                    fitnessTuple.0 = fitnessSum/(popCopy.len() as
                          f32);
28                    if fitnessTuple.1/(fitnessTuple.0.powf(3.0)) <
                          0.03 {
29                        counter += 1
30                    }else{
31                        counter = 0;
32                    }
33                    if(counter == 3){
34                        break;
35                    }
36                }
37            }
38        }
39        popCopy = s.population();
40        popCopy.sort_by(|a, b| (a.fitness()).partial_cmp(&b.fitness
              ()).unwrap());
41        let mut selectedSolutions : Vec<MyPheno> = Vec::new();
42        for i in 0..( (population * 10)/ 100) as usize{
43            selectedSolutions.push(popCopy[i].clone());
44        }
45        let mut avg_fitness = 0.0;
46        for x in &selectedSolutions{
47            avg_fitness += x.fitness() as f32;
48        }
49        avg_fitness = avg_fitness/ (selectedSolutions.len() as f32)
              ;
50        return (selectedSolutions, avg_fitness);
51    }
```

The run function takes the population size, the user's schedule slice, and the list of user submitted hobbies as input. The run function returns a list of genomes and the average fitness value as output. The fitness value is determined by calculating the smallest interval between events. Once calculated, the length of this interval is recorded and stored as the fitness value. Line 2 initializes the genome pool. Lines 12-13 calculates the average fitness of the initialized pool. Lines 15-39 are repeated until the ratio of the two-most recent generations' average fitness are less than 0.3 or until lines 11-29 have been repeated for 50 generations. Line 15 creates a new genome generation. Lines 18-21 prints "Failure" when there is an error whilst

creating the new generation of the genome pool. Lines 22-24 stops simulating the genetic algorithm. Lines 26-35 checks if the ratio of the average fitness of the newest generation to the average fitness of the older generation raised to the power of 3 is less than 0.03. Lines 39-44 collects 10 of the fittest genomes of the latest genome pool. Lines 47-49 calculates the average fitness of the selected genomes from lines 39-44.

5.3.5 Ant Colony Optimization Algorithm

Within the Ant Colony Optimization area of the API, the ant is described by the following struct:

Listing 5.20: Ant Struct

```
1 #[derive(Debug, Clone)]
2 pub struct Ant{
3     pub path: Vec<(i128, i128)>,
4     pub curr_node: (i128, i128),
5     pub EndOfCycle: f32
6 }
```

The path variable represents the path the ants travel on the graph. The *cur_node* variable represent the current node of the ant. The *EndOfCycle* variable is the value of the last time block of the user's schedule time slice.

Each node on the graph is described by the following struct:

Listing 5.21: Node Struct

```
1 #[derive(Debug, Clone, Copy)]
2 pub struct Node{
3     pub interval: (i128, i128),
4     pub pheromone: f32
5 }
```

The interval variable represents the range of time blocks associated with the node. The pheromone variable represents the pheromone value of the node.

The graph for the ant colony Optimization algorithm is described by the following code snippet:

Listing 5.22: Graph Struct

```
#[derive(Debug, Clone)]
pub struct Graph{
    pub nodes: Vec<Vec<Node>>
}
```

The nodes variable is a list of node lists.

The graph struct is initialized by the following code snippet:

Listing 5.23: Graph Intialization

```
pub fn init(&mut self, list_of_new_events: &Vec<model::
    RequestedEvent>, list_of_free_intervals: &Vec<(i128, i128)>)
    {
        let mut nodes = Vec::<Vec<Node>>::new();
        for new_event in list_of_new_events{
            let mut node_list = Vec::<Node>::new();
            for interval in list_of_free_intervals{
                for i in 1..((interval.1 - interval.0) as f32/(
                    new_event.length * 12.0)) as i128 + 1{
                    let mut node: Node = Node{
                        interval: (0, 0),
                        pheromone: 0.0
                    };
                    node.interval = (interval.0 + ((i-1) as f32
                        * new_event.length * 12.0) as i128 ,
                        interval.0 + (i as f32 * new_event.
                        length * 12.0) as i128);
                    node.pheromone = 0.0;
                    node_list.push(node);
                }
            }
            nodes.push(node_list);
        }
        self.nodes = nodes;
}
```

This function inputs a list of user submitted hobbies and a list of all non-allocated time blocks. Line 2 creates a temporary form of the graph variable named *nodes*. Lines 6-16 calculates each possible interval for each user submitted hobby. Once

these intervals are calculated, they are added to the *nodes* variable. At the end of the function, the *nodes* variable is outputted.

Before the ant colony optimization algorithm is run, the ants are initialized and the graph is updated. The solutions from the genetic algorithm step are applied to the ant colony optimization graph. For each interval mentioned in the genetic algorithm solutions, the pheromone value of the corresponding node is increased by 10.

With the graph updated, and the ants initialized, the ant colony optimization algorithm can begin. The steps of the ant colony optimization algorithm is described in the following snippet:

Listing 5.24: Ant Colony Optimization Loop

```
1  let mut counter = 0;
2  while !is_over(&aco_graph, counter){
3      for i in 0..aco_graph.nodes.len(){
4          aco_graph.nodes[i].retain(|&x| x.pheromone != 0.0);
5      }
6      reinitalize_ants(&mut ants, population);
7      move_ants(&aco_graph, &mut ants, &pheromone_sum, model_data
           );
8      update_pheromone(&mut aco_graph, &mut ants, chromosones[0].
           EndOfCycle, model_data);
9      pheromone_sum = get_pheromone_sum(&aco_graph);
10     counter += 1;
11 }
```

Line 1 creates a counter variable named *counter*. This variable counts the amount of times lines 3-12 are run. Line 2 checks if the ending conditions are met. The ending conditions are: if all the ants follow a single path or that *counter* exceed 50. Lines 3-5 remove any nodes with a pheromone value of 0. Line 6 reinitializes the ants by clearing their paths and setting their position to the start node. Line 7 moves the ants to the end node using the pheromone principal described in Section 3.1. The heuristic information here is the fitness value of the combination of the current path of the ant, the user's schedule slice and the node being evaluated.

Once all the ants are at the end node, the ant colony optimization graph is updated. For the paths the ants travelled, the pheromone value of each node within those paths are increased by 20. Then, the pheromone value for each node in the ant colony optimization graph is reduced by 10. This update happens in Line 8. Line 8 calculates the *pheromone_sum*. This value is used when the ants move from node to node.

CHAPTER 6

Results

To evaluate the accuracy of the application, Backlog Burner was used by 21 College of Wooster students. This chapter is devoted to detailing the results of this testing. The students' responses for each question are available in Appendix C.

6.1 Testing Procedure

The experiment of this study is split into 3 stages:

1. Before the student uses the application
2. Whilst the student uses the application
3. After the student uses the application

Each stage of the experiment consists of the student using a feature of the software and responding to questions about that feature.

Before the student uses the application

The experiment begins by explaining the software being tested. Once the application has been explained, the student is asked a few preliminary questions about themselves. These questions are in Appendix B.2 under the "Before The User Uses The App" subsection. Then the student logs in into the application.

Whilst the student uses the application

The student then proceeds to add their current schedule and hobbies. Whilst the student is using the application, questions about the participant's experience are asked. The specific questions are detailed in Appendix B.2 under the "At The End of The Session" subsection.

After the student uses the application

Once the new schedule is received, the student is asked about their thoughts on the application and their new schedule. The specific questions asked are in Appendix B.2 under the "Whilst The User Is Using The Application" subsection.

6.2 The Students

These students are from the College of Wooster, classes of 2022, 2023, and 2024. Referring to Figure 6.1, the sample population has a biased towards the class of 2022. The majority of the students are Computer Science or Mathematics majors. From 1-10, with 10 being the busiest and 1 being the least busy, the student gave a rating of 7.57 on average. In qualitative terms, the students surveyed are busy but not overwhelmed. Interestingly, each class year reported similar numbers except for the class of 2023.

Class Year	Average Busyness Rating
2022	7.40
2023	8.20
2024	7.33

Table 6.1: Student Class Year with Average Student Busyness Rating

All these students have a variety extracurricular activities. The most common extracurricular activities are: Wooster Clubs, reading and video games.

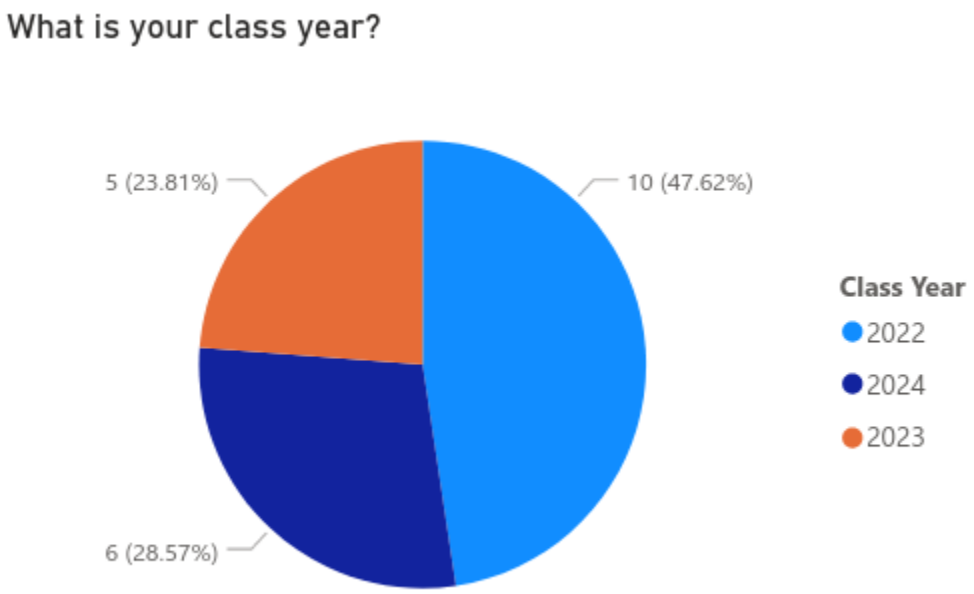

Figure 6.1: Visual Representation of the participant's class years

6.3 How Well Does Backlog Burner Organize Hobbies?

From 1-10, with 10 being good and 1 being bad, participants rated 7.05 on average on how they liked the schedule given by Backlog Burner. In qualitative terms, the students found their new schedule decently favorable. Most of the students gave a rating of 7 when asked about their new schedule.

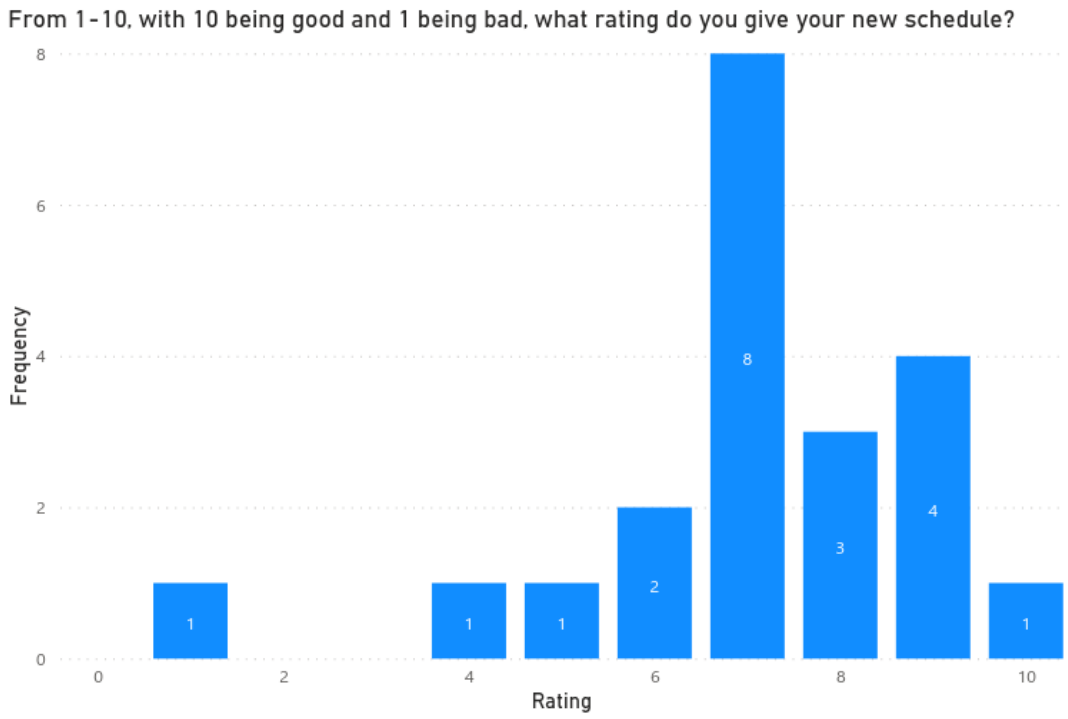

Figure 6.2: Distribution of student schedule ratings

Using the distribution figure above, about 76% of the rating are in the 7-10 range. The rating of 7 serve as both the median and mode.

6.4 Student Thoughts of Backlog Burner Application

Generally, the participants found the website easy to use. On average, the student rated 7.81 on how comfortable they were with the website.

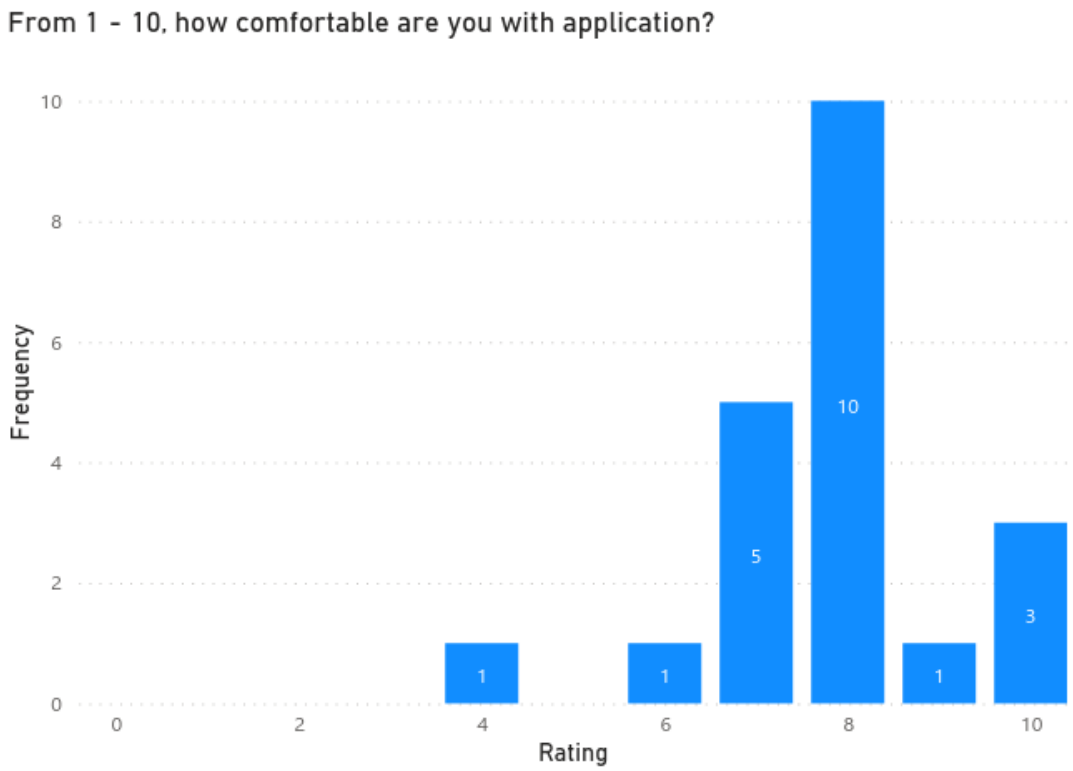

Figure 6.3: Distribution of student comfortable ratings

67% of the student participant rated between 8-10. A few students found the visual layout of the website to be confusing. Furthermore, the descriptions of the displayed options confused some participants. There was also confusions with selecting the hobby times that were given by the GACO algorithm. Some students wished there was more color coding for events as well. Despite these issues, 86% of the participants said they would use Backlog Burner in the future.

CHAPTER 7

Conclusion

The goal of this project was to create an automated scheduling program designed for the College of Wooster student population. This program uses the user's current schedule and a list of hobbies and outputs a proposed the user's schedule where the hobbies are organized not to conflict with the user's classes or commitments. The integer programming, ant colony optimization and genetic algorithm models designed to solve the nurse scheduling problem were discussed and analyzed. Combing properties from these models, an algorithm was purposed that could schedule a user's hobbies. The software developed in this project was used by 21 College of Wooster students. On average, the students found their new schedule decently favorable. In a numerical sense, the students broadly rated their new schedule 7.05. Whilst the software achieves its purpose, there are areas where Backlog Burner can be improved upon.

7.1 Future Work

Fitness Function

The fitness function for the GACO algorithm does not account the user's inclination for when a hobby should be organized. For instance, a user might want to read for three hours and play poker for three hours with friends in the evening. From

the perspective of the GACO algorithm, reading for three hours is equivalent to playing poker for three hours. Thus, the algorithm organizes these hobbies in the same manner despite their differences. A good next step is to improve the fitness function so that it accounts for the user's inclination for specific time frame.

Visual Interface

The current visual layout of the application confused a selection of the student participants. This is due to how there is no visual indicator towards how the application should be used. Thus, a good extension for this project will be redesigning the visual layout of Backlog Burner. The visual interface can be improved to accommodate people's disabilities. This can be achieved by using a color scheme more accommodating to those with color blindness. The entire application should be accessible for screen readers. Backlog Burner's layout should be responsive to how zoomed in or out the web browser in. This feature will help users differing levels of vision.

Entering Schedules and Hobbies

The process of entering hobbies and schedules can be improved. Currently, the user presses a button and a pop-up appears with a variety of options to add the user's schedule.

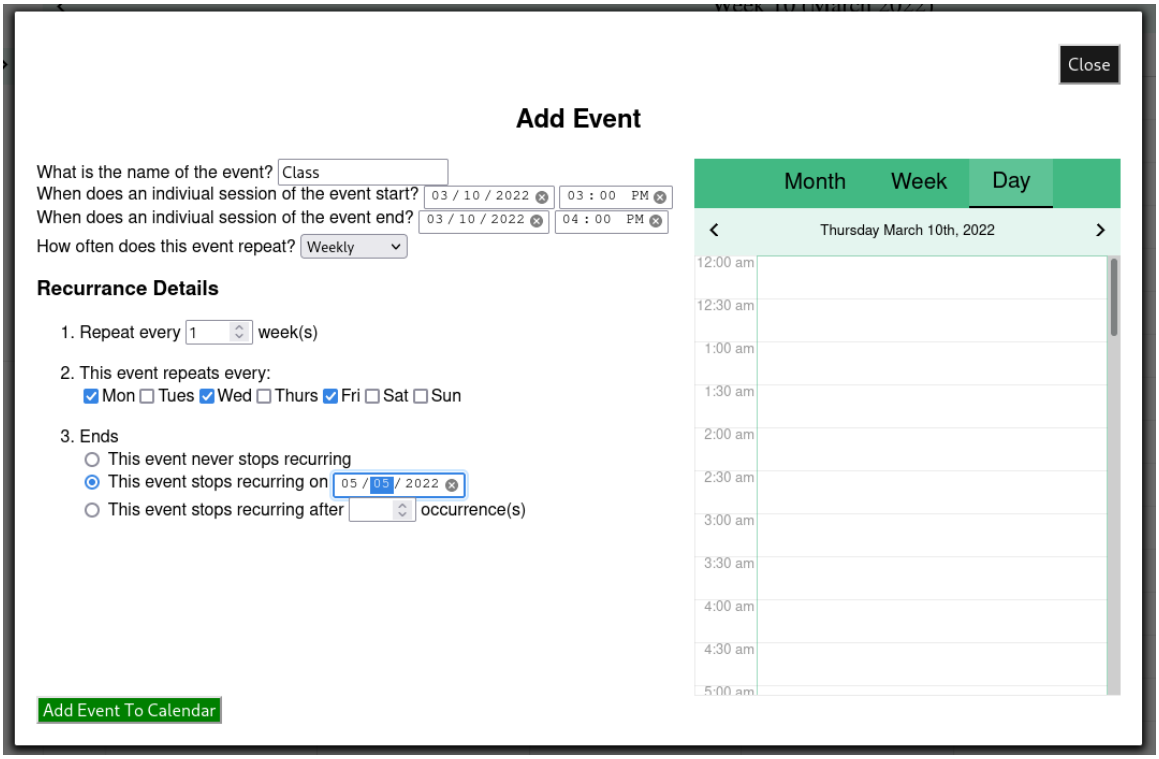

Figure 7.1: Screenshot of the popup that adds in the user's schedule

The current process of entering a user's schedule relies on the user's ability to perfectly understand the options given to them. These options are explained by the text to the left of the input areas. A portion of the student participants were confused on the meaning of each input field in the pop-up forms. One avenue of future work is to improve how the application is communicated to the user. This feature can be achieved through reducing the reliance of text and using icons to explain each options. A video or document explaining the use of option should reduce the amount of confusion.

New Platforms

Currently, Backlog Burner is designed for desktop use. However, a majority of college students primarily use mobile phones. Hence, Backlog Burner should be

adaptable for phone use. This could be accomplished by making the application visually fit within a phone screen or by creating a phone app.

Connecting to other schedules

As of writing, Backlog Burner only incorporates the user's current schedule. One avenue of future work is the ability to incorporate other schedules with the user's current schedule. These other schedule could be a facility's operating hours or other student schedules. Furthermore, Backlog Burner should be able to save the current schedule to other scheduling applications.

Connecting to other application

Backlog Burner solely relies on the user creating hobbies for the application to schedule. One area of future work is connecting Backlog Burner to other applications like Netflix to show a selection of movies based on the availability of the user. Furthermore, the application can be connected to applications similar to Slack or Microsoft Teams in order to schedule leisure activities at work.

New Survey

The survey explained in Chapter 6 was very limited. The participant population did not represent the student body of the College of Wooster. Thus, Backlog Burner should be tested with a larger population of student body. The new set of participants should from each class year at the College of Wooster. However, no class year should be a majority of the population. In addition, Backlog Burner should ve tested against students that do not study at the College of Wooster. This insures that Backlog Burner can handle a variety of schedules since the College of Wooster students' have similar schedule structures.

APPENDIX A

Usability Test Permission Form

Purpose

You are being asked to participate in a research study. We are investigating the effectiveness of a scheduling program designed for College of Wooster students.

Procedures

If you decide to volunteer, you will be asked to answer several questions about yourself, and your experience with the scheduling program. The experiment will take approximately 15 minutes to complete.

Risks

There are no risks associated with the experiment

Benefits

You may receive an improved schedule.

Compensation

There is no compensation with this experiment.

Confidentiality

Any information you give will be held confidential. Unique number codes will be stored on a Microsoft Word file. This file will be destroyed once all data is collected. Thus, all data will become anonymous at the conclusion of the study.

Costs

There is no cost to you beyond the time and effort required to complete the procedure described above.

Right to Refuse or Withdraw

You may refuse to participate in the study. If you decide to participate, you may change your mind about being in the study and withdraw at any point during the experiment.

Questions

If you have any questions, please ask me. If you have additional questions later, you can contact me by email at aolubusi22@wooster.edu. You may also contact my advisor, Thomas Montelione, at tmontelione@wooster.edu.

Consent

Your signature below will indicate that you have decided to volunteer as a research subject, that you have read and understand the information provided above, and that you are at least 18 years of age.

Signature of participant:

Date:

APPENDIX B

Usability Test Script

B.1 Introductions

Hello, USER. My name is Anjololuwa Olubusi. Thank you for joining this session. Before we test the software, I believe it is best we go over the purpose of this session before diving into the website. This testing session is meant to assess the usability of this website. While you are using the website, I am going to ask some questions about your experience.

B.2 Survey Questions

Before The User Uses The App

1. What is your major/majors?
2. From 1-10, with 10 being the busiest and 1 being the least busy, how busy would you say you are?
3. Do you do any extracurricular activities?
4. What are your hobbies?

Whilst The User Is Using The Application

5. Just from glancing at the screen, does the website seem easy to use?
6. (When the user is adding a new event) Are the presented options sufficient?
7. If not, what options should be added?
8. Does there seem to be anything confusing about the application?

At The End of The Session

9. From 1-10, with 10 being the absolutely perfect and 1 being god awful, how what rating do you give your new schedule?
10. From 1-10, how comfortable are you with this application?
11. Were there features you excepted that were not there?
12. If so, please list them?
13. Do you believe that the application properly handled any errors that may have occurred?
14. Do you imagine yourself using this application in the future?

B.3 Tasks

1. Login
2. Add New Event/Events
3. Receive New Schedule
4. Engage Satisfaction Of User

APPENDIX C

Participant Data

Participants 4 - 6

ID	4	5	6
What is your class year?	2022	2022	2023
What is your major/-majors?	Urban Studies	Chemistry	History/Philosophy Double Major
From 1-10, with 10 being the busiest and 1 being the least busy, how busy would you say you are?	5	6	7
Do you do any extracurricular activities?	Yes	Yes	Yes

If so, what are your hobbies?	Photography, people watching, listening to music	Playing guitar, playing piano, learning music theory, cooking, baking, and playing video games.	Poker, reading, clubs, games, puzzles.
Just from glancing at the screen, does the website seem easy to use?	No	Yes	Yes
What difficulties do you see with the website?	Visual hiearchy of more importatn and less important information.		
(After you have entered in your schedule) Are the presented options sufficient?	Yes	Yes	Yes
If not, what options should be added?			
Does there seem to be anything confusing about the application?	Yes	Yes	Yes

If so, what are those confusions?	hiearchy structure and form	For the hobby there was an extra spot for how long the hobby would go for which seemed repetitive since before filling that out the start and end date were already determined.	Some of the language around recurring events, timeframes (the text boxes for 'when does this event start, when does this event end' when creating new events aren't immediately intuitive in meaning). When creating a hobby, "within which days do you want to derive the time for this hobby" was a bit confusing, especially as you already can give ranges of days for it to occur on).
From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	7	7	7
From 1-10, how comfortable are you with this application?	7	8	8

Were there features you excepted that were not there?	No	No	No
If so, please list them?			
Do you believe that the application properly handled any errors that may have occurred?	Yes	No	No
Do you imagine yourself using this application in the future?	Yes	Yes	Yes

Participants 7 - 9

ID	7	8	9
What is your class year?	2022	2022	2022
What is your major/-majors?	Computer Science, Mathematics	Computer Science & Economics	Communication Studies
From 1-10, with 10 being the busiest and 1 being the least busy, how busy would you say you are?	10	7	8

Do you do any extracurricular activities?	Yes	Yes	Yes
If so, what are your hobbies?	Gym, Ballroom	Watching basketball and working out	Writing, Music, Cooking/Baking, Gaming
Just from glancing at the screen, does the website seem easy to use?	Yes	Yes	Yes
What difficulties do you see with the website?			
(After you have entered in your schedule) Are the presented options sufficient?	Yes	Yes	Yes
If not, what options should be added?			
Does there seem to be anything confusing about the application?	Yes	Yes	No

If so, what are those confusions?	Not sure how to remove an event after adding it. Easy to remove hobby. Maybe you can add something like that for events as well.	The how many weeks does this activity repeat (but also I'm dumb and it made sense immediately when explained).	
From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	7	1	7
From 1-10, how comfortable are you with this application?	8	8	8
Were there features you excepted that were not there?	No	No	No
If so, please list them?			
Do you believe that the application properly handled any errors that may have occurred?	Yes	No	Yes

Do you imagine yourself using this application in the future?	Yes	No	Yes

Participants 10 - 12

ID	10	11	12
What is your class year?	2022	2022	2022
What is your major/-majors?	Education and Sociology	Political Science, International Relations	Computer Science
From 1-10, with 10 being the busiest and 1 being the least busy, how busy are you generally?	9	7	7
Do you do any extracurricular activities?	Yes	Yes	Yes

If so, what are your hobbies?	Member of the following clubs: Asia Supporters in Action, Let's Taco Bout Food Club, College of Wooster PC Gaming Club, Student Assistant at Campus Access	ASIA club member, JCA member, reading, playing video games, hanging out with friends, and going to events with said friends.	Drawing, painting, occasionally playing video games
Just from glancing at the screen, does the website seem easy to use?	Yes	No	Yes
What difficulties do you see with the website?		It is very wordy, cluttered, and complex, the rules can be simplified for anyone to understand	
(After you have entered in your schedule) Are the presented options sufficient?	Yes	Yes	Yes
If not, what options should be added?	Color code each individual task so it's eyecatching	Minor entertainment hobbies like reading and games and work like homework	

Does there seem to be anything confusing about the application?	No	No	Yes
If so, what are those confusions?			Depending on what was chosen, the options that appear under recurrence details can be repetitive (e.g: Selecting daily then having to fill out repeat every __days)
From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	8	9	7
From 1-10, how comfortable are you with this application?	8	8	7
Were there features you expected that were not there?	No	No	No
If so, please list them?			

Do you believe that the application properly handled any errors that may have occurred?	Yes	Yes	Yes
Do you imagine yourself using this application in the future?	Yes	Yes	Yes

Participants 13 - 15

ID	13	14	15
What is your class year?	2022	2023	2022
What is your major/-majors?	Music Performance - Voice	Environmental Studies	Mathematics
From 1-10, with 10 being the busiest and 1 being the least busy, how busy are you generally?	8	9	7
Do you do any extracurricular activities?	Yes	Yes	No

If so, what are your hobbies?	COWBelles, A Round of Monkeys, Dukes, The Goliard, Hillel, Jewish Life, Wooster Chorus, Music composition	Reading, photography, electric bass, working out, hiking	
Just from glancing at the screen, does the website seem easy to use?	Yes	Yes	Yes
What difficulties do you see with the website?			
(After you have entered in your schedule) Are the presented options sufficient?	Yes	No	Yes
If not, what options should be added?			
Does there seem to be anything confusing about the application?	No	No	Yes
If so, what are those confusions?			It doesn't automatically add hobbies to schedule.

From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	7	5	4
From 1-10, how comfortable are you with this application?	10	7	8
Were there features you expected that were not there?	No	Yes	No
If so, please list them?		Color coding you're schedule	
Do you believe that the application properly handled any errors that may have occurred?	No	No	No
Do you imagine yourself using this application in the future?	Yes	Yes	No

Participants 16 - 18

ID	16	17	18

What is your class year?	2024	2024	2024
What is your major/-majors?	BCMB	CS major econ minor	Statistical & Data Sciences and Business Economics
From 1-10, with 10 being the busiest and 1 being the least busy, how busy are you generally?	7	5	9
Do you do any extracurricular activities?	Yes	Yes	Yes
If so, what are your hobbies?	ASUKEZWRITING CENTER JOBSTEM ZONE OFFICE HOURS	gym, movies, reading	Watching movies and hiking
Just from glancing at the screen, does the website seem easy to use?	Yes	Yes	Yes
What difficulties do you see with the website?			

(After you have entered in your schedule) Are the presented options sufficient?	Yes	Yes	Yes
If not, what options should be added?			
Does there seem to be anything confusing about the application?	Yes	No	No
If so, what are those confusions?	Add in a prompt to tell user to select the checkbox to add their hobby time		
From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	9	8	9
From 1-10, how comfortable are you with this application?	8	9	10
Were there features you expected that were not there?	No	No	No
If so, please list them?			

Do you believe that the application properly handled any errors that may have occurred?	Yes	Yes	Yes
Do you imagine yourself using this application in the future?	Yes	Yes	Yes

Participants 19 - 21

ID	19	20	21
What is your class year?	2024	2024	2024
What is your major/-majors?	Mathematics and Computer Science	Computer Science and Arts	Computer Science major and French minor
From 1-10, with 10 being the busiest and 1 being the least busy, how busy are you generally?	6	9	8
Do you do any extracurricular activities?	Yes	Yes	Yes

If so, what are your hobbies?	Chess, Gaming, Comics, Sports	Singing, dancing, reading, analyzing, drawing	I am on the Ballroom Dance Club and Equestrian Club on campus.
Just from glancing at the screen, does the website seem easy to use?	Yes	Yes	Yes
What difficulties do you see with the website?			
(After you have entered in your schedule) Are the presented options sufficient?	Yes	Yes	Yes
If not, what options should be added?	Description box for events and hobbies		
Does there seem to be anything confusing about the application?	No	Yes	No
If so, what are those confusions?		The hobby does not show on the calendar	

From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	10	6	9
From 1-10, how comfortable are you with this application?	10	7	8
Were there features you expected that were not there?	No	Yes	No
If so, please list them?		Hobby to show on the calendar	
Do you believe that the application properly handled any errors that may have occurred?	No	Yes	No
Do you imagine yourself using this application in the future?	Yes	No	Yes

Participants 22 - 24

ID	22	23	24

What is your class year?	2023	2023	2023
What is your major/-majors?	Computer Science	Neurobiology	BCMB
From 1-10, with 10 being the busiest and 1 being the least busy, how busy are you generally?	9	8	8
Do you do any extracurricular activities?	Yes	Yes	Yes
If so, what are your hobbies?	Track and Field, art	Watching YouTube videos / Drawing / painting Exercising Walking	Art, watching movies, reading books. being involved with the south asian community
Just from glancing at the screen, does the website seem easy to use?	Yes	Yes	Yes
What difficulties do you see with the website?			

(After you have entered in your schedule) Are the presented options sufficient?	Yes	Yes	Yes
If not, what options should be added?	N/A	/na	
Does there seem to be anything confusing about the application?	Yes	Yes	No
If so, what are those confusions?	A lot of wording describing the thing you are doing	Not being able to select the time for the hobby and had to select the itinerary that best corresponds to the hobby	
From 1-10, with 10 being good and 1 being bad, what rating do you give your new schedule?	7	6	8
From 1-10, how comfortable are you with this application?	7	4	6
Were there features you expected that were not there?	Yes	Yes	No

If so, please list them?	color coding the events	I think it would have been more convenient if I could select the time from the big calendar by clicking the box and then choose its occurrence	
Do you believe that the application properly handled any errors that may have occurred?	Yes	Yes	Yes
Do you imagine yourself using this application in the future?	Yes	Yes	Yes

References

1. Aickelin, Uwe, and Kathryn A. Dowsland. "An indirect Genetic Algorithm for a nurse-scheduling problem." *Computers & Operations Research* 31, 5: (2004) 761–778. https://www.sciencedirect.com/science/article/pii/S0305054803000340. vi, 3, 36, 42, 43, 44, 45

2. Bard, Jonathan F., and Hadi W. Purnomo. "Preference scheduling for nurses using column generation." *European Journal of Operational Research* 164, 2: (2005) 510–534. vi, vii, 17, 18, 19, 20, 21, 22, 23

3. Blum, Christian. "Ant colony optimization: Introduction and recent trends." *Physics of Life Reviews* 2, 4: (2005) 353 – 373. http://www.sciencedirect.com/science/article/pii/S1571064505000333. vi, 2, 24, 25, 26, 27

4. Burke, Edmund K., Patrick De Causmaecker, Greet Van den Berghe, and Hendrik Van Landeghem. "The State of the Art of Nurse Rostering." *Journal of Scheduling* . 46

5. Carter, Michael, Camille C. Price, and Ghaith Rabadi. *Operations Research: A Practical Introduction*. CRC Press, 2018. vi, 2, 5, 6, 7, 8, 9, 11, 17

6. Cheang, B, H Li, A Lim, and B Rodrigues. "Nurse rostering problems—a bibliographic survey." *European Journal of Operational Research* 151, 3: (2003) 447–460. https://www.sciencedirect.com/science/article/pii/S0377221703000213. 46

7. Dorigo, Marco, and Christian Blum. "Ant colony optimization theory: A survey." *Theoretical Computer Science* 344, 2: (2005) 243 – 278. http://www.sciencedirect.com/science/article/pii/S0304397505003798. vi, 26

8. Jan, Ahmad, Masahito Yamamoto, and Azuma Ohuchi. *Search Algorithms For Nurse Scheduling With Genetic Algorithms*, 2002. vi, 39, 40, 41, 42

9. Klabnik, Steve, and Carol Nichols. *The Rust Programming Language: Covers rust 2018*. No Starch Press, 2019. 67

10. Kumar, Rajesh. "What is Bearer token and How it works? - DevOpsSchool.com.", 2022. https://www.devopsschool.com/blog/what-is-bearer-token-and-how-it-works/. 63

11. Macy, Marsh, Alex Buck, Ryan Wike, Eunice Waweru, Celeste de Guzman, Mauricio de los Santos, and Jean-Marc Prieur, 2022. https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-client-application-configuration.

12. Mozilla, 2022. https://developer.mozilla.org/en-US/docs/Glossary/SPA. 52

13. ———, 2022. https://developer.mozilla.org/en-US/docs/Web/CSS. 55

14. ———, 2022. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction. 59

15. ———, 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. 59

16. Ramli, Razamin, Rosshairy Abd Rahman, and Nurdalila Rohim. "A HYBRID ANT COLONY OPTIMIZATION ALGORITHM FOR SOLVING A HIGHLY CONSTRAINED NURSE ROSTERING PROBLEM." *Journal of Information & Communication Technology* 18, 3: (2019) 305 – 326. https://search.ebscohost.com/login.aspx?direct=true&db=iih&AN=140226377&site=ehost-live. vi, 31, 32, 33, 34, 35

17. Sivanandam, S.N., and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer, 2008. 36

18. VueJS, 2022. https://v3.vuejs.org/guide/introduction.html. 52, 53, 58

19. ———, 2022. https://v3.vuejs.org/guide/instance.html. 53

20. ———, 2022. https://v3.vuejs.org/guide/component-basics.html. 53, 56

21. Web, Actix, 2022. https://actix.rs/docs/getting-started/. 67, 68

22. Wu, Jie-jun, Ying Lin, Zhi-hui Zhan, Wei-neng Chen, Ying-biao Lin, and Jian-yong Chen. "An Ant Colony Optimization Approach for Nurse Rostering Problem." In *2013 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2013. http://dx.doi.org/10.1109/smc.2013.288. vi, 27, 28, 29, 30

23. Yilmaz, Ebru. "A Mathematical Programming Model for Scheduling of Nurses' Labor Shifts." *Journal of medical systems* 36: (2010) 491–6. vi, vii, 1, 13, 14, 15, 16, 17, 23

24. Zhang, Wei-guo, and Tian-yu Lu. "The Research of Genetic Ant Colony Algorithm and Its Application." *Procedia Engineering* 37: (2012) 101–106. 4, 47