Automating a Complex Workplace Timetabling Problem

Wil Hutchens
955525

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Master of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

April 27, 2021

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

I would like to dedicate this work to my Nana, she will be forever missed.
RIP.

Abstract

Timetabling problems constitute a set of high dimensional and challenging optimisation problems. In this document, a workplace timetabling problem domain is described in detail, including reasoning behind the need for automating this specific problem and the origin of its complexity. Comparisons are drawn to education timetabling problems, including university course and university exam problems. Popular education timetabling algorithms are discussed, leading to results of multiple algorithm comparisons and analyses. A min-conflicts hill-climbing solution is formulated. This algorithm uses a local search combined with constraint measuring heuristics to find a viable timetable. Performance is measured in terms of CPU time and results are discussed.

How the data is initialised is found to be a large factor in performance, different initialisation methodologies are presented. A soft constraint unknown to educational timetabling prompts a unique methodology. Finally, the proposed solution shows promising results when compared to the domain of university timetabling problems and very promising results when compared to the RNLI's current methods, which will inspire further work.

Acknowledgements

I would like thank my parents for always supporting me and Bertie Muller for being a kind mentor over the last two years and providing help whenever I need it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Timetabling is a heavily researched field in computing that has been dated back to the 1950's *(Phala, J.M 2003)*. However, it is also a field that needs continuous input from algorithm experts because firstly, there is no universal algorithm to be used in all cases *(McMullan et al 2007)*. This stems from the complexity of timetabling, every institution has a different set of rules and constraints meaning they need distinct methodologies to create their timetable. Secondly, many solutions freely available are also often composed for use on a simplified data set, not actually aligning with the complexity of a real world problem *(McCollumn, 2006)*. The most common use of timetabling algorithms is in education systems; lecturers, students, courses and resources must be organised into a weekly schedule. The schedule must meet the hard constraints of all people involved, and soft constraints must be satisfied where possible. Hard constraints are requirements that must be met in order for the timetable to be usable, e.g. a lecturer cannot be teaching two classes at the same time. Whereas soft constraints are requirements that all parties would like to be satisfied but can be foregone if necessary, e.g. students having their exams spread out evenly. In this paper, an intelligent backtrack-free min-conflicts-hill-climbing (MCHC) algorithm for solving a specific workplace timetabling problem is proposed. This is done by searching for local solutions using MCHC, and then allowing the algorithm to come back to previously solved areas after many changes have been made in order to search for new possible solutions.

1.1 Motivations

This project was undertaken because of a specific case that was found in a charity who manages their timetables in a very inefficient way. It seemed it would make sense to design a timetabling system for them as they have resembling complexities of requirements to that of an education establishment. However, their requirements differ somewhat in certain aspects, meaning a typical university or school system would not be applicable. Their current solution is using an excel document and manual calculations to figure out their entire annual timetable. This means for each area (there are around 50, each containing a number of locations and upwards of 60 employees), someone has to work out a suitable set of timetables every week.

For educational institutions, timetabling software is globally adopted as every instance of timetable fits into a grouping of courses, lecturers, students, and rooms, with a few minor variations. On the other hand, most workplaces do not require timetables of the same complexity as they often have set working hours for their entire staff, i.e. nine-to-five jobs. Or teams of staff working in shifts, neither of which need much computation. The RNLI (Royal National Lifeboat Institution), however, has an obscure case of requirements for their scheduling. There are groups of employees working across multiple areas. There are also seven days of time slots to be filled whilst employees are only able to work five days per week. Areas contain multiple locations (normally beaches) and every day each location must have a senior employee, along with a given number of employees with multiple different qualifications. Employees are all graded 1-4 by their experience and each location also requires a certain number of the top 3 grades of employees. A more detailed description of these requirements follows in chapter 2.

The RNLI currently employ around 60 people to manage these timetables (one for each area). This document aims to present a best possible solution to this complex timetabling problem incorporating the most efficient university timetabling methods. The RNLI's scheduling requirements differ somewhat in certain aspects from university timetabling, meaning many adaptations must be made to a university or school system method, as previously mentioned. Every school, university, college and workplace (that does not conform to standard hours for all staff members) is presented yearly with an often highly complicated timetabling problem. Solving these problems could add up to hundreds of thousands of hours but timetabling algorithms have allowed people to come up with solutions to complex problems in seconds. Of course, data must be inputted to systems but this is negligible in comparison to alternative methods.

1.1.1 Objective

In this document, we intend to present the methodologies adopted in creating a solution to the RNLI scheduling problem, along with the results from testing our solution, and how we intend to proceed from here. An exact problem description is given in section 4.2.1.

1.2 Overview

The remainder of chapter 1 outlines the document structure and the key contributions of this work. Chapter 2 describes the timetabling problem in the case of educational timetabling and our workplace problem. Soft and hard constraints are explained and compared. Chapter 3 delves into the different approaches to timetabling and how they are applicable to our workplace timetabling problem. In chapter 4, results from preliminary algorithm tests are considered, the workplace solution's method design is explained along with how constraints are modelled. Results of the final solution are given. Lastly, in chapter 5 results and future work are summarised and a conclusion is drawn.

1.3 Contributions

The main contributions of this work can be seen as follows:

- **An outline of timetabling problems.**

  Educational timetabling; course and examination scheduling. Workplace timetabling.

- **A description of various timetabling methods**

  Evolutionary Algorithms. Graph Colouring. Min-Conflicts Hill-Climbing.

- **An analysis of our solution to a complex workplace problem.**

  A Min-Conflicts Hill-Climbing method is detailed and results are analysed and compared to previous works.

- **A conclusion of our findings.**

  Summary of results and conclusion are given along with future work.

Chapter 2

Timetabling Problems

2.1 Overview

Educational timetabling is a heavily researched area of computing and algorithms and it has been studied for decades, therefore providing a lot of information about possible methodologies. It is also still regarded as a very difficult problem in AI. [4] A number of techniques including graph coloring, genetic algorithms, search based methods and constraint based approaches, etc., for tackling the problem have been studied [5]. This section explains education timetabling, discusses results of previous related work and draws comparisons to our workplace problem, where education methods are relevant or related.

2.2 Educational Timetabling

- **Hard Constraint:** a specification that must be satisfied for a solution to be feasible.
- **Soft Constraint:** a specification that penalises a solution when not satisfied but can be foregone whilst still producing a feasible solution.

2.2.1 Course Timetabling

A course timetable is a weekly university timetable, consisting of a number of lectures, lecturers, students and rooms. Lecturers, courses and students are assigned a number of lecture slots from a predetermined list of time slots and rooms of specified capacities and availability. Courses require a certain number of time slots and lectures are taught 5 days per week. A list of hard constraints must be satisfied. Some examples of typical hard constraints are as follows:

1. Students and lectures must not have any concurrent obligations.

2. Course subjects must have a room for each of their scheduled hours.

3. Rooms cannot be booked by two courses at the same time.

4. Room availability and their capacities must be satisfied.

When finding optimal course timetables, soft constraints must also be considered. A few example soft constraints:

1. Students and lecturers should have an hour break every day. (A non-scheduled time slot that is neither at the end nor the beginning of a block of scheduled time slots.)

2. Course subjects should not have more than one non-consecutive lecture per day.

3. Lecturers may have preferred hours.

4. No student or lecturer should be scheduled for more than a number of continuous hours of teaching/learning without a break.

The aim is then to find a solution that comprises of a weekly schedule of lectures that is deemed acceptable to all people involved whilst satisfying hard as well as soft constraints as efficiently as possible. [6]

2.2.2 Examination Timetabling

The difference between course timetabling and examination timetabling is relatively small. Examination timetabling requires students and exams to be scheduled in rooms in given time periods, however there is a heavy emphasis on optimising the schedule to keep each students' exams spread out as evenly as possible. Schaerf (1999) agrees exams and course problems are similar enough to be modelled using the same model. [5]

2.3 Workplace Timetabling

2.3.1 Introduction

Our workplace problem has many similarities to course and examination timetabling, such as its assigning of people (employees rather than students) to time slots and its hard and soft constraints. This section details the problem and how it relates to educational timetabling.

2.3.2 Specification

Our workplace problem to be solved is outlined here: there are a large number of employees who need to be given locations that they will be working on each day during a week and locations that need a number of employees to be working at every day. Locations require a certain number of employees with different attributes. Employees may have certain attributes. Employees may also either be signed up to work full time, where they must be provided with 5 days of work per week. Or be signed up to part time work, where they can work any number of days from 1 to 5. Finally, locations are open 7 days a week.

The aim of this project is to create a system that is capable of creating and presenting a suitable weekly timetable from a large number of employees, locations and timetable criteria (constraints).

In the RNLI timetabling problem, 3000 employees must be sorted into 250 locations every day of the week. A location is usually a beach, but it can be an office or a boat station, so here we reference as location. Locations have a number of required employees for 7 days per week, whereas employees work up to 5 days per week. They also need employees who have specific amounts of experience and qualifications. An employee can hold one or more of 3 different qualifications; boat driver, boat crewman and jet ski driver. For context, the boat driver qualification is the hardest to acquire and usually takes 4-5 years before you can attempt the exam, the other two can be acquired within 1-3 years of working for the RNLI. Locations require a number of people with these qualifications when they have boats or jet skis at the locations. Employees are also given an overall rank between 1 and 4. Employees of rank 4 are also named senior employees and can manage a location, and every location needs someone of rank 4 every day. Some locations also require a number of rank 3 and 2 employees.

These locations are grouped into areas, an area would typically be made up of a number of beaches, a boat facility, and an office. The RNLI has around 50 areas with an average of around 5 locations per area. All locations represent actual locations around the UK, meaning

employees can be further away or closer to certain locations, which gives rise to a soft constraint to be discussed. As stated in *motivation* 1.1: the RNLI employs around 50 people to manage timetables. This allows each of these timetable managers to base themselves in one area and focus on the timetables within their area. However, this does not mean that each area's timetables are separate, as most areas have to switch and borrow employees from other areas in order to fulfill the requirements of their locations. Our solution builds timetables for each area from a list of all possible employees, but prioritises employees who are located closer to the area in focus. Section 4.4.2 details how this is dealt with in our application.

2.3.3 Constraints

Constraints here are similar to those in a typical educational timetabling problem, but there are some differences that require special attention, such as employees living near areas and further away from others. This requires employees to be matched to areas closer to them and this can be compared to the aspect of optimisation in examination scheduling 2.2.2 where exams must be spread out as much as possible for every student. This is a global optimisation in that it applies to every student at the same time and a perfect solution is the least constraint-violating solution average for every student. In our workplace problem the employees must all work as close as possible to their given location. The other hard and soft constraints for our workplace timetable are as follows:

Hard Constraints:

1. Employees must not have any concurrent obligations, also known as double booked.
2. Locations must have the required amount of employees per day.
3. Locations must have at least the required number of employees ranked 2,3 and 4 every day.
4. Locations must have at least the required number of employees with the required qualifications each day.
5. Employees must not be timetabled to work more than 5 days per week.
6. Full time employees must be offered 5 days of work per week.
7. Part time employees must be offered somewhere between 1 and 5 days per week.

Soft Constraints:

1. Employees should work only in the area closest to them.
2. Full time employees should work mostly at their chosen location.
3. Part time employees should work 2 or 3 days per week.
4. Full time employees should have their two days off consecutively.

Educational timetabling has more complexity in its 8 separate hours per day in comparison to our workplace timetabling having 1 single time slot per day. The workplace timetable is however adversely affected by having more constraints. This list of constraints is not necessarily exhaustive for all cases, as there may be circumstances that can not be foreseen. Such as two employees that work together poorly so must be separated. Constraints like these must be dealt with by a user of this timetabling system. Therefore, the ability to make changes after a timetable is presented is vital. [7] This also allows for users to check for and resolve inconsistencies and errors, and to reflect changes in real world circumstances, e.g. someone is ill and needs covering.

Chapter 3

Timetabling methods

Methodologies considered for our timetabling problem are outlined here.

3.1 Evolutionary algorithms

3.1.1 Genetic Algorithms (GA)

Genetic algorithms (GA) are a widely used form of machine learning. They are based on Darwin's theory of evolution. [8] Defined by, given a target in the form of a set of constraints that need to be fulfilled and an initial permutation of data, an algorithm repeatedly tries many different permutations and the 'fittest' permutation has more chance of passing its attributes to the next generation. Fitness of each permutation is measured by a fitness function. Fitness proportionate selection [9], roulette wheel selection, [10] Pareto Ranking [11] and tournament selection methods are commonly used for selecting best permutation. Tournament selection is described well in Miller, Brad L. (1995) [12]. Genetic operators are then used to create the offspring. Genetic operators commonly used include: crossover, mutation and reproduction. The population is repeatedly refined until the algorithm converges or the termination criteria for the particular problem are met. [13].

Crossover in genetic algorithms is where two so-called parents, that would have been determined to have the greatest fitness, are chosen to pass some of their attributes to the next generation. Sections of both of these two parents are then combined together to create a new offspring, which is a combination of both, much like when two living creatures breed, their genes crossover and their offspring is a mixture of them both.

Mutation comes from real world mutations where new born living creatures can have ran-

dom new traits that do not come from either parent. Mutations are important in evolution as without new changes, offspring can only have the attributes from their parents. No new attributes can ever come. In genetic algorithms, without mutation then the initial data would have to hold all of the ingredients to come up with a solution. Mutation allows random data to evolve into a solution. In a genetic algorithm, random changes are made at each iteration, meaning features that do not currently exist can be formed. This is important in finding new solutions.

Fitness of a solution is how close the solution is to the desired result. Measuring fitness is problem-specific. In timetabling the least amount of constraint violations is usually an accurate way of measuring fitness. Paquete and Fonseca [11] study Pareto Ranking to measure fitness of examination timetables. Pareto dominated solutions are those that an improvement to one individual will not make another worse-off, i.e. fitness is equal to how easy it is to make changes to the timetable without creating more violations elsewhere.

A genetic algorithm is produced in Wong et al. [14] They use the genetic algorithm to create an examination timetable. The timetable managed to adhere to all hard constraints as well as not many students having to sit consecutive exams. Each timetable is given by a matrix that represents a single examination. Each matrix holds the time slot that it would take place on, and the room it would take place in.

Chu and Fang [15] also use a genetic algorithm to come up with a solution to an examination timetabling problem. Each timetable is an array of cells. A single cell represents a single exam and each cell contains the time slot and day the exam has been allocated to. Mutation and crossover operators are applied to generate the next generation. The crossover works by choosing two positions in two given parents. It then swaps these two points. Our algorithm follows this same method for a portion of its calculation. The mutation then works by selecting two examinations and swapping their time slots. The evolution continues until a viable timetable is found or the number of iterations reaches the maximum.

Another genetic algorithm is described in Ozcan and Ersoy. [16] This time, however, they look at combining the GA with a directed hill-climbing method. Using a genetic algorithm with a local search can also be described as a Memetic Algorithm. A timetable is represented in a chromosome, each chromosome contains a number of exams, each exam given in the form of a gene. Each gene then stores the exam's day and time slot. Their crossover method is to use one-point crossover, and the mutation again switches the day or time slot of the exam. The directed hill-climbing is then applied locally to find changes that make each timetable more

optimal. This local search reduces likelihood of premature convergence.

3.1.2 Memetic Algorithms (MA)

First introduced by Moscato and Norman [17], memetic algorithms are evolutionary algorithms where a local search is taken advantage of. We know genes are passed between parents in order to create an offspring, a meme however reproduces itself. Each individual adapts the meme to become as optimal as it can before passing to a new generation. This gives an advantage of a massively reduced search space. This is because every solution finds its local optima before being passed on, meaning the solutions are reduced to the values that are locally optimal. A visualisation of this is given in Burke, E. K. [1]

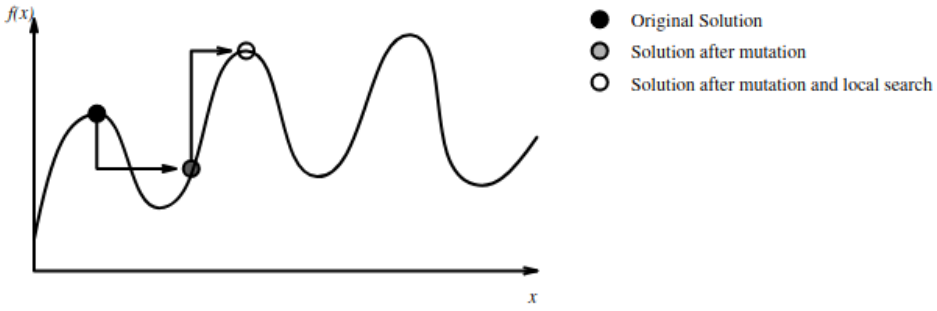

Figure 3.1: Stages of memetic algorithm. Burke, E. K et al. [1]

A child produced from an iteration of the GA produces a solution (given as original solution/the black dot) A mutation would then generate a new solution, this solution is however less optimal than before the mutation. A standard GA would then carry on with this new less optimal solution. The memetic algorithm applies a local search to this new mutated solution, finding a new local optima. This new local optima is more optimal than the pre-mutated solution. This is the premise for a memetic algorithm and promising timetabling solutions have been produced using MAs.

S. N. Jat and S. Yang [2] used a memetic algorithm to solve a university course timetabling problem and found that the addition of local search made their GA more powerful. They looked at two different search techniques, one based on the events and the other based on the time slots.

S. N. Jat and S. Yang give basic description of their memetic algorithm in figure 3.2. Their algorithm follows a standard memetic algorithm: two parents are selected from a population, a child is created using crossover, a mutation is applied, then local search is carried out. The

Algorithm 1 The proposed memetic algorithm

1: **input** : A problem instance **I**
2: **for** $i = 1$ to population size **do**
3: $s_i \leftarrow$ random initial solution
4: $s_i \leftarrow$ solution s_i after Local Search 1 (LS1)
5: $s_i \leftarrow$ solution s_i after Local Search 2 (LS2)
6: **end for**
7: sort population by fitness
8: **while** termination condition not reached **do**
9: select two parents from population by tournament selectio
10: $s \leftarrow$ child solution after crossover with a probability P_c
11: $s \leftarrow$ child solution after mutation with a probability P_m
12: $s \leftarrow$ child solution after applying Local Search 1 (LS1)
13: $s \leftarrow$ child solution after applying Local Search 2 (LS2)
14: child solution s replaces the worst member of the popula tion
15: sort population by fitness
16: $s_{best} \leftarrow$ best solution in the population
17: **end while**
18: **output** : The best solution s_{best} achieved for **I**

Figure 3.2: Memetic algorithm. S. N. Jat and S. Yang [2]

algorithm differs when the child then replaces the worst member of the population. Most often in a genetic algorithm, children are made repeatedly until the number of parents is matched, then all parents are replaced by their children. Replacing the worst member on each iteration improves efficiency greatly, which is sought after in timetabling due to its high computational costs.

3.2 Graph Colouring (GC)

Graph colouring involves the labelling of all vertices in a graph with the minimum amount of differing colours such that no two adjacent vertices are the same colour. A simple sequential GC algorithm is described in Leighton, F. [18], given a graph $G = (V,E)$ where V are vertices/nodes and E edges, the nodes are ordered randomly such that $V = V1,...,Vn$. Colours are then assigned to nodes: The first node, $V1$ is assigned colour number 1, V2 is assigned colour number 2, ect. to Vn. Once a node has been coloured, $Vi+1$ where the ith node is $(1 <= i <= n-1)$ is given the smallest colour number possible where no coloured node adjacent to $Vi+1$ has been assigned the same color number.

This method relies heavily on the ordering of the nodes. A poor ordering of nodes means a

poor colouring of nodes. Therefore, many different methods of ordering the nodes have been developed. E. K. Burke (1994) [19] proposes a solution using room allocation techniques combined with graph colouring and backtracking to design a timetabling system. [20] E. K, Burke (2001) here proposed a solution using graph colouring combined with a genetic algorithm to group the nodes and produced good results on difficult problems.

Babaei [3] describes the simplest form of graph colouring for timetabling, originally proposed by De Werra (1985) [21] as a solution for the first timetabling problem, which was itself proposed by Gotlieb (1965). [22] De Werra modelled the timetabling problem using a non-directional graph. The vertices must be coloured in the least amount of colours where no two adjacent nodes are the same colour number. A timetable must have no conflicts. The timetables are modelled as a group of connected coloured nodes. The events are the nodes and constraints are the edges. Colours are then the time slots. Figure 3.3 shows how a non-directional graph is coloured with the least amount of colours whilst maintaining that all adjacent nodes have different colours.

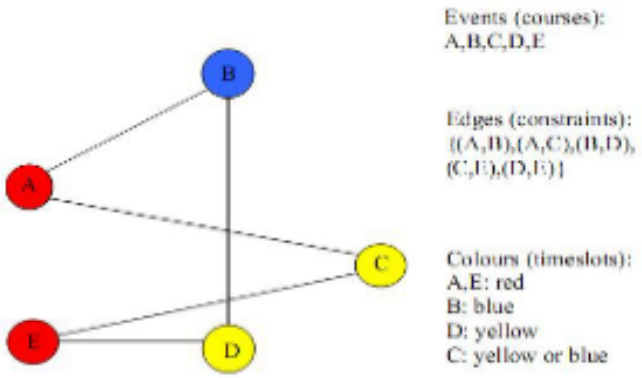

Figure 3.3: Graph colouring visualised. Babaei (2015) [3]

3.2.1 Ant Colony Optimisation (ACO)

Ant algorithms were originally proposed by Dorigo et al [23] as an approach to solving combinatorial optimisation problems (COPs), such as the travelling salesman problem. Ant Colony Optimisation (ACO) is used to find good solutions to GC problems, so when used in conjunction with GC, strong timetabling solutions are found. [24] Ant colonisation is the technique ants use to search for food in their colonies. Multiple ants are sent in different directions to

look for food. When food is found, they release pheromones which direct the other ants in their direction.

Bessedik, Malika [25] details an ant colonisation technique to solve graph colouring problems (CGPs): Ants from a colony are concurrently sent through adjacent states of the graph, they each apply stochastic decisions to optimise their local area of graph. The ants then build solutions to the problem individually. These solutions are then passed between the ants, mimicking the pheromone trails of real ants, and solutions are results of collected ant findings combined together. To construct a solution, the partial solutions from ants are evaluated and the components are used to lead the next iteration of ants to follow optimised paths.

In Eley (2006) [26], an ant system is applied to a timetabling problem. Iterations of ants generate timetables and hill-climbing is applied to the best one of each iteration. The hill-climbing finishes when no more improvements can be made.

3.3 Min-Conflicts Hill-Climbing (MCHC)

Min-conflicts hill-climbing is an algorithm that involves making sequential changes to move a solution in the direction of least possible constraint violations - hence hill climbing. One starts with a random initiation of variables and searches for possible repairs. [27] The decision of what changes to make is controlled by a min-conflicts heuristic (MCH), which finds the variable change that will give the greatest decrease in violations. MCHC can be used as a local search method in conjunction with GAs and other previously discussed methods as a timetabling solution.

MCH has been very successful in solving highly complex constraint satisfaction problems (CSPs) and constrained optimisation problems (COPs) including the well known n-queens problem [28]. Minton et al [27] propose a minimising conflict method to effectively solve CSPs such as n-queens and scheduling problems. They also explain well why MCHC is so effective: upon examining a highly effective neural network which is used on highly complex problems such as observations on the hubble space telescope [29] they found that the effectiveness comes from the networks state transitions guided by trying to reduce the number of inconsistencies (violations) in the network. It does this by changing the value at the neuron whose output is most inconsistent with its input. The network is reassigning the variable whose change in value will bring the biggest reduction in constraint violations. MCHC reproduces this method within its own algorithm, variables are reassigned to produce the largest reduction in constraint violations.

Look-forward heuristic (LFH) is a method of looking ahead at what state the domain will be in after a change has occured, before the change has occured and using the viability of the future states to guide the search in the direction of the most viable domain space. Vincent, T. [30] promotes the look forward heuristic for use with MCHC to solve complex timetabling problems. A pay off between time to compute the future states and time saved by computing future states must be considered before deciding whether to implement LFHs.

These algorithms are implemented to solve timetabling problems and discussed in chapter 4.

Chapter 4

Workplace Timetabling

4.1 Introduction

Our workplace problem (WPP) differs from any previously discussed, which requires methods to be adapted to suit a solution. This section discusses how our problem differs from others and what methods are used to solve this WPP.

4.2 Modelling Data

Educational timetables are made up of time slots, whereas timetables here are made up of day slots, each day has a number of slots that need to be filled and a timetable has 7 days. A day slot contains one employee. Each employee has attributes related to their experience which includes a rank and possible qualifications. Along with whether or not they are to work 5 days per week (full time).

A timetable is to find employees for a single location, so each location has their own timetable. Yet, the timetables are all linked as they all draw from the same employees, see stack 4.4.2. An area is a grouping of locations. The RNLI groups its locations into areas in the real world so this is reflected here.

4.2.1 Dataset

The RNLI's exact numbers relating to employees, beaches, offices etc. are not freely available online, therefore estimations have been made in creating data for testing. The number of employees needed at each location varies greatly, commonly between 2 and 12. Within the RNLI,

area managers are employed to ensure there are at least the minimum number of employees. Meaning we can assume that there will always be enough employees available. This lead to the decision to make the number of employees required at each location stay consistently at 6, the number of locations per area 5 and the number of employees per area 60. In application of this scheduling solution, each area manager would upload the information about their own area and employees in order to produce specific timetables. But here there is no advantage of leaving all these numbers at consistent values as making sure there are enough employees is not within the scope of this solution, so the solution would still work on the real dataset.

The aim for our final solution is to be able to repeatedly produce a viable schedule for a dataset of 3000 employees, 250 locations and 60 areas.

For use in preliminary testing, three different miniature datasets were generated through online data generating tools. For both these and the full size dataset: employees ranks were randomly generated between 1-3 and rank 4 employees were less likely to be generated as in real life this is the highest level and fewer employees reach this level. Employee qualifications were generated with weights, to reflect how likely it is for real life employees to hold them. There was a 0.3% chance for an employee to hold a boat driving qualification, 0.6 for boat crewman qualification and 0.5 for jet ski user qualification. Over the large dataset these numbers represent accurately how many employees hold these qualifications, but over the small dataset there is more chance of variation. Due to this, one small generated dataset could have been impossible to solve or at least not represent the real problem, so three were generated. The full size dataset was produced in the same way but to fit the full scale.

4.2.2 Constraints

Constraints are modelled by counting constraint violations (CV) within each area, each violation is counted by simply searching through the timetables and gathering required information to decide violations. For example, check each day who is working in each area and if anyone is working twice on the same day, a constraint violation is found. For rank and qualification requirements, count how many of each are working on a given day, if the minimum number required by that location is not reached then a violation is found for each number below the required amount. For hard constraints, each violation adds one to the violation count.

Soft constraints follow the same method but are weighted with differing values between 0 and 1 to prioritise the more important optimisation over the less important. Soft constraint optimisation is separate from generating a timetable that meets all hard constraints, so 1 rep-

resents the most important soft constraint even though hard constraints are far more important and also worth 1.

1. Weight = 1 - Employees working in the area closest to them is weighted at 1 as the most important soft constraint.

2. Weight = 0.8 - Part time employees working 2 or 3 days per week, it is best if work is spread evenly between part time employees rather than one employee working a near full time schedule and another working 1 or no days per week.

3. Weight = 0.6 - Full time employees having their two days off per week consecutively, employees often prefer this weekend-style work schedule.

4. Weight = 0.4 - Full time employees working at their choice of location. Full time employees can choose their favourite location within the area they primarily work and timetable managers aim to put them there when they can.

4.3 Preliminary Algorithm Testing

To gauge the most appropriate technique to generate the workplace timetables as efficiently as possible, different methods are tested on the three small dataset. This section describes the methods used and discusses results. Each dataset contains one area, which means 5 locations and 60 employees need to be scheduled into 7 days. Only hard constraints are considered here.

4.3.1 Method Implementations

4.3.1.1 GA

Firstly, a simple genetic algorithm: 50 instances of the dataset's area are initialised randomly, each of the areas' locations are populated randomly from the employees available. Upon each iteration of the GA, the *likelihood* of an instance to be chosen for reproduction is directly proportional to the instance's *fitness*.

$$Fitness = \frac{1}{\sum CVs} \tag{4.1}$$

$$Likelihood = \frac{Fitness}{\sum_{1}^{50} Fitness'} \tag{4.2}$$

Offspring are generated by taking half the days from each parent and combining them together to create a new timetable. 4 from the one with highest fitness and 3 from the other. There is also a 1% chance at each day slot for an employee to be switched with another random employee, this gives the mutation required for evolution. An issue came to mind upon implementing this algorithm in that some constraints can be solved on a per-day basis but some involve the days employees work throughout the entire week, combining days between two timetables will sometimes cause offspring timetables to be less viable than their parents. To combat this, after a number of iterations, parents can only replace their children if they yield a higher fitness. The aim is for the algorithm to find the optimal solutions for the per-day constraints. Then when offspring cannot replace parents without being more optimal, mutations make the changes required to reach a final solution that satisfies the weekly effecting constraints.

4.3.1.2 MCHC

Secondly, min-conflicts hill-climbing is implemented: data initialisation follows that of the genetic algorithm, randomly populated, except this time there is only one instance of the area. Firstly, a random day slot with a violation is chosen. A violation can be caused by having not enough employees with a certain attribute, in which case a day slot with an employee without that attribute is chosen as the violation position. It can also be caused by an employee being in a position they can't be, i.e. they're double booked, in which case that employees position becomes the violation position. From this violation position, many switches from this position are tested concurrently, and each time a switch is made the new timetable is evaluated using the min-conflicts heurisitic (MCH). Much like the fitness function of the GA, the MCH counts the constraint violations in a given timetable. The timetable with lowest heuristic is then taken as the new timetable and the process is repeated until no violations remain.

4.3.1.3 MA

The last algorithm in the preliminary tests is a memetic algorithm: here we combined the GA process and an MCHC local search to find a solution. After each evolution generation, each area applies MCHC on a limited portion of its timetables.

4.3.1.4 Results

Time in milliseconds (ms)

Each algorithm's average across all three datasets:

	Genetic Algorithm			MCHC			Memetic Algorithm		
Dataset	1	2	3	1	2	3	1	2	3
	5389	5729	4743	2652	3013	2706	3529	3639	4552
	5923	5409	5032	3054	2782	2583	3312	4704	4204
	5039	6203	5232	2849	3410	2738	4150	3542	5376
	5283	7520	4839	1910	1812	1917	3823	3382	4839
	4920	6093	6293	3102	1904	2478	3649	4337	5930
	5982	6302	5415	1709	3326	2653	3494	3619	5398
	5739	5847	4523	1840	2203	1727	4203	3813	4783
	5489	6430	4920	2501	3492	2605	3250	3520	5382

Table 4.1: Execution time for algorithms

- Genetic Algorithm - 5595 s^{-3}
- Min-Conflicts Hill-Climbing - 2540 s^{-3}
- Memetic Algorithm - 4184 s^{-3}

As shown, MCHC outperformed the memetic algorithm by 160% efficiency. In educational timetabling, nearly all scheduling is for individuals who have no requirements themselves other than not being double booked. In that case, it makes sense that evolutionary algorithms would perform much better as combining parents is less likely to bring negative changes to the timetable. Here, each new generation is likely to reverse the progress of its predecessors, meaning the evolutionary algorithms under-performed compared to our expectations. For this reason MCHC will be the chosen method for designing our full scale timetabling application.

4.4 Solution

The small-scale solution in the preliminary testing does not exactly scale up to the full size dataset, so this section describes the method used for timetabling the full dataset and the differences between this full size version and the preliminary MCHC algorithm.

4.4.1 Initialisation

In the RNLI, employees are hired by the staff of a certain area, but are often asked to work other areas around them and even sometimes further. Here a soft constraint is employees should work as close to the area they were hired by as possible and if not, *in* the area. If the dataset was

initialised randomly and a generated timetable placed employees all around the country, then there would be a very great task for the soft constraint optimiser to re-schedule them to work as close to their area as possible. Our solution uses a more guided initialisation approach. You must first know that each area's timetables are generated sequentially, which allows for all employees to be initialised within the timetables of their primary area. This massively reduces the workload for the optimising algorithm as the given timetables are much closer to a final solution.

4.4.2 Stack

Employees only working in the areas closest to them gives a problem that is impossible to solve, hence why in real life employees are asked to work across many areas. So employees have to be able to be asked to work different areas. In our solution, when generating each area's timetables, after they are complete, all employees free on each day are added to one of 7 stacks. One for each day of the week. These stacks are then passed to the next area to be added to their timetables if they don't have the capabilities to fill all requirements with the employees initialised in the area. This next area's free employees are then added to the stacks. This is repeated for all areas.

Using a stack to pass free employees means that each area will look at the area surrounding them in order of distance. Once a full loop of all areas is finished the first area is checked again with the free employees. Our solution then searches backwards to allow employees to be passed in the opposite direction, this part is done in minimal time as the timetables are usually already near-complete if not complete. This method combined with the initialisation method ends with a generated timetable that has already fulfilled the soft constraint of employees working close to their area as best as possible. The soft constraint is still implemented in the optimisation phase in case employees were taken from far away in the forward-moving section of the algorithm when there were employees closer to the area in the areas after that could have been taken.

4.4.3 Min-Conflicts Heurisitic (MCH)

The heuristic used in our solution follows that of a MCH. [27] At every step it aims to minimise the sum of all constraint violations.

```
stack[] employeesFree
For areas:
    For locations in area:
        while violationCount != 0:
            violationPosition = getViolationPosition()
            ttcopy = swap(violationPosition, newPosition)
            if heuristicMeasure(ttcopy) < heuristicMeasure(ttoriginal)
               ttoriginal = ttcopy
    while areaviolationCount != 0:
        violationPosition = getAreaViolationPosition()
        areaCopy = swap(violationPosition)
        if heuristicMeasure(areaCopy) < heuristicMeasure(area)
           area = areaCopy
    employeesFree.add(getFreeEmployees())
```

Figure 4.1: Pseudo Code for final solution

4.4.4 Description

Overview of the timetable generating portion of the application is shown above 4.4.4. The program attempts to solve the location-specific constraints such as minimum number of employees with boat-driver qualification, for each location sequentially within the area. It then tries to solve the constraints that are effected across all locations such as employees working more than 5 days per week. As described, employees can work across multiple locations so information from one location does not tell whether this constraint is violated. In the real application the entire code is repeated again in reverse order of areas with a new stack to pass employees from areas in the opposite direction. Also, "newPosition" references a function that can return the position of an employee within the timetable, in the stack, or an employee from the list of employees who are free that day.

Soft constraints are then dealt with separately. There exists an optimise function that aims to optimise for soft constraints. It follows the same pattern as hard constraint optimising except the violationPosition finds soft constraints. A second heuristic measure is also taken to figure whether a change will bring a reduction to soft constraint violations, this heuristic measure includes the weightings from 4.2.2. Optimising for soft constraints is also only able to incur changes if there are no hard constraints affected.

Look-forward heuristics were initially implemented with hopes of improving efficiency but our solution ended up taking longer to produce a solution, this may have been due to a whole new area needing to be produced for each change because of the (MCH) measuring areas as a whole. This has to be weighed against the time it takes for random changes to find the best position. In our case this does not outweigh the previous, therefore LFHs were removed.

As mentioned in 2.3.2: some timetables have to be changed for reasons that cannot be fore-

Test number	CPU Time (s)
1	51.02
2	120.63
3	115.32
4	48.37
5	73.82
6	57.79
7	95.03
8	82.42
9	112.27
10	56.62

Table 4.2: Execution time for producing all timetables

seen or therefore implemented in an application. To allow for this, timetables can be manually edited, giving users the capability to switch the employee in each day slot.

4.5 Results

Results of the performance tests vary greatly depending on the initialisation of the timetables. Tam, V. [30] tested a MCHC algorithm on 20 different timetabling problems and found results between 9 and 1600 seconds. Our solution is tested on the same problem repeatedly but still shows substantial variation through each test. It should be noted that that our program was executed on an Intel Core 2.90GHz machine using Ubuntu 20.04. Table 4.1 gives the results from 10 executions.

We see a variation of 72.26 seconds with a maximum of 120.63 seconds. Assuming these 10 tests are a fair representation of all runs of the application, the results are quite promising. After each of these tests, soft constraint optimisation is executed. These executions perform surprisingly well, every execution manages to reduce soft constraint count from 500-1000 to 0 in a few seconds. The workload of minimising soft constraints for our workplace problem is far less than that of a standard educational scheduling problem. Solving for hard and soft constraints is usually performed simultaneously but here we chose to split them for analysis.

For the RNLI's timetabling efforts, 60 people are employed to spend 3 of their 5 scheduled days per week managing theses timetables. Some of their work does however include interaction with employees to make changes to their proposed timetables, i.e. an employee needing a day off. Combining these employees' efforts means a total of 1440 hours per week are spent managing timetables. For a worst case scenario, we can assume 3 minutes are spent on generat-

ing and optimising a timetable with our solution. We can also assume half the employee hours are spent managing changes that will still have to be made with our solution. This means to generate and manage timetables with our solution takes **50%** less time when compared against doing the same without our solution. Effectively saving 720 hours of managers' time per week.

If a user wishes to re-run the timetable generation, the application will re-initialise all timetables and start from fresh. This is because the algorithm will not make any changes to an already solved timetable. Users being able to switch employees by hand as discussed in section 4.4.4 consoles this issue.

Chapter 5

Conclusions and Future Work

In this document we have proposed and implemented a scheduling system to generate timetables for a large-scale workplace timetabling problem. Our proposal includes a min-conflicts hill-climbing search algorithm that within a few minutes, is able to produce 300 7x6 timetables from 3000 employees. Each of which meets all hard and soft constraints of the locations and the employees used. For the RNLI, a huge efficiency increase could be found when utilising our solution.

5.1 Future Work

Our proposed solution has been designed to allow for one user to manage timetables for all areas of the UK, meaning that same user would have to deal with all employees who need to request a change or a day off somewhere. For 3000 employees this would be a huge manual task. Initially, this system could be used for each area locally. In that case, switches between areas would have to be dealt with manually as they are now. Eventually however, a solution that allowed employees to create accounts within the system and request days off would be more optimal. This would also allow employees to receive personalised versions of the timetable that only include the days and locations they are working along with who they are working with. And receive notifications about changes in the timetable that affect them.

Bibliography

[1] E. K. Burke, J. P. Newall, and R. F. Weare, "A memetic algorithm for university exam timetabling," pp. 241–250, 1996.

[2] S. N. Jat and S. Yang, "A memetic algorithm for the university course timetabling problem," vol. 1, pp. 427–433, 2008.

[3] H. Babaei, J. Karimpour, and A. Hadidi, "A survey of approaches for university course timetabling problem," *Computers Industrial Engineering*, vol. 86, pp. 43–59, 2015, applications of Computational Intelligence and Fuzzy Logic to Manufacturing and Service Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835214003714

[4] B. M. et al, "Setting the research agenda in automated timetabling: The second international timetabling competition," 2009.

[5] A. Schaerf, "A survey of automated timetabling," pp. 87–127, 1999.

[6] R. P. Badoni. (2014) A new approach for university timetabling problems.

[7] K. Murray, T. Müller, and H. Rudová, "Modeling and solution of a complex university course timetabling problem," pp. 189–209, 2007.

[8] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan, 1975.

[9] J. Arabas, Bartnik, and K. Opara, "Dmea — an algorithm that combines differential mutation with the fitness proportionate selection," pp. 1–8, 2011.

[10] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.

[11] L. Paquete and C. Fonseca, "A study of examination timetabling with multiobjective evolutionary algorithms," 2001.

[12] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[13] S. L. Kosakovsky Pond, D. Posada, M. B. Gravenor, C. H. Woelk, and S. D. Frost, "GARD: a genetic algorithm for recombination detection," vol. 22, no. 24, pp. 3096–3098, 11 2006.

[14] T. Wong, P. Côté, and P. Gely, "Final exam timetabling: a practical approach," *IEEE CCECE2002. Canadian Conference on Electrical and Computer Engineering. Conference Proceedings (Cat. No.02CH37373)*, vol. 2, pp. 726–731 vol.2, 2002.

[15] S. Chu and H. Fang, "Genetic algorithms vs. tabu search in timetable scheduling," *1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No.99TH8410)*, pp. 492–495, 1999.

[16] E. Özcan and E. Ersoy, "Final exam scheduler - fes," *2005 IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1356–1363 Vol. 2, 2005.

[17] J.-D. Wei, *Approaches to the Travelling Salesman Problem Using Evolutionary Computing Algorithms*, 09 2008.

[18] F. Leighton, "A graph colouring algorithm for large scheduling problems," 1979.

[19] E. K. Burke, D. G. Elliman, and R. Weare, "A university timetabling system based on graph colouring and constraint manipulation," *Journal of Research on Computing in Education*, vol. 27, no. 1, pp. 1–18, 1994. [Online]. Available: https://doi.org/10.1080/08886504.1994.10782112

[20] W. Erben, "A grouping genetic algorithm for graph colouring and exam timetabling," pp. 132–156, 2001.

[21] D. de Werra, "An introduction to timetabling," *European Journal of Operational Research*, vol. 19, no. 2, pp. 151–162, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221785901675

[22] C. Gotlieb, "The construction of class-teacher timetables," vol. 62, pp. 73–77, 1963.

[23] M. Dorigo, "Optimization, learning and natural algorithms," *Ph. D. Thesis, Politecnico di Milano*, 1992.

[24] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 04 1999. [Online]. Available: https://doi.org/10.1162/106454699568728

[25] M. Bessedik, R. Laib, A. Boulmerka, and H. Drias, "Ant colony system for graph coloring problem," pp. 786– 791, 12 2005.

[26] M. Eley, "Some experiments with ant colony algorithms for the exam timetabling problem," vol. 4150, pp. 492–499, 09 2006.

[27] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1, pp. 161–205, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000437029290007K

[28] S. Minton. (1992) Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems.

[29] H.-M. Adorf and M. D. Johnston, "A discrete stochastic neural network algorithm for constraint satisfaction problems," pp. 917–924, 1990.

[30] V. Tam and D. Ting, "Combining the min-conflicts and look-forward heuristics to effectively solve a set of hard university timetabling problems," pp. 492–496, 2003.