UNIVERSITY OF CAPE TOWN

MASTER'S DISSERTATION

Examination timetabling at the University of Cape Town: a tabu search approach to automation

Author:
Ebrahim Steenkamp

Supervisor:
Dr Rosephine Georgina Rakotonirainy

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science

in the

Department of Statistical Sciences
Faculty of Science

September 2, 2022

Declaration

I, Ebrahim Steenkamp, declare that this dissertation titled, "Examination timetabling at the University of Cape Town: a tabu search approach to automation", which is submitted in partial fulfillment of the requirements for the Degree of Master of Science, represents my own work except where due acknowledgement have been made. I further declared that it has not been previously included in a thesis, dissertation, or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Signed: _____

Date: September 2, 2022

Abstract of dissertation entitled

Examination timetabling at the University of Cape Town: a tabu search approach to automation

Submitted by

Ebrahim Steenkamp

for the degree of Master of Science

at The University of Cape Town

in September, 2022

With the rise of schedules and scheduling problems, solutions proposed in literature have expanded yet the disconnect between research and reality remains. The University of Cape Town's (UCT) Examinations Office currently produces their schedules manually with software relegated to error-checking status. While they have requested automation, this study is the first attempt to integrate optimisation techniques into the examination timetabling process. *Tabu search* and *Nelder-Mead* methodologies were tested on the UCT November 2014 examination timetabling data with tabu search proving to be more effective, capable of producing feasible solutions from randomised initial solutions. To make this research more accessible, a user-friendly app was developed which showcased the optimisation techniques in a more digestible format. The app includes data cleaning specific to UCT's data management system and was presented to the UCT Examinations Office where they expressed support for further development: in its current form, the app would be used as a secondary tool after an initial solution has been manually obtained.

Acknowledgements

I would like to thank and acknowledge...

The University of Cape Town's (UCT) Department of Statistical Sciences, more specifically Dr Rosephine Georgina Rakotonirainy. From inception to completion, Georgina has exceeded my expectations of a supervisor and despite the challenges 2020-2021 brought, Georgina was always available - with swift replies. The improvements, both I and, this dissertation have undergone because of Georgina's guidance cannot be understated.

The UCT Examinations Office for making this a reality, *literally*. Had they not made the time to meet with me and discuss this research, I would have had to choose an entirely different topic.

My family and friends. Without their help and understanding, the pressures of Masters may have become insurmountable.

The National Research Foundation (NRF), the South African Statistical Association (SASA), and UCT for their financial assistance towards this research. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF, SASA or UCT.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

UCT	**U**niversity of **C**ape **T**own
GUI	**G**raphical **U**ser **I**nterface
NP	**N**ondeterministic **P**olynomial
GP	**G**oal **P**rogramming
CSP	**C**onstraint **S**atisfaction **P**roblem
LP	**L**inear **P**rogramming
CP	**C**onstraint **P**rogramming
HC	**H**ill **C**limbing
SA	**S**imulated **A**nnealing
TS	**T**abu **S**earch
PSO	**P**article **S**warm **O**ptimisation

Chapter 1

Introduction

1.1 Background

For the sake of optimisation and efficiency, schedules have become the standard in today's society. Where schedules are needed, it begs the question of whether an optimal schedule exists and what it looks like. This type of inquisition fuels scientific research where the interest lies in both the theoretical aspects and real-world applications of scheduling problems (Chandrasekharan and Wauters [16]). Regarding scheduling problems, timetabling is a subset that has garnered sufficient interest such that international conferences are held for it. The Practice and Theory of Automated Timetabling (PATAT) is one such series of conferences, held biennially since 1995. One of the most widely studied timetabling problems is educational in nature, namely university course timetabling (Qu et al. [53]).

A simple timetabling problem can be thought of as assigning $R = \{r_1, r_2, ...\}$ resources and $T = \{t_1, t_2, ...\}$ times to $M = \{m_1, m_2, ...\}$ meetings subject to $C = \{c_1, c_2, ...\}$ constraints (Burke, Kingston, and De Werra [13]). If M is the examinations to be scheduled, R the available venues, and T the time slots available during the examination period, then the timetabling problem becomes one of examination timetabling and venue allocation. The constraints associated can be classified into *hard* or *soft* constraints. Hard constraints are the rules that cannot be violated by the timetable. On the other hand, soft constraints are not necessary, but enforced, to improve the quality of the timetable. An example of a hard constraint is "no student can have two of their exams scheduled at the same time" whereas a soft constraint would be "no student should have to write three exams within a 24-hour day".

With timetables becoming increasingly complex, there has been more of a shift from manually completing timetables to full automation of the process. Figure 1.1 reflects this sentiment when accessing Digital Science's Dimensions software and searching the publications with terms "timetabling AND university AND automation". Technology innovation has accelerated this shift by allowing more computationally-intensive approaches, and, while there have been a growing number of ever-evolving

algorithms (Kingston [44]), many of them have not been tested on realistic data (Torres-Ovalle et al. [67]). Benchmark datasets exist but these are far removed from reality and the intricacies it involves. While certain constraints are universal for specific timetabling problems, real-world problems generally have some unique constraints which do not allow for a "plug-and-play" approach to algorithm implementation and thus make finding an appropriate solution difficult (Vrielink et al. [69]).

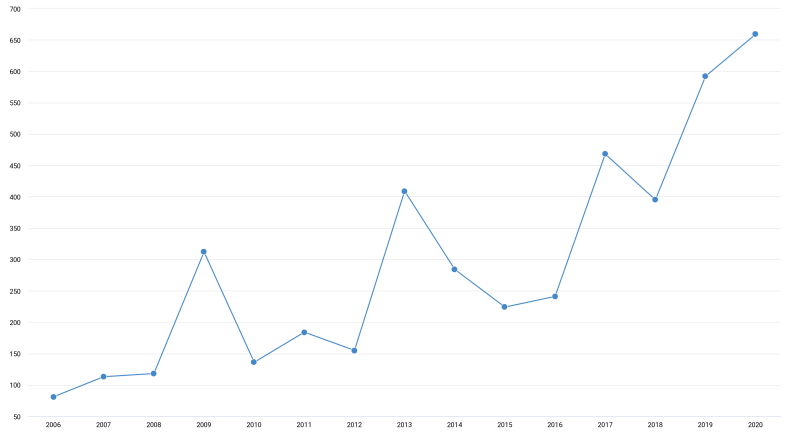

Figure 1.1: Publications related to the terms "timetabling AND university AND automation" from 2006-2020. Sourced from [29].

This rings true for the University of Cape Town (UCT). Currently, their examination timetabling is done manually - with software only being utilised to track constraint violations. The University of Cape Town's problem is complicated by their philosophy on a student's academic career: students are allowed to take additional courses (electives) as long as they meet its prerequisites; as opposed to prescribing a course plan for each year of a degree as some other universities do. This degree-flexibility compounds the difficulty in creating an exam timetable void of clashes. Nevertheless, the UCT Examinations Office (hereafter, *Examinations Office*) administers this task and accomplishes it in three phases. Using student records, they construct the first provisional timetable where students and lecturers can then give feedback on its viability as issues may arise which cannot be foreseen by the Examinations Office. For example, a lecturer not being present at the university for the given day(s) in the examination period. Based on their input, a second provisional timetable is created. Feedback is then sought again, and a final timetable is released for the examination period based on the response.

The main aim of this study is to streamline the construction of the first provisional timetable by developing an algorithm that will automate the process. The Examinations Office have expressed that they are generally able to create a first provisional timetable without violations, increasing the likelihood of successful automation. This study will not explore algorithms that will address the subsequent timetables in the process. Perturbation theory is recommended for those interested.

1.2 Objectives of the thesis

The main objectives of this study are:

1. to automate the first phase in UCT's examination timetabling process and evaluate its applicability,
2. to present the automation in a simple and easy-to-use manner such that users will not be required to have expertise in order to operate, and
3. to begin discourse on automation of UCT's examination timetabling.

1.3 Thesis organisation

In this chapter we have given a brief introduction into the nature of this research and what it aims to achieve. The next chapter gives an overview of methods employed in timetabling literature. Chapter three contains an exploratory analysis of the data received from the Examinations Office and subsets the data into what will be used throughout the remainder of the paper. Chapters four and five describe the timetabling problem formally and discuss the proposed algorithms, respectively. Following this, the results of the automation are presented and reviewed (Chapter six). In an attempt to make the algorithm and its output more usable, a graphical user interface (GUI) was created as a means of operating the algorithm. Chapter seven details the creation of this GUI and displays images of what users will see when operating the programme. Though GUIs tend to be the focal point in software applications rather than the underlying algorithms (Vrielink et al. [69]), the algorithm and GUI demonstrated herein are equally valuable. Chapter eight closes the paper through summary and discussion, giving recommendations and propositions for future work.

Chapter 2

Literature Review

A nondeterministic polynomial (NP) problem is a problem where it takes polynomial time to check if a solution exists but longer to find a solution. If a problem is NP and all other NP problems can be reduced to it in polynomial time, the problem is known as NP-complete (Britannica, The Editors of Encyclopaedia [7]). Currently, no efficient algorithms exist for solving NP-complete problems, and whether or not the possibility exists is a Millenium Prize Problem. In 1995, Cooper and Kingston [22] proved that, in practice, timetabling is an NP-complete problem. This explains the abundance of differing approaches which Burke, Bykov, and Petrovic [8] classified into four groups: *graph-based sequential techniques*, *decomposition/clustering techniques*, *constraint-based techniques*, and *meta-heuristics*. As there have been many new developments since then (Qu et al. [53]), the number of groups have expanded. Some of these techniques are explored in this chapter.

2.1 Graph-Based Sequential Techniques

A basic timetabling problem can be represented as a vertex colouring problem where each exam is represented as a vertex and an edge exists between two exams if and only if at least one student is writing both exams. Figure 2.1 demonstrates how solving a vertex colouring problem can be equivalent to solving a timetabling problem. The aim is to colour the vertices in the minimum number of colours where vertices that share an edge are differently coloured. Such a number, often denoted $\chi(G)$, is called the chromatic number of the graph G and relates to the minimum number of time slots needed to schedule exams without clashes.

If we denote the number of edges exam i has as d_i, and assume without loss of generality that

$$d_1 \geq d_2 \geq \cdots \geq d_n, \tag{2.1}$$

then it can be proved that $\chi(G) \leq d_1 + 1$ by Vizing's theorem (Diestel, Schrijver, and Seymour [28]). Welsh and Powell [70] proved that the number of colours needed is at

most $\alpha(G)$ where

$$\chi(G) \leq \alpha(G) \leq \max_i \ \min \ (i, d_i + 1) \tag{2.2}$$

The significance of (2.2) is that the upper bound is easily computed since d_i is at most the maximum number of students that are registered for exam i. If the university can accommodate the number of time slots suggested, the authors created a simple algorithm that will assign each exam to one of those time slots.

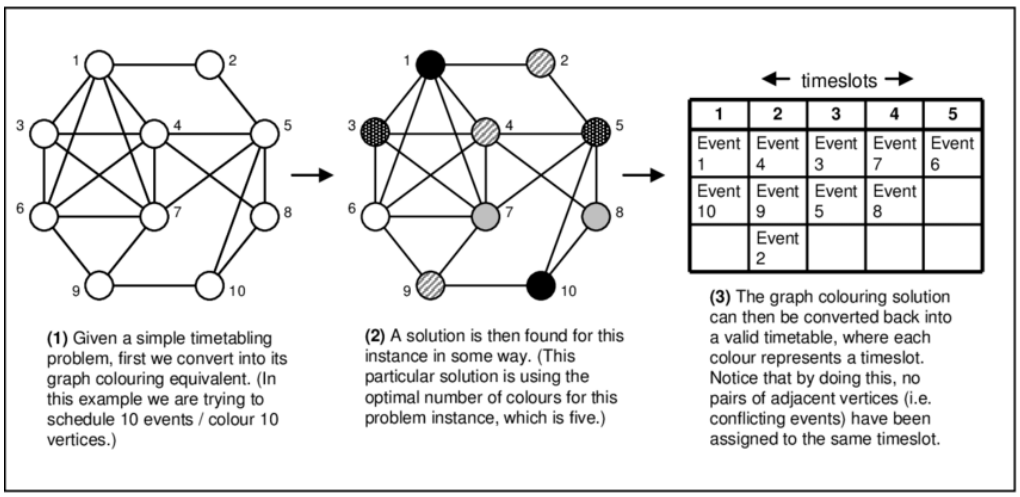

Figure 2.1: An illustrated example showcasing the conversion of a graph colouring problem to a timetable solution. Only the clashing of events, in this case exams, are considered in this example. Two events with the same colour are clashing if there exists an edge linking the two vertices (credit: Lewis, Paechter, and Rossi-Doria [46]).

For some cases, the above approach may not be applicable. For example, in university timetabling, the algorithm may suggest an upper bound of time slots well into the thousands, hence the length of the exam period would be infeasible. An alternative approach that has been considered in the timetabling literature is a maximum clique ordering method. A maximum clique is the largest set of vertices in a graph such that every two distinct vertices share an edge (adjacent vertices). Once an ordering is established, vertices are coloured accordingly.

Arguably one of the most famous clique strategies is that of Kempe chains which were introduced by Alfred Bray Kempe [42] in his attempt to prove the four-colour problem. A Kempe chain works by selecting two adjacent vertices. To ensure proper colouring, these two vertices have to be different colours, say, c_1 and c_2. Create a chain by finding all adjacent vertices, to either coloured vertex, that can be coloured c_1 or c_2. This colouring process is continued, i.e. colouring adjacent vertices to c_1- or c_2-coloured vertices until no other vertex can be added (maximal chain). This is known as a (c_1,c_2)-Kempe chain. The Kempe chain process depicted in Figure 2.2 can be applied as an extension of the second step in Figure 2.1. With an initial colouring,

a Kempe chain can be used to create an alternative colouring without affecting the colour feasibility of vertices outside the chain. In practical terms, this reshuffling may be more suitable than the original solution.

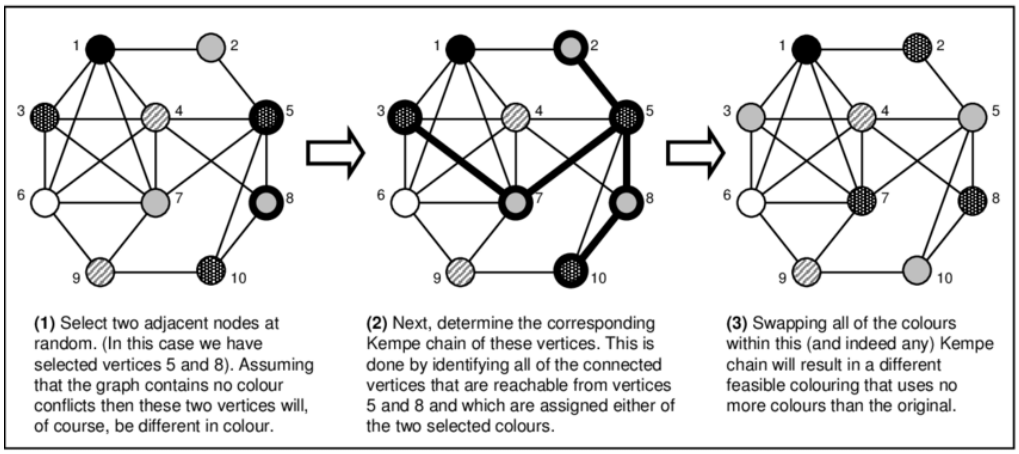

Figure 2.2: Example of a Kempe chain neighbourhood operator (credit: Lewis, Paechter, and Rossi-Doria [46]).

Another widely used strategy is that of saturation degree ordering defined by Daniel Brélaz [6]. The saturation degree of a vertex is the number of adjacent differently coloured vertices. Coleman and Moré [21] adapted this strategy by using incidence (number of adjacent coloured vertices) instead of saturation. These strategies are alternate colouring techniques to Kempe chains, however, graph theory techniques are more commonly used as components of other techniques - to add to their robustness.

2.2 Decomposition/Clustering Techniques

Decomposition techniques involve dividing a given problem into smaller subproblems, which are optimised using appropriate algorithms. These techniques attempt to find near-optimal solution by solving easier problems to reduce complexity. Despite being one of the simpler techniques, very few papers implement decomposition techniques in timetabling literature (Qu and Burke [52]). Difficulty is faced when attempting to cluster timetabling problems as composite solutions may lose optimality, or be infeasible (Abdul Rahman et al. [1]), and certain constraints may not allow clustering as they cannot be evaluated on a decomposed problem (Qu et al. [53]). This inability to reach optimality will almost surely result in dominable solutions. Nevertheless, the improvement gained from an optimal solution may not justify the increased computational time required. In addition, decision-makers may only be interested in a feasible solution or have a limited time frame in which a solution must be obtained.

In a paper by Burke and Newall [11], the authors decomposed a timetable problem into subproblems which their population-based algorithm could more effectively handle. They noted that the execution time had considerably reduced and the quality of solutions improved. However, if decomposition is not an option, execution time could still be reduced if the algorithm allows parallelisation. The lack of timetable parallelisation is due to sequential constructive algorithms being one of the most popular methods (June et al. [40]). As the name suggests, these type of algorithms create solutions successionally and thus do not contain independent chunks which can be parallelised. Despite the difficulty that parallelisation entails, the payoff may be substantial. Wu [76] used parallelisation in their course timetabling system, comparing a single computer to five, and found that performance was improved by a factor of 12.47. The parallelisation speedup was superlinear in that the improvement factor was greater than the number of computers used.

2.3 Multi-Criteria Techniques

Researchers in the field tend to focus solely on a singular objective function when dealing with examination timetabling optimisation problems. There are, however, multiple stakeholders that need to be considered, namely: administration, departments, and students (Romero [56]). By its nature, timetabling problems have to take into account various criteria when finding a solution, yet, when constructing objective functions, the convention is to weight the criteria into a single value to be optimised (Silva, Burke, and Petrovic [63]). The solution is highly dependent on the weights chosen and therefore the choice of weights needs appropriate reasoning. With students preferring long breaks between exams, for example, their considerations are almost always treated as soft constraints. This is reflected in the objective function with the associated weights being lower. However, as the student-as-consumer model becomes more prevalent, the objectives of universities may change and likewise for their objective functions' weights (Clayson and Haley [20]).

The most commonly used multi-criteria technique to handle timetabling problems is the well-known goal programming (GP), characterised by Charnes and Cooper [18]. Using this approach, each criterion, z_k, is associated with a goal value, g_k, for K different criteria. An example of a criterion in the context of a timetabling problem is to obtain a spacious timetable for students. A goal could be that no student has to write exams back-to-back. In principle, each pair of criterion-goal is associated with two deviation variables, d_k^- and d_k^+, which measure the underachievement or overachievement of the goal. Using the Archimedean metric, the objective is to minimise the sum of the weighted deviations or violation of the various criteria. That

is, for K different criteria:

$$\begin{aligned} &\min \sum_{k=1}^{K} (w_k^- d_k^- + w_k^+ d_k^+) \\ &\text{subject to} \\ &z_1 + d_1^- - d_1^+ = g_1 \\ &\dots \\ &z_K + d_K^- - d_K^+ = g_K \end{aligned} \tag{2.3}$$

If the purpose is to produce a more balanced solution then Chebyshev GP is employed, conceptualised by Richard Bailey Flavell [32]. This variant minimises the worst-case scenario for any criteria and therefore is also known as minimax GP as it minimises the maximal deviation for any constraint. The difference between the two variants is the objective function:

$$\min \max_k (w_k^- d_k^- + w_k^+ d_k^+) \tag{2.4}$$

Archimedean and Chebychev GP are considered special cases of preemptive/lexicographic GP (Ignizio [38]), which optimises according to constraint priority - only once the minimum objective function value is found for a certain priority-class of constraints, are the next (lower) priority-class of constraints introduced. For example, if there are two priority classes, H and M, then the formulation of the objective function for H-class criteria will look identical to (2.3) - assume α is the minimum value attained by this function. The formulation of the M-class criteria objective function is:

$$\min \sum_{k \in M} (w_k^- d_k^- + w_k^+ d_k^+) \tag{2.5}$$

The H-class criteria constraints remain and the M-class criteria constraints are now added. There is an additional constraint for each class above the current priority. In this example, there is only one so the addition is:

$$\sum_{k \in H} (w_k^- d_k^- + w_k^+ d_k^+) \leq \alpha \tag{2.6}$$

Preemptive GP splits the solving process into multiple phases, essentially decomposing the problem. This variant should only be pursued when a decision-maker can firmly assign priorities to criteria and this is why setting up a preference for all criteria considered is a critical step when dealing with multi-criteria problems (Salas-Molina et al. [59]). These three major variants all have differing philosophies of GP, and hence have differing underlying distances in their calculations (Jones et al. [39]).

2.4 Constraint-Based Techniques

In 1999, Brailsford, Potts, and Smith [5] published a paper on constraint satisfaction algorithms and their applications. They intended to introduce these methods to the operational research field as these methods were primarily used in artificial intelligence and not widely known among operational researchers. A constraint satisfaction problem consists of a set of variables, with defined domains, and a set of constraints. As the name suggests, if the solution to a constraint satisfaction problem is feasible then the selected variable values satisfy all constraints. This problem definition was not unfamiliar to operational research due to George Bernard Dantzig inventing linear programming (LP) in 1947 (Dantzig and Thapa [25]).

Constraint programming (CP) and LP have many similarities and have been increasingly integrated in research (Van Hentenryck [68]). The main difference between them is how constraints are defined and approached. The constraints in LP are necessarily linear, whereas they can be non-linear in CP. In the case of timetabling, the decision variables take the form of positive integers which results in integer LP being used. CP and integer LP both use search tree methodology but LP algorithms emphasise the objective function and prune suboptimal solutions, whereas, CP algorithms focus on constraints and prune infeasible candidate solutions. Brailsford, Potts, and Smith remarked that if the problem does not allow much pruning then CP is likely to be more efficient. Furthermore, if CP is used in a pure form without adaptation, it is unlikely to be competitive with local search methods.

More than just a tool, CP advanced from applying constraints in logic programming to constraint programming as a language (Rossi, Van Beek, and Walsh [57]). Merlot et al. [48] published a paper where they employed a three-phase algorithm, with constraint programming in the first phase, to solve the University of Melbourne's exam timetabling problem. The optimisation programming language they used, OPL, was created by Pascal Van Hentenryck [68]. The algorithm proved successful as it was superior to the university's current process and obtained the best results reported in literature on several benchmark timetabling problems. As suggested by Brailsford, Potts, and Smith, Merlot et al. achieved this feat by incorporating local search methods in phases two and three to further improve the constraint programming algorithm.

2.5 Local Search-Based Techniques

Local search-based techniques traverse search spaces iteratively by moving from the current solution to a neighbouring solution (Pirlot [51]). These techniques usually consist of general search principles arranged in a general search strategy (meta-algorithims). They are more commonly known as general heuristics or metaheuristics. These techniques also do not guarantee an optimal solution as they search non-systematically and are non-exhaustive (Schaerf and Di Gaspero [61]). That being the

case, the choice of neighbourhood construction is paramount to the effectiveness of these techniques.

The most popular local search-based techniques are *hill climbing* (HC), *simulated annealing* (SA), and *tabu search* (TS) (Schaerf and Di Gaspero [61]). HC is the simplest form of the three, only performing moves that improve (or keep constant) the value of the objective function. The main problem HC faces is converging towards local extrema. SA and TS have differing philosophies on dealing with this. A move in SA can have one of three outcomes: it improves the objective function and is accepted, it worsens the objective function and is rejected, or it worsens the objective function and is accepted based on an iteration-dependent probability function. The probability function used decreases as iterations increase, making it less likely that a worse solution will be accepted. The idea is to avoid converging too soon in hopes to circumvent local extrema. Kirkpatrick, Gelatt, and Vecchi [45], the inventors of SA, were inspired by annealing (metallurgy) and therefore the probability function resembles a cooling schedule. In contrast, Fred Glover [36] presented TS which utilises a memory-based strategy. At each iteration, the best neighbouring solution is chosen irrespective of the change in the objective function. The algorithm will remember previous solutions and forbid moves towards those solutions in the future - the solutions are said to be on the tabu list. The list encourages exploration of the search space by not allowing moves towards previously explored areas. However, as iterations increase, solutions are removed from the list based on the memory length (short/intermediate/long) and can be revisited.

A downside to metaheuristics is that solution optimality cannot be proven and thus there is no obvious stopping criteria. This enforces a more interactive approach to problem-solving as algorithms are more reliant on user input, from parameter specification to solution evaluation, and the timetabling research community recognises this as crucial (Schaerf [60]). Metaheuristics give the possibility of starting from any initial solution. This ability supports interactive work by allowing the opportunity for changes to be made to the algorithm without it breaking. For example, users are given the freedom to adjust the current solution or update the constraints, as UCT may do in the second and final phase of their exam timetabling process. These settings can be used for initialisation and the solving process can continue. Real-life situations are extensions of classical problems and metaheuristics have the advantage of easily integrating problem-specific adjustments into their solving strategies. The requirement of interaction balances automation and experience. Over time, the user's domain expertise grows, and they are able to adjust and adapt the algorithm to further reduce the time required to solve the task (Chahal and De Werra [15]). It should be noted that while metaheuristics are sensitive to their parameter configuration (Huang, Li, and Yao [37]), fine-tuning is only vital when concerned with optimality (Pirlot [51]).

Thompson and Dowsland [66] produced the 1993 Swansea University timetable using SA. The final timetable produced was a substantial improvement over previous years, only having 490 cases of (over 3000) students sitting two exams in two days. Within a minute, their method could produce a feasible timetable. The authors also employed Kempe chains to generate neighbourhood solution and reported that Kempe chain neighbourhoods significantly outperformed SA but did not satisfy the reachability condition: a concept where a path exists between any two vertices. Nevertheless, the authors planned to use Kempe chains in future research as the methods they designed to satisfy the reachability condition either performed poorly or required excessive execution time. White and Xie [71] implemented a four-phase TS algorithm using both short-term and long-term memory at the University of Ottawa. They found that a 34% improvement was attributable to the algorithm's long-term memory by comparing schedules produced with and without this feature. From their research, they found that short-term TS techniques make quick improvements from an initial solution but fail to generate high-quality solutions and that the addition of long-term memory will result in better solutions. Furthermore, they were able to automatically determine an appropriate size for long-term memory.

2.6 Population-Based Algorithms

Similar to local search-based techniques, population-based algorithms are classified under metaheuristics but were largely inspired by nature. They attempt to overcome the issue of local extrema by generating multiple solutions, thereby increasing the exploration of the search space (Cheng et al. [19]). An additional advantage of these approaches is the parallelisability of the solution generation process (Giagkiozis, Purshouse, and Fleming [35]). Having multiple solutions, the Pareto front can be approximated. This set of non-dominated solutions award decision-makers multiple equally-optimal solutions when problem-solving.

Motivated by Charles Darwin's theory of evolution, Lawrence Jerome Fogel devised evolutionary programming in 1960. Around this time, *genetic algorithms*, a subset of evolutionary algorithms with a focus on gene propagation, were ideated by John Henry Holland (De Jong, Fogel, and Schwefel [26]). An alternative subset is differential evolution which was designed by Storn and Price [65] to be a stochastic search method. Farmer, Packard, and Perelson [31] presented an algorithm based on the immune system and, although not by design, the algorithm shares many similarities to Holland's. Substantiating Holland's notion that natural systems provide numerous insights for parallel computation, there are population-based algorithms that mimic species interactions: two of which are *particle swarm optimisation* (PSO), devised by Eberhart and Kennedy (Shi and Eberhart [62]), which mimics flocking of birds and *ant colony optimisation*, formulated by Dorigo and Di Caro [30], which mimics a colony of ants. Hybrid approaches to population-based algorithms also exist. For

example, *memetic algorithms* integrate a local search-based technique on the individual solutions generated by an evolutionary algorithm (Burke, Newall, and Weare [12]).

Of the evolutionary algorithms researched in exam timetabling, genetic algorithms, and memetic algorithms, in particular, have been the most studied (Qu et al. [53]). The simplest genetic algorithms are initialised with multiple solutions. These solutions are known as the population (of solutions) and each solution, called a (parent) chromosome, has an associated objective function value - a fitness value. Through some selection method, two chromosomes are selected and crossover to produce a new (child) chromosome. The crossover operation allows exploration of the search space, whereas mutation is exploitative as they are applied to the child chromosomes. Finally, the child chromosome will replace a chromosome from the initial population and the process repeats. The replacement process can be influenced by fitness values, leaving only the best solutions remaining in the population - directly paralleling the "survival of the fittest" nature of evolution.

Arogundade, Akinwale, and Aweda [3] published a paper wherein they described their use of genetic algorithms to solve a Nigerian university's exam timetabling issues. The university was not efficiently allocating its resources to meet its growing demands; its grid-like system required manual adjustments to create a feasible solution. The entire system was reworked, and the genetic algorithm implemented produced better results while being more cost- and time-efficient. The improved system also allowed larger inputs and more flexibility. Abela [2] showcased a parallelised genetic algorithm on a school timetabling problem. They stated that parallelisation was easily implemented owing to inherent parallelisability. However, their results showed less than significant speedups as a consequence of certain critical paths of the program not being parallelised. In their research, they achieved a sublinear speedup with a peak speedup of 9.2 from 10 processors. Marie-Sainte [47] showed the performance of PSO on data from the King Saud University. Their particular method was limited to 100 exams, but the results revealed positive implementation nonetheless. The University of Technology Malaysia used PSO to model the undergraduate information and communication technology courses and also reported it suitable for timetabling (Foong and Rahim [33]). Their results included a User Acceptance Test which the system scored between 50% and 80% on all their criteria.

2.7 Hyper-heuristics

The term "hyper-heuristic" was first used by Denzinger and Fuchs [27] in artificial intelligence and was independently used by Cowling, Kendall, and Soubeiga [23] to describe "heuristics which choose heuristics" in optimisation. A hyper-heuristic is a high-level approach to problem-solving. At each decision point, a hyper-heuristic selects and applies an appropriate low-level heuristic (Burke et al. [9]). The motivation

is to increase the general applicability of search methodologies.

Hyper-heuristics have largely been assessed on their ability to generate optimal solutions; little research has been done to evaluate their generality. Pillay and Qu [50] published a paper wherein they provided a taxonomy that classifies hyper-heuristics into four levels based on their generality. If a hyper-heuristic can only be applied to a single problem, then it is classified as having a generality level of one or two, based on whether or not it can perform well on multiple benchmarks. Level three is given to a hyper-heuristic if it can be applied to a single domain of multiple problems. The highest level a hyper-heuristic can achieve is when it can be applied across domains. The authors also categorised hyper-heuristics based on the heuristic nature of the search space - whether the hyper-heuristic is selective or generative. A hyper-heuristic is selective if it selects a low-level heuristic and generative if it creates one. A low-level heuristic can be further classified based on whether it is constructive or perturbative, i.e. whether it creates a solution or changes an existing one. In 2010, Burke et al. [9] extended the classification of Pillay and Qu by introducing a new dimension, feedback. Feedback deals with the learning mechanism of the hyper-heuristic and has three learning possibilities: online learning, offline learning or without learning. Online learning is when the hyper-heuristic learns as it solves a problem. Offline learning is when the hyper-heuristic is trained beforehand and without learning is when the hyper-heuristic does not use feedback from the search process. The two dimensions hyper-heuristics are classified across are depicted in Figure 2.3.

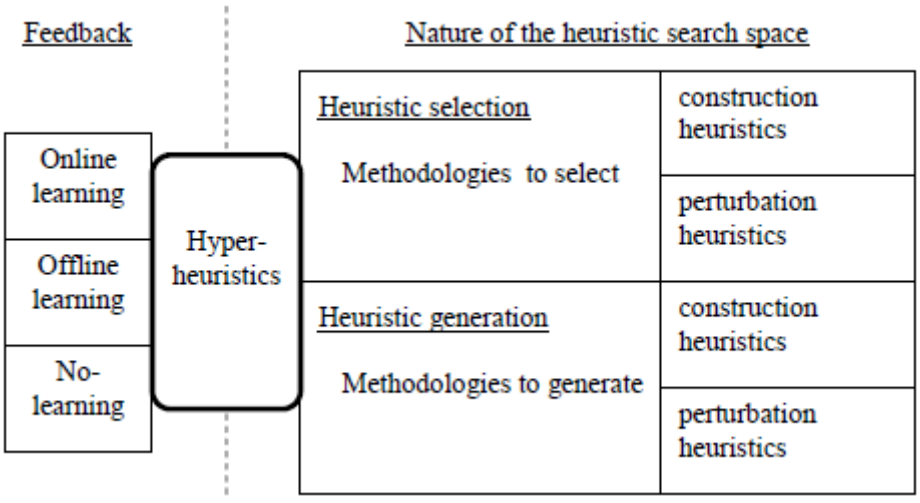

Figure 2.3: A classification of hyper-heuristic approaches, according to two dimensions (i) the nature of the heuristic search space, and (ii) the source of feedback during learning (credit: Burke et al. [9]).

Burke et al. [10] applied a hyper-heuristic approach to solve education timetabling problems. The hyper-heuristic they proposed was classified as selection constructive as it used a tabu search algorithm to select a low-level constructive graph-based heuristic. There were five heuristics to choose from and they each incorporated either online learning or without learning in their feedback. Their results indicated that the hyper-heuristic performed well on all the benchmark course timetabling problems, even obtaining the best-reported result for one benchmark. They tested both a single-stage graph-based hyper-heuristic and its two-stage counterpart and found that the two-stage obtained worse results. They hypothesised that this is due to the limitations of the algorithm's starting points. Experimental results also showed that increasing the number of low-level heuristics would improve efficiency, but at the cost of increasing the search space, and therefore also increasing the computational time.

2.8 Summary

This chapter has reviewed the literature on timetabling, specifically examination timetabling. The purpose is to expose the reader to various methods explored, but also implemented in practice. With UCT having done no automation in this field, their infrastructure is lacking. This paper aims to start building the foundation wherein later iterations can improve upon. Literature makes it clear that many methods can solve this problem and therefore the focus is on selecting one and making it work rather than selecting the most appropriate necessarily.

Chapter 3

Data wrangling

Before defining the problem and its probable solution(s), it's important to delve into the available data as it governs what can be explored and what needs to be assumed. Three files were received from the Examinations Office in connection with the November 2014 exams: student data, venue locations and capacities, and a provisional timetable. The data was obtained from an external human resources management system and required processing. Data cleaning was done in R (R Core Team [54]) and RStudio (RStudio Team [58]) along with the `readxl` (Wickham and Bryan [74]) and `stringr` (Wickham [73]) packages.

3.1 Student data

Students each have a unique ID associated with them. By filtering IDs, this file shows all courses that each student was registered for in 2014. Each course ends in an indicator which identifies the type of course and when it is offered. Only courses that were going to be examined in the November period are of interest, and the rest were removed. Also of interest is whether a student requires special needs. Special needs requirements are equipment-orientated (such as needing a computer) or time-orientated (requiring extra time). The specific requirements each student has is detailed in the document where extra time is recorded as a number of additional minutes per hour the student is allowed. Regardless of the nature, students with special needs must be scheduled in venues separate from those without special needs.

Table 3.1: An example of the student's data after cleaning

ID	Code	Special	Venue	Extra-Time
911069	ACC1012S	0	0	0
928206	MEC2000X	0	1	10
906266	BUS5000W	1	1	15

In Table 3.1, *Special* is a binary variable which denotes if a student has a special need other than just additional time, *Venue* indicates whether the student will write in the venue separate from non-extra-time students, and *Extra-Time* records the number of additional minutes per hour the student is entitled to.

3.2 Venue locations and capacities

All venues available to UCT and their seating capacities are contained herein. Unfortunately, many problems inhibit correct usage of this information:

- There exist multiple venues with 1000 seating capacity which are used as catch-alls when departments, themselves, handle the venue allocation.
- There are *Take home exam* and *Online* venues that cannot be allocated unless some specification is met - not included in any of the files obtained from UCT.
- The file contains out-of-date venues. These result when venues are renamed, such as Jameson Hall to Sarah Baartman Hall. Both these venues exist in this file and, without information on all the renamings that have taken place, duplicates cannot be removed.
- No indication is given to special venues, such as computer labs, nor was there any indication to exams that required special venues.
- The Examinations Office indicated that they had a priority system to venue allocation, however no such priority is shown in the file itself.

For these reasons, venue allocation will not be possible and is not included in the models.

3.3 Provisional timetable

Ideally, prospective scheduling would create this file as it includes a provisional timetabling of the exams. The necessary information from this file is the courses that require scheduling, number of assessments scheduled for those courses and their durations (Table 3.2). The provisional solution will be used for comparative purposes later in this study.

Table 3.2: An example of the extracted information from the provisional timetable

Course	Assessment	Duration (Hours)	Date	Starting time
BUS3043S	Paper 2	2.00	31/10/2014	03:00 PM
CHM5003W	Online exam	1.75	07/11/2014	01:00 PM
CSC3003S	Paper 1	3.17	28/10/2014	10:00 AM

3.4 Joint cleaning

After cleaning each file separately, the files were compared. The *Special* and *Venue* indicators could be removed from the student data as venue allocation would not be possible. Many courses could also be removed from the student data as they were not present in the provisional timetable. A reason for this is that certain courses may simply not have exams and therefore do not require one to be scheduled. Likewise, the provisional timetable included courses that had no students from the student data. A possible reason for this was given by the Examinations Office: certain departments take care of their scheduling themselves as their students would not have courses outside of their department. For example, dance students will have their exams scheduled by their department so the Examinations Office may not have their student data, but the timetable will include their exams as part of the entire UCT exam timetable.

After removing the aforementioned courses and exams, the data was merged. Courses were renamed based on the course and assessment. For courses with multiple assessments, each course-assessment pair was treated as a unique course to make scheduling easier. Using the duration variable from the provisional timetable, the duration for each course-assessment (hereafter, *exam*) was calculated for each student, including extra time if necessary. Table 3.3 showcases the format of the final dataset after the cleaning process has completed. This data is the data that will be used during implementation of the methods.

Table 3.3: An example of the final data after the cleaning process has completed

ID	Exam	Duration (Hours)
911069	ACC1012S-Paper 2	2.00
994706	CSC2002S-Paper 1	3.17
964304	PBL4601S-Paper 1	10.00

Before cleaning, the student data and provisional timetable contained 92940 and 650 observations respectively. After their individual wrangling, these observations dropped to 92936 and 645 observations - virtually no change. Once these datasets were combined however, the resultant dataset only contained 60100 observations. About a third of the data proved unnecessary and could be removed. The new data included 15636 unique students who were required to sit 598 unique exams. On average, each student would have to sit 3.84 exams.

Chapter 4

Problem Formulation

The UCT November 2014 examination period spanned 15 days with an assumed 19 possible starting times for exams from Monday to Thursday and 16 for Friday. The solution created by the Examinations Office started on the hour or at the half-hour. The earliest and latest starting times were inferred from this solution and it was assumed that every half-hour is a possible starting time. This resulted in 276 possible starting times (throughout the week) for the 598 examinations. The examination timetabling problem can be represented as follows:

- E: A set of e examinations to be scheduled E_1, E_2, ..., E_e
- S: A set of s students who will write the e exams S_1, S_2, ..., S_s
- T: A set of t starting times T_1, T_2, ..., T_t
- D: An allocation matrix where D_{mi} denotes the number of full periods student S_m is allocated for exam E_i
- R: A matrix where R_{mi} denotes that student S_m will sit exam E_i. $R_{mi} = 1$ if $D_{mi} > 0$.
- The scheduled examination timetable X where $X_i = k$ denotes that exam E_i is scheduled to start at time T_k

The problem is to assign the set of e exams to the available t starting times such that the costs are minimised. It is noted that the Mara University of Technology had exams of similar durations (120, 150 or 180 minutes) allowing Kendall and Hussin [43] to create time slots long enough for any exam to be scheduled within. This meant that their algorithm dealt with uniform slot durations, which made allocation of exams much easier as they wouldn't have to consider overlaps between exams. This simplification, however, cannot be performed for UCT's examination timetabling problem for two reasons. Firstly, the range of durations for their exams are much larger (from 30 minutes to 8 hours), and, secondly, the students may require extra time. Extra time concession is given to a student on a per hour basis and the most given in 2014 was 30 minutes. This can result in an 8-hour exam needing a 12-hour slot. Forcing exams to be scheduled according to the most time required by a student for that exam would be too restrictive on the scheduling process, thus, exams need to be scheduled on a student level.

4.1 Constraints

Hard constraints are those which are required to be met. In this work, only a single constraint is considered:

- *All exams must be scheduled, and only once*:

$$\mathrm{X}_i \in \{1, 2, ..., t\} \qquad \forall\, i = 1, 2, ..., e \tag{4.1}$$

4.2 Costs

Soft constraints are governed by the problem-holder's policies and procedures, in this case the University of Cape Town. Soft constraints dictate the quality of the timetabling and can be violated as they have related costs. The soft constraints considered are:

- *Conflict*: Ideally, a student should be able to sit all their exams at the scheduled time. However, this cannot always be guaranteed. Let $\delta_{mij} = 1$ if student S_m cannot sit both exams E_i and E_j and let $\alpha_{ij} = 1$ if exam E_i is scheduled after exam E_j. The total number of clashes/conflicts ($\mathcal{Z}_c$) within a timetable X is therefore:

$$\mathcal{Z}_c = \sum_{m=1}^{s} \sum_{i=1}^{e-1} \sum_{j=1+1}^{e} \delta_{mij}$$

subject to

$$(1 - \alpha_{ij})(1 - \delta_{mij})(\mathrm{X}_i + \mathrm{D}_{mi}) < \mathrm{X}_j$$
$$\alpha_{ij}(1 - \delta_{mij})(\mathrm{X}_j + \mathrm{D}_{mj}) < \mathrm{X}_i$$
$$(1 - \alpha_{ij})\mathrm{X}_i < \mathrm{X}_j$$
$$\delta_{mij} \in \{0, 1\}$$
$$\alpha_{ij} \in \{0, 1\}$$

  D_{mi} does not necessarily equal D_{mj} therefore two equations are needed. Here, α_{ij} is used to nullify an equation based on the timeline of events.

- *Proximity*: A cost $\mathcal{P}_{ij}$ is given whenever a student has to sit two exams E_i and E_j scheduled n periods apart. If $\mathcal{P} = 1.001^{276-n}$, then the total proximity cost ($\mathcal{Z}_p$) is:

$$\mathcal{Z}_p = \sum_{m=1}^{s} \sum_{i=1}^{e-1} \sum_{j=i+1}^{e} \mathrm{R}_{mi}\mathrm{R}_{mj}\mathcal{P}_{ij}$$

subject to

$$|\mathrm{X}_i - \mathrm{X}_j| = n_{ij}$$
$$\mathcal{P}_{ij} = 1.001^{276-n_{ij}}$$

The *proximity* cost decreases exponentially as the number of periods between exams increase. This forces solutions to prefer schedules that have large gaps in timetables from the student's perspective. The constant of 1.001 is used to ensure the cost is relatively small even for large proximities.

4.3 Objective Function

The objective function consists of minimising the total *conflict* cost ($\mathcal{Z}_c$) and total *proximity* cost ($\mathcal{Z}_p$). As is, the objective function will be affected by the scales of the costs, therefore, each cost needs to be normalised. This requires knowledge of the minimum and maximum achievable values for each cost. The maximum is easily identified as any solution with all examinations scheduled to start at the same time, on the same day, will contain the maximal conflict and proximity costs. Obtaining the minimal conflict and proximity costs are not as simple. A minimiser ran for over two days and showed that a no-conflict solution exists. An estimate for the smallest proximity cost was obtained but it may not be the minimum. Table 4.1 below displays all estimates for both costs.

Table 4.1: Cost extrema

	Maximum	Minimum
Conflict	105618	0
Proximity	139169.3	121401*

* Not necessarily the true minimum

Let $\mathcal{Z}^U$ and $\mathcal{Z}^L$ denote the maximum and minimum of each cost respectively, then a normalised cost $\mathcal{Z}^*$ is:

$$\mathcal{Z}^* = \frac{\mathcal{Z} - \mathcal{Z}^L}{\mathcal{Z}^U - \mathcal{Z}^L} \tag{4.2}$$

A normalised cost is dimensionless. Simply adding the two costs together is equivalent to assigning the same priority to both costs. With *conflict* dictating the feasibility of a solution, it should be weighed much higher than *proximity*. A constant multiplier, C, is incorporated into the *conflict* cost to guarantee that it is given preference during the minimisation process. Three levels are tested: $C = 1, C = 139170$, and $C = 70000$. Each level was ran five times with 500 iterations for the minimiser each time. The results for each cost is displayed (Figure 4.1). The $C = 70000$ runs are omitted as they were identical to the $C = 139170$ runs.

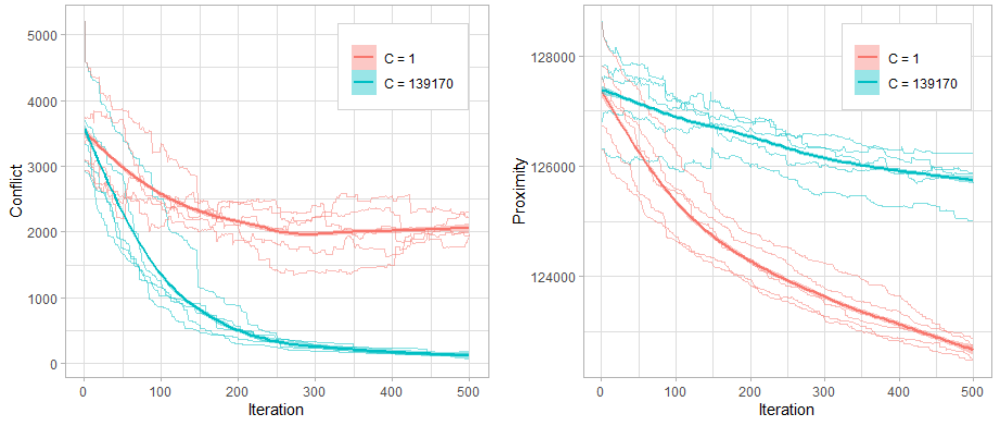

Figure 4.1: Sensitivity analysis of the multiplier, C, on each cost. Runs were grouped and smoothed using general additive models.

When costs were weighted equally, a point was reached where the objective function began to prefer minimising proximity over conflict. This choice is synonymous with minimising in favour of infeasibility and is clear evidence for the requirement of the multiplier. With $C = 139170$, the conflict cost is strictly decreasing upon closer inspection of individual runs. This causes the minimisation of the proximity cost to suffer, as expected. The objective function has shown to behave correctly when $C = 139170$, thus, the final objective function is: $139170\mathcal{Z}_c^* + \mathcal{Z}_p^*$.

Chapter 5

Proposed Algorithms

This chapter outlines the algorithms that will be used to solve the problem formulated in the previous chapter. The main algorithm considered is *tabu search*. The reasoning for this is that literature has proven its success with similar-sized datasets and the author is familiar with its implementation. *Nelder-Mead* was selected as another algorithm as a deterministic counterpart. This allows comparisons to be made against the stochastic nature of tabu search. Nelder-Mead was specifically chosen as it is the default method in R's `optim` function.

5.1 Tabu Search

Tabu search is a metaheuristic that employs local search methods for optimisation. The local search methods prohibit certain solutions (hence the term *tabu*) based on the optimisation history, but concession can be made if aspiration criteria are met. With an initial solution, tabu search begins by identifying and selecting neighbouring solutions. A *neighbour* is defined by the local search method implemented. This dissertation has three neighbourhoods, only using one during an iteration. Let X be a solution, then a neighbour N can be:

- *Random*: All N_i can be different to all X_i;
- *Swapped*: There exists i and j such that $N_i = X_j$ and $N_j = X_i$ but $N_k = X_k$ if $k \notin \{i, j\}$; or
- *Changed*: Except for a single j, $N_i = X_i \; \forall \; i \neq j$.

A tabu list functions as the method's memory since it records which moves create chosen neighbours. While moves are in memory, neighbours created from those moves are disallowed. Aspiration criteria detail when disallowed neighbours can be chosen but no such criteria is introduced in this dissertation as the prohibitions are integrated into the neighbour generation process. Thus, all neighbours created are explicitly allowable. That said, the list has limited recollection, allowing previously forbidden moves to become permissible as newer moves enter the list. Selected neighbours are evaluated and

the best neighbouring solution replaces the current solution - the incumbent solution is updated if necessary. The method can then begin again as the iteration has concluded.

5.2 Nelder-Mead

In any given space, a simplex is the simplest possible "flat"-sided geometric object. In this context, "flat" means that each side of a $(k+1)$-polytope consists of k-polytopes that may have $(k-1)$-polytopes in common. The simplest three-dimensional polyhedron is a tetrahedron, making it a simplex in the third dimension. Similarly, the triangle is a two-dimensional simplex. Nelder and Mead [49] used the ideas introduced by Spendley, Hext, and Himsworth [64] to create a minimisation method which uses simplices effectively. In a k-dimensional simplex, there are $k+1$ vertices and each vertex is a solution to the optimisation problem. The Nelder-Mead method starts by the creation and evaluation of a simplex. Let y_i denote the evaluation of the vertex v_i, then, without loss of generality, order the vertices $v_1, v_2, ..., v_{k+1}$ where $y_1 \leq y_2 \leq ... \leq y_{k+1}$.

By consistently updating vertices, the simplex traverses the search space towards the optimal vertex/solution. This process begins by calculating the centroid of the simplex, $\bar{v}$:

$$\bar{v} = \frac{1}{k} \sum_{i=1}^{k} v_i \tag{5.1}$$

The centroid is the average of all vertices, excluding the worst (v_{k+1}). If a vertex is an n-dimensional solution, then each element of the centroid is the element-wise average of the vertices. The centroid is used to calculate new vertices that will replace v_{k+1} if certain criteria are met. These vertices are created using three geometric transformations - *reflection*, *contraction*, and *expansion*. The *reflected vertex*, v_r, is defined as:

$$v_r = (1+\alpha)\bar{v} - \alpha v_{k+1} \qquad \alpha > 0 \tag{5.2}$$

Let y_r denote the evaluation of v_r. If $y_r < y_1$, a new minimum has been produced and the v_r is expanded:

$$v_e = \gamma v_r + (1-\gamma)\bar{v} \qquad \gamma > 1, \gamma > \alpha \tag{5.3}$$

The *expanded vertex*, v_e, is evaluated, and, if $y_e < y_1$, then v_e replaces v_{k+1}. Otherwise, v_r replaces v_{k+1}. The process can now restart with the new simplex. If $y_1 \leq y_r \leq y_k$, i.e. v_r is at least as bad as the second-worst solution, then v_r replaces v_{k+1} and the process restarts with the new simplex. If $y_k < y_r$, the worst vertex can be replaced by v_r if $y_r \leq y_{k+1}$. After which the *contracted vertex*, v_c, can be calculated:

$$v_c = \beta v_{k+1} + (1-\beta)\bar{v} \qquad 0 < \beta < 1 \tag{5.4}$$

If the evaluation of v_c, y_c, is better than the worst vertex, v_c replaces v_{k+1}. If the contraction has failed, i.e $y_c > y_{k+1}$, each vertex of the simplex is shrunk towards the best vertex:

$$v_i = \frac{1}{\delta}(v_i + v_1) \qquad 0 < \delta < 1 \tag{5.5}$$

This completes an iteration of the Nelder-Mead method and it can be restarted with the new simplex.

5.3 Parameter fine-tuning

An algorithm's effectiveness can have widely varying results depending on the choice of parameter values. While some of these parameters, such as the run time of an algorithm, may have clear relations to the success of the algorithm, others may require tuning. This section explores the main parameter(s) in each algorithm and suggests baseline characteristics.

5.3.1 Choice of neighbourhood

By running the tabu search with an exclusive neighbourhood, the optimal threshold for each neighbourhood can be discovered. Five 500-iteration runs were ran for each neighbourhood. A tabu list length of 50 was used with five neighbours selected during each iteration. The results thereof are shown by Figure 5.1.

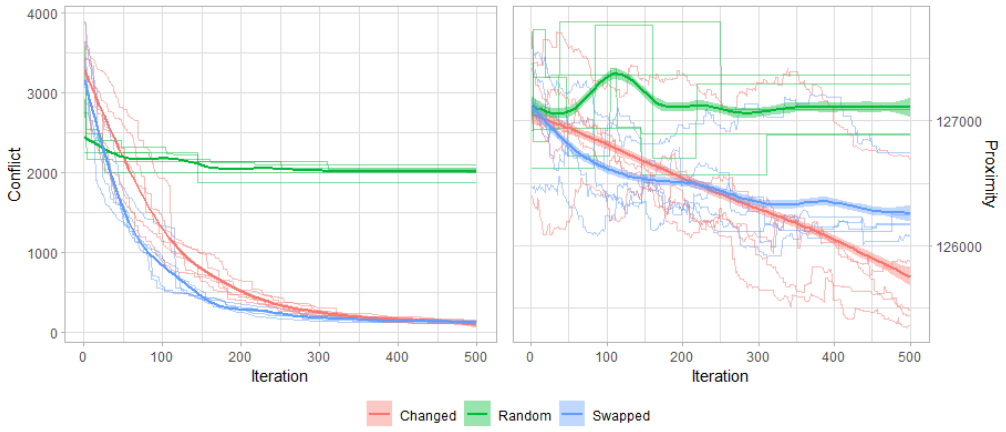

Figure 5.1: Sensitivity analysis of the tabu search neighbourhood on each cost. Runs were grouped and smoothed using general additive models.

The *random* neighbourhood had the fastest initial improvement in solution. The sharp drop in the number of conflicts show that it should be the neighbourhood used upon initialisation of the method. Once solutions improve and reach a conflict value of around 2500, the *changed* neighbourhood should be implemented as it had the steepest slope after 2500 clashes until roughly the 600-clashes mark is reached. It can be

argued that the *swapped* neighbourhood should be used below this threshold. However, the problem with the *swapped* neighbourhood is that it is the best neighbourhood if, and only if, the optimal selection of periods has already been chosen. The *swapped* neighbourhood essentially shuffles periods around in an attempt to find the optimal timetabling. An artefact of this is that the method becomes trapped in a local minimum. For this reason, the *changed* neighbourhood will be used from the 2500 clashes mark and below as it allows the possibility of obtaining the optimal solution.

5.3.2 Coefficients

In their 1965 paper, Nelder and Mead tested various coefficient combinations. Their results showed that the combination of $\alpha = 1, \gamma = 2, \beta = 0.5, \delta = 2$ worked best. In all their tests, they kept δ fixed. This *baseline* combination is compared to a combination with *enlarged* coefficients and a combination with *shrunken* coefficients. All combinations fix $\delta = 2$:

- *Baseline*: $\alpha = 1, \gamma = 2, \beta = 0.5$
- *Enlarged*: $\alpha = 1.5, \gamma = 3, \beta = 0.75$
- *Shrunken*: $\alpha = 0.5, \gamma = 1.5, \beta = 0.25$

In Figure 5.2 below, each coefficient combination was used in five runs each, with 100 iterations per run. From Figure 5.2, the *baseline* combination produces the best results and will be used in future runs. Nevertheless, there does not seem to be a significant difference between combinations. It should be noted that not much improvement has been made by any of the runs over the 100 iterations, especially when compared to the first 100 iterations of the tabu search runs. This may suggest that tabu search is better suited to this problem than Nelder-Mead.

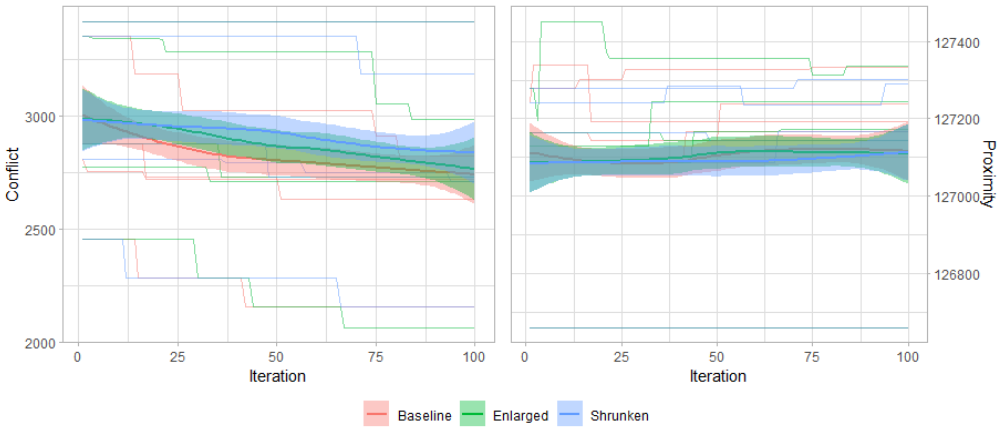

Figure 5.2: Sensitivity analysis of the Nelder-Mead coefficient combination on each cost. Runs were grouped and smoothed using local polynomial regression.

5.4 Coded methods

The methods described were coded in R. In order to complete the coding of both methods, a stopping criterion is required. A natural stopping point would be at a specific objective function value or once a maximum number of iterations has been completed. This requires intimate knowledge of the data, the objective function and possibly the method(s). A layman is unlikely to possess the ability to ascertain appropriate values for any of these criteria. Therefore, a more intuitive stopping criterion is a time limit. By limiting time, multitasking is encouraged as users can expect results at a later time and continue with other work. The `stringr`, `dplyr` (Wickham et al. [75]), and `parallel` (R Core Team [55]) packages were used to help speed up methods.

Let f be the objective function to be minimised on a solution v and $f_c(v)$ the conflict value of solution v, then tabu search and Nelder-Mead can be described by Algorithms 1 and 2 respectively. Where possible, default values are supplied for optional input if they are not given. In the case of Nelder-Mead, it uses a simplex so assume a solution is k-dimensional. In this dissertation, the other k vertices in the simplex are *changed* neighbours of the initial solution.

Algorithm 1 Template of the *tabu search* method

Required input: Exam data, time limit
Optional input: Initial solution, tabu list length, number of neighbours, seed
set seed
$v \leftarrow v_0$ ▷Generation of an initial solution if none is given
$v_b \leftarrow v$
Generate a tabu list ▷Determined by the tabu list length
repeat ▷Based on the number of neighbours, n, and the tabu list
    if $f_c(v) > 2400$ **then**
        Generate n *random* permissible neighbours $v_1, v_2, ..., v_n$
    else
        Generate n *changed* permissible neighbours $v_1, v_2, ..., v_n$
    Select v_i from the neighbours such that $f(v_i) \leq f(v_1), f(v_2), ..., f(v_n)$
    $v \leftarrow v_i$
    if $f(v) < f(v_b)$ **then**
        $v_b \leftarrow v$
    Update tabu list based on neighbours generated
until *time limit is exceeded*
Output: Best solution found, v_b

Algorithm 2 Template of the *Nelder-Mead* method

Required input: Exam data, time limit
Optional input: Initial solution, coefficients $(\alpha, \gamma, \beta, \delta)$, seed
set seed
$v \leftarrow v_0$ ▷Generation of an initial solution if none is given
Generate a simplex S from v ▷S has $k+1$ vertices
Order the vertices of S: $v_1, v_2, ..., v_{k+1}$ such that $f(v_1) \leq f(v_2) \leq ... \leq f(v_{k+1})$
repeat
    $\bar{v} = \frac{1}{k} \sum_{i=1}^{k} v_i$ ▷centroid of S (without the worst vertex)
    $v_r = (1+\alpha)\bar{v} - \alpha v_{k+1}$ ▷reflective vertex of S
    if $f(v_r) < f(v_1)$ **then**
        $v_e = \gamma v_r + (1-\gamma)\bar{v}$ ▷expanded vertex of S
        if $f(v_e) < f(v_1)$ **then**
            $v_{k+1} \leftarrow v_e$
        else
            $v_{k+1} \leftarrow v_r$
    else if $f(v_r) \leq f(v_k)$ **then**
        $v_{k+1} \leftarrow v_r$
    else
        if $f(v_r) \leq f(v_{k+1})$ **then**
            $v_{k+1} \leftarrow v_r$
        $v_c = \beta v_{k+1} + (1-\beta)$ ▷contracted vertex of S
        if $f(v_c) > f(v_{k+1})$ **then**
            $v_i = \frac{1}{\delta}(v_i + v_1)$ for all $i \in \{1, 2, ..., k+1\}$ ▷shrink the entire simplex
        else
            $v_{k+1} \leftarrow v_c$
until *time limit is exceeded*
Output: Best solution found, v_1

Chapter 6

Results

This dissertation implemented and tested the tabu search and Nelder-Mead methods described in Proposed Algorithms, on the UCT examination timetabling data, on a PC with an Intel i7 3.6 GHz processor, 8 Gb RAM and Windows 10. Each method was tested at time limits of 30 minutes, 1 hour, and 2 hours. The 2-hour time limit may seem short but, for example, papers by Čupić, Golub, and Jakobović [24] and Carter, Laporte, and Lee [14] managed to obtain results within minutes. Another reason for the shorter time limit is to push the envelope. The debate may be that if a computer takes just as long as a human then the human should do it as their skill can be trusted. By showing what's possible within only two hours the narrative changes to what the computer can do given more time. This links back to objective three of this paper, discourse. Tests were conducted five times each and the results obtained are discussed in this chapter. Five repetitions were chosen to gain some insight into what the average performance of the algorithms are. All figures and tables were created using the `ggplot2` (Wickham [72]), `viridis` (Garnier et al. [34]), and `ggpubr` (Kassambara [41]) packages in R and can be found in Appendix A.

6.1 Tabu Search

A tabu list length of 50 with five neighbours generated per iteration was used for all runs, with each run having a random seed value. The differences between the initial (seed) solution and the best solution, after the time limit was reached, was recorded and tabulated. A collated summary of these tables is given on the following page (Table 6.1). For example, in the best 30-minute run, the tabu search method reduced the initial solution's total cost by 5866.390 after 30 minutes elapsed. Surprisingly, the average improvement from the 1-hour time limit is greater than that of the 2-hour time limit. This is likely due to the randomness of the initial solutions coupled with the increasing difficulty in improvement as solutions get closer to feasibility.

Table 6.1: Summary of improvements from tabu search runs on random initial solutions

Run		Total Cost	Conflict Cost	Proximity Cost	Run Time
30-mins	Best	5866.390	4452	1943.3	30.0125 mins
	Average	4913.661	3729	1025.7	30.0485 mins
1-hour	Best	6114.076	4640	1297.4	1.0013 hours
	Average	5413.893	4109	1790.8	1.0012 hours
2-hour	Best	6355.227	4823	1595.8	2.0004 hours
	Average	5187.538	3937	2177.9	2.0004 hours

Without an idea of the initial and final conflicts, it is difficult to put the improvements into perspective. Figure 6.1 shows the change in conflicts of each of the five 2-hour time limit runs. In these runs, tabu search has reduced each solution by a factor of around 100. Despite this, none of the solutions have zero clashes: the minimum clashes reached was 27. This suggests that more than two hours is required to obtain a feasible solution if a random solution is used.

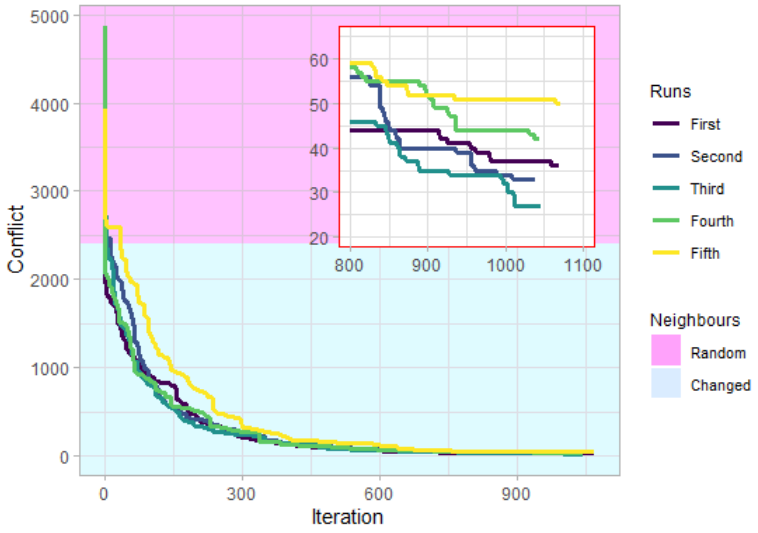

Figure 6.1: Conflict value as best solutions change in tabu search runs with 2-hour time limits

Figure 6.1 also reveals that the threshold for changing neighbourhoods is possibly too low, seeing the kink in at least one run. The 30-minute and 1-hour time limit figures exhibit the same problem as this one. In this case, the fifth run has been significantly hindered in its improvement by not switching neighbourhoods sooner. This then creates a time sink as the method struggles to optimise the run until it has at most 2400 clashes so that its neighbourhood can be changed. A possible improvement would be to increase the threshold value to 2600.

6.2 Nelder-Mead

Nelder-Mead was tested with two objectives in mind. Firstly, its performance relative to tabu search and, secondly, its performance as a secondary optimiser. The Nelder-Mead baseline coefficient values were used: $\alpha = 1, \gamma = 2, \beta = 0.5, \delta = 2$.

6.2.1 Against tabu search

In order to test its performance relative to tabu search, it was given the same seeds so that the improvements from each Nelder-Mead run can be compared directly to its tabu search counterpart. By comparing the Nelder-Mead summary table (Table 6.2) to the tabu search summary table (Table 6.1), a clear difference in optimisation ability can be seen. In the best 30-minute run, the proximity cost actually worsened instead of improved thus the negative value in the table. This perfectly demonstrates the intention of choosing the weight values for the objective function as conflict was reduced by worsening the proximity cost.

Table 6.2: Summary of improvements from Nelder-Mead runs on random initial solutions

Run		Total Cost	Conflict Cost	Proximity Cost	Run Time
30-mins	Best	1478.429	1122	-4.7	32.3064 mins
	Average	923.427	701	29.0	33.6169 mins
1-hour	Best	3143.977	2386	164.1	1.0738 hours
	Average	1784.395	1354	41.9	1.0792 hours
2-hour	Best	2412.668	1831	147.6	2.0968 hours
	Average	1928.550	1464	55.9	2.1153 hours

Table 6.2 also shows that tabu search improvements are at least three times greater than Nelder-Mead on average. For these runs, the average improvement in the 1-hour time limit runs is worse than that of the 2-hour time limit runs. A reason for this can be explained by looking at Figure 6.2. After two hours, the best solution obtained from the five runs still had 1768 clashes. For context, the best tabu search run obtained a solution with 27 clashes in this time. Moreover, this Nelder-Mead run had the best initial solution out of the five and the best solution from the tabu search runs had the second-worst initial solution out of the five - an initial difference of 1455 conflicts. The Nelder-Mead runs had not yet reached a point where the difficulty in improvement really mattered as solutions were still far from feasible. These results show that tabu search is the clear favourite between the two methods.

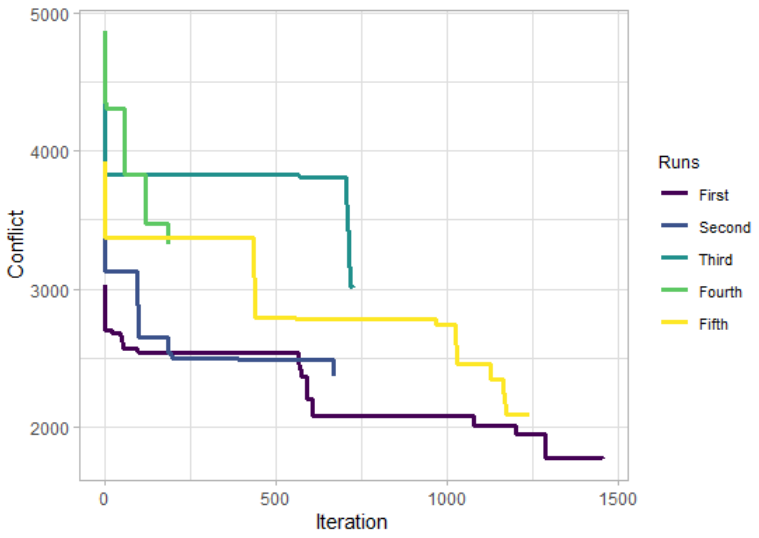

Figure 6.2: Conflict value as best solutions change in Nelder-Mead runs with 2-hour time limits

6.2.2 Ensembled with tabu search

Nelder-Mead is demonstrably worse than tabu search at improving random initial solutions. However, if the solutions are already near-feasible then Nelder-Mead may be more effective. To test this hypothesis, the tabu search runs used previously had their output used in Nelder-Mead runs. These Nelder-Mead runs did not use new seeds and continued from the tabu search seeds. The time limit given to the Nelder-Mead runs were the same as those given to the tabu search runs, e.g., a 30-minute tabu search run would have a 30-minute Nelder-Mead run after. Table 6.3 displays the summary of these runs.

Table 6.3: Summary of improvements from Nelder-Mead runs on tabu search initial solutions

Run		Total Cost	Conflict Cost	Proximity Cost	Run Time
30-mins	Best	140.991	107	-1.0	41.5034 mins
	Average	70.100	53	-11.7	39.8016 mins
1-hour	Best	25.036	19	6.4	1.1516 hours
	Average	18.449	14	26.4	1.1693 hours
2-hour	Best	23.715	18	-54.7	2.2269 hours
	Average	10.010	8	-70.8	2.0997 hours

Apart from the 1-hour time limit runs, the promixity cost figures have all increased. Moreover, the run times for these Nelder-Mead runs are higher than the previous section's Nelder-Mead runs: as solutions improve, the probability of the worst vertex improving in the Nelder-Mead simplex decreases. Thus, the simplex is more likely to be shrunk. Out of all its operations, shrinking the simplex requires the most time and therefore the average run time will increase as shrinking becomes more common. The Nelder-Mead method has shown that it can improve solutions, even near-feasible ones, but it begs the question whether the time spent on it should rather be used to increase the tabu search time limit.

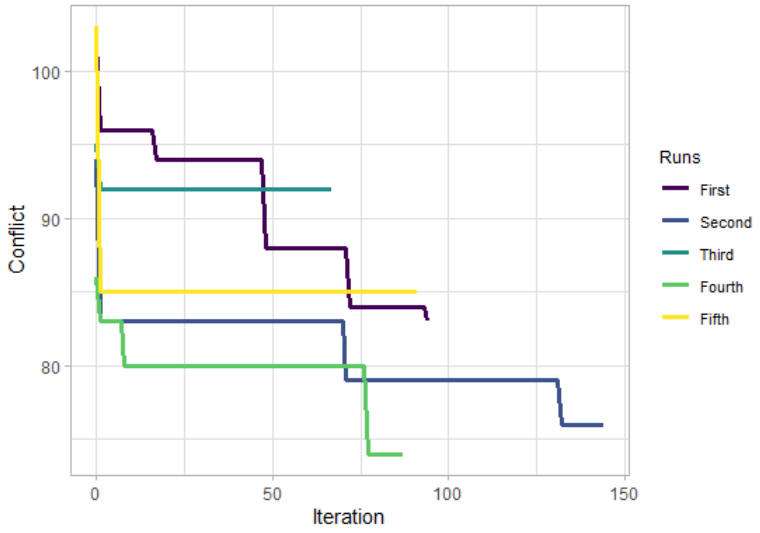

Figure 6.3: Conflict value as best solutions change in Nelder-Mead runs with a 1-hour time limits on tabu search runs with 1-hour time limits

Figure 6.3 shows the 1-hour time limit Nelder-Mead runs. At best, they have only decreased the total conflicts by 30 in a 1-hour time span. Tabu search has shown that, with an additional hour, it can reduce conflicts by much more than this. A positive note is that the 2-hour time limit runs were further decreased when Nelder-Mead was applied. This does not disprove that a 4-hour time limit tabu search run will not outperform the 2-hour time limit Nelder-Mead runs but it does prove that the method is capable, given enough time. Further testing needs to be done to ascertain the usefulness of the Nelder-Mead method.

Chapter 7

Graphical User Interface

In this chapter, a decision support system is designed so as to provide the Examinations Office with a tool that may be used to generate examination timetables in line with their objectives. The platform may also be seen as a tool to enhance the repeatability of the work performed in this dissertation. The app was created using the `shiny` (Chang et al. [17]), `shinyjs` (Attali [4]), `readxl`, and `stringr` packages in R.

7.1 App architecture

The main purpose of the app is to provide support to a user during the process of generating examination timetables. It is envisaged that the user of the app will be an Examinations Office manager and that such a user, having access to this dissertation, can gain a basic understanding of the solution approaches employed in the app. The user is, however, not expected to know the working of the tabu search and Nelder-Mead methods prior to using the app. The code of these methods are neatly packaged and the platform allows the user to easily change the problem parameters without having to explicitly make changes to the programming code.

The user is required to import the data into the application if no data has been stored locally. From here, the user gains the ability to view the data as well as run the solution approaches according to user-specified parameters. Once a run has completed, the application is updated and the user can view the new timetable. Various forms of output are created by the app and the user may download these if required.

7.2 App implementation

Upon starting the app, the user lands on the *Data* tab. Users will be met with the "new data" selection under **Data** and are required to import the necessary data files into the application, using the "browse" buttons (Figure 7.1, 1), before the additional tabs are unlocked (Figure 7.1, 2). These data files are the student data, provisional timetable, and venue locations and capacities files described in Implementation and should be

imported under **Student data**, **Exams**, and **Venues** respectively. In practice, these files would be pulled from the human resources management system and imported directly into the app. For easy assimilation, the app requires *.xls* and *.xlsx* file extensions. Once all files have been imported, the user has the option to upload a solution under **Solution (Optional)** in the form of a text file (*.txt*). If none is provided, a default solution will be used.

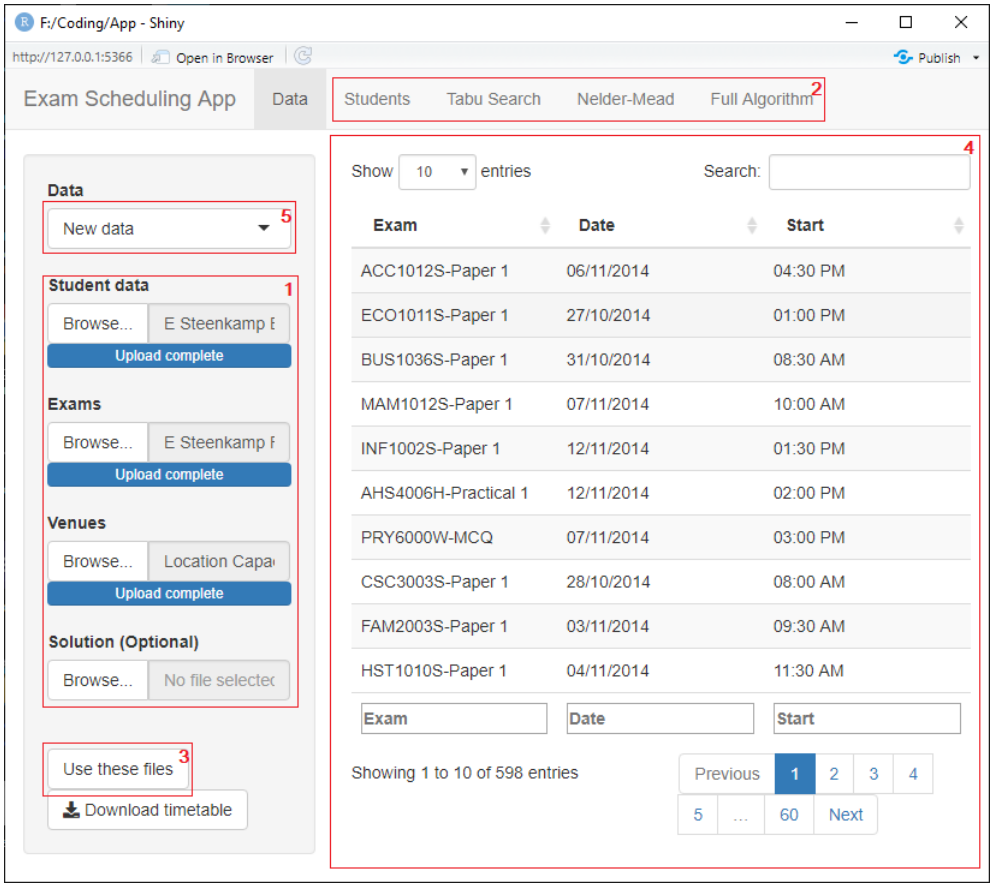

Figure 7.1: An example of the app landing page and its features

The user has now completed the initial requirements and can select the "use these files" button (Figure 7.1, 3). If done, the application will begin cleaning the attached files according to the **Data wrangling** section in Implementation. The cleaned files are stored by the app and will be used in all further processes. The first of which is to create an initial examination timetabling with the solution (default or provided). The solution will be displayed on the *Data* tab (Figure 7.1, 4) and the user can cycle through the entries using the page numbers in the bottom right of the app. The user may also filter the entries using the text boxes above the page numbers or download the timetable in excel format via the "download timetable" button. If the data has been cleaned previously, the user can select "existing data" from the **Data** drop-down box

(Figure 7.1, 5) as the cleaned files should be stored locally. The user will then only be given the option of uploading a solution and the same process will be completed once the user selects the "use this file" button, previously the "use these files" button. After either form of initialisation, additional tabs are unlocked: *Students*, *Tabu Search*, *Nelder-Mead*, and *Full Algorithm*.

By selecting the *Students* tab, the user can view the timetable from a student perspective: entries are given for all students and the exams they will be sitting (Figure 7.2, 1). Here, **Duration** is the student-specific (extra-time included) exam duration allowed. In the top-left box, the objective function value of the timetable is displayed along with its component values (Figure 7.2, 2). Below this, the "show clashes only" button subsets the data to only show the entries which result in conflicts (Figure 7.2, 3). Selecting this button transforms the button to "show full data" where the data will be reverted if selected. The full and subsetted datasets are downloadable as excel files using the "download timetable" and "download clashes timetable" respectively.

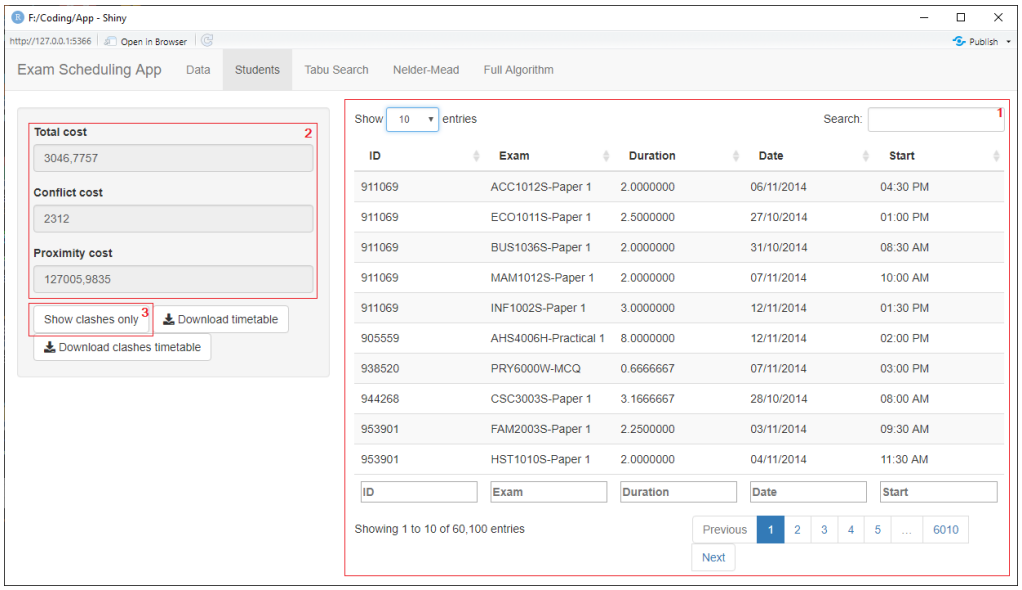

Figure 7.2: An example of the *Students* tab of the app

The last three tabs (*Tabu Search*/*Nelder-Mead*/*Full Algorithm*) should be used when improvements are needed for the current timetable. When selecting the *Tabu Search* or *Nelder-Mead* tab, the user is confronted with a checkbox on whether to use the uploaded solution (Figure 7.3, 1). This refers to the solution used in the current timetable (default or provided). Checking this box means that the current timetable should be used as the initial solution for the optimisation method. Leaving this box unchecked means that a new random initial solution will be used, affected by the **Seed value**. The user should then select the required run time duration (Figure 7.3, 2). The available units of time are "seconds", "minutes", and "hours" with 60, 60, and

24 being the maximum time limit for the respective units. As a default, the **change arguments** box is unchecked (Figure 7.3, 3). This ensures that users do not accidentally change the optimisation method's parameters by prohibiting changes to the current arguments. If the user would like to change the arguments, they should tick the box and the prohibition will be lifted (Figure 7.3, 4). After configuration, the user should select the "run algorithm" button to being optimisation (Figure 7.3, 5).

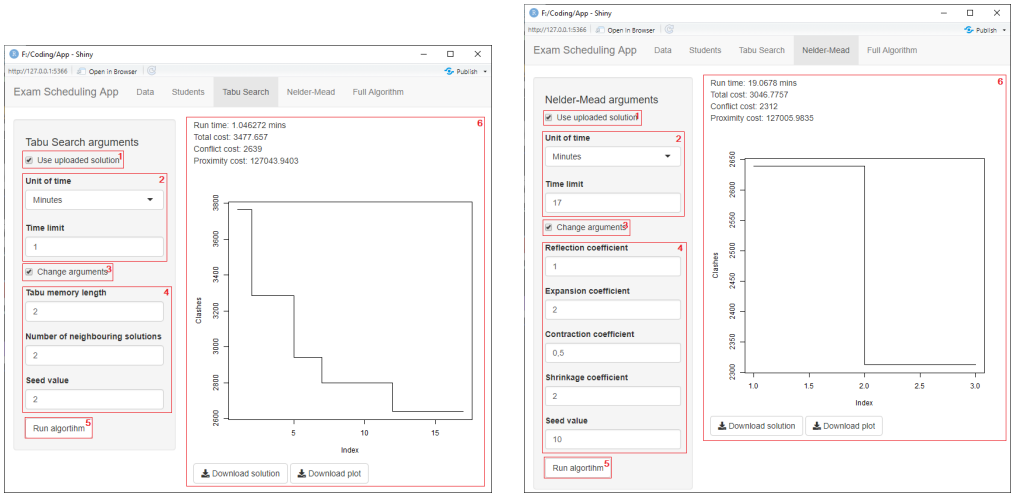

Figure 7.3: An example of the *Tabu Search* (left) and *Nelder-Mead* (right) tabs of the app

Once the run time duration has reached the specified time limit, the method will stop optimisation and update the app. The current tab will be updated with the run time of the method, the objective function value and its components, and a plot of the history of the conflict values, which may be downloaded as a *.png* file by selecting the "download plot" button (Figure 7.3, 6). The tabu search plot will contain a red horizontal line at the 2400-clashes mark if the method has changed its neighbouring solution generation process during the run. The "download solution" button will download the final solution from the optimisation as a *.txt* file. This file is formatted to allow it to be directly imported into the app on future use. The app also uses the final solution to update the *Data* and *Students* tabs. These tabs will now reflect the new timetable from the optimisation while retaining all previous functionality. Lastly, the *Full Algorithm* tab is simply an amalgam of the *Tabu Search* and *Nelder-Mead* tabs. The reason for this tab is to allow ensembled runs as explained in Results. The only additional parameter is the checkbox whether to use the seed for the Nelder-Mead portion of the method (Figure 7.4, 1). To replicate the ensembled runs, this box should not be ticked. Ticking this box is equivalent to running the *Nelder-Mead* tab with "use uploaded solution" after running the *Tabu Search* tab.

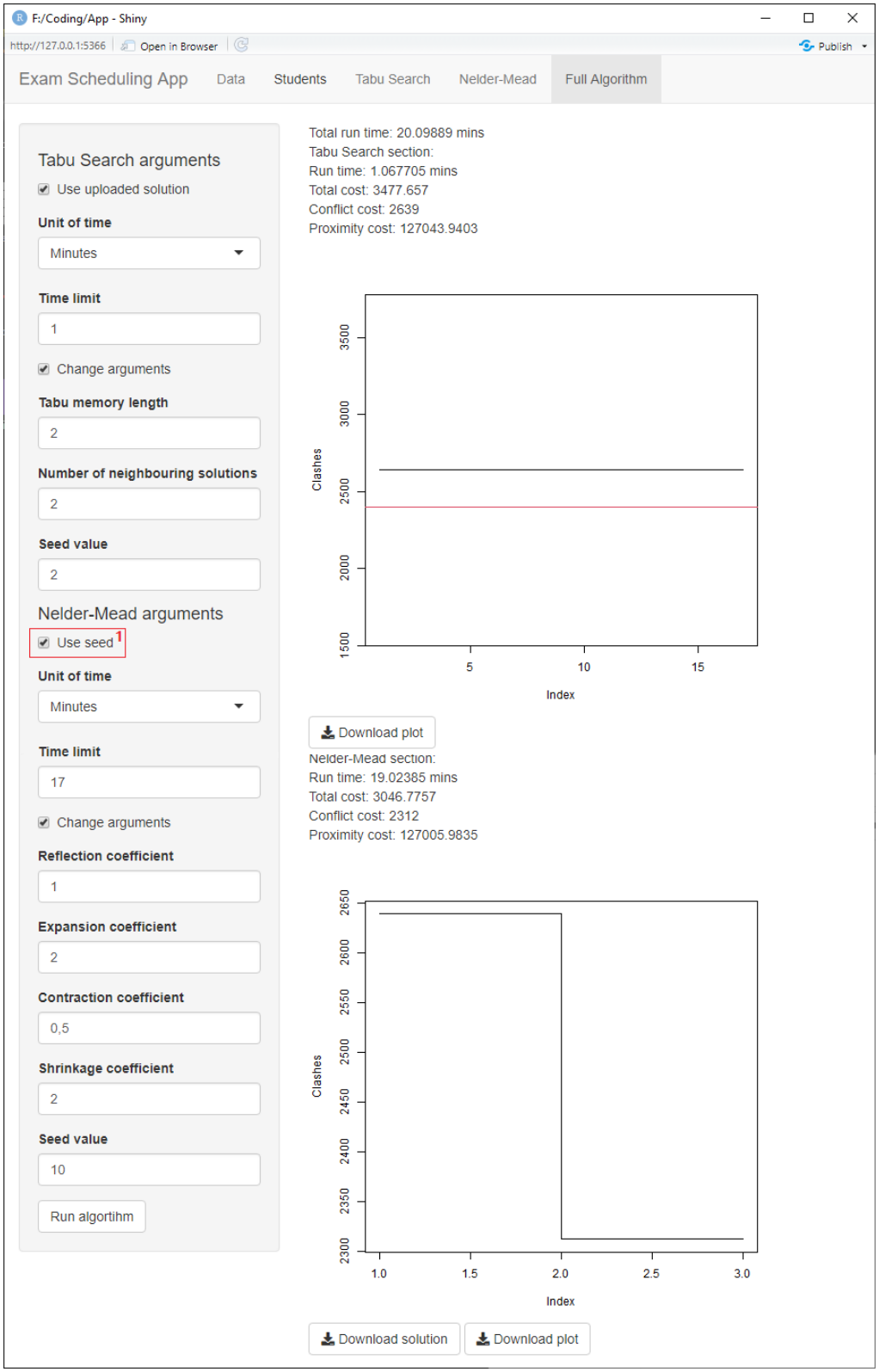

Figure 7.4: An example of the *Full Algorithm* tab of the app

Chapter 8

Discussion and Conclusion

In essence, the UCT examination timetabling process has successfully been automated herein and been presented in a user-friendly manner with the accompaniment of the app in Graphical User Interface. Neither the automation nor the app are free from improvements however. Nevertheless, each section of the automation process is discussed in this chapter with future considerations on the development of the respective section.

8.1 Solution

The best solution achieved through the test runs still included 27 conflicts and is therefore infeasible but when compared to the provisional timetable, the solution is notably favourable. Table 8.1 shows the cost breakdown of the provisional timetable and the best test run, including a row which shows the difference between the two solutions' costs. It is reasonable to believe that this provisional timetable solution was not the one published for November 2014 examination period as there would have definitely been issues for certain students. That said, one can assume that at least some time was spent creating this solution and if the time spent exceeded 2 hours then utilisation of the app is substantially superior.

Table 8.1: Cost breakdown

	Conflict	*Proximity*	*Total*
Provisional timetable	307	126664.6	404.8
Best test run	27	124864.6	35.8
Difference	280	1800	369

The conflicts in the provisional timetable are due to 302 unique students and 28 unique exams whereas the conflicts in the best test run are due to 27 unique students and 39 unique exams. Bear in mind that for a conflict to exist, a student should be unable to sit two or more exams and that's why the number of unique exams are larger than the number of conflicts for the best test run. Note that the conflicts in

provisional timetable are generated by less unique exams than those in the best test run. This suggests that conflicts in the provisional timetable are more likely a case of difficult-to-schedule courses whereas the conflicts in the best test runs are likely a case of difficult-to-schedule students. As mentioned in the introduction, degree-flexibility allows more unique course combinations and therefore may require a more complex exam timetable. Out of the two solutions, there are no students that feature in both and only one exam, PSY2003S paper 1, causes conflicts in both solutions.

8.2 Data

Insufficient data can severely impact the scope of a problem as is the case with the UCT November 2014 examination scheduling problem. It was immediately clear that automation of venue allocation would not be possible due to the lack of data, unfortunately making this an incomplete automation process. Here, data does not just pertain to the files obtained from the Examinations Office but also includes the information from meetings and documentation. Literature on examination scheduling assisted in identifying the data that was needed to automate UCT's exam scheduling but assumptions were still required in cases where the data was not available. For example, the longest exam scheduled to start at 5 pm (the last period) in the Examinations Office's solution was 3 hours and 15 minutes. The most amount of time required by an extra-time student for any of these exams was 15 minutes per hour. This means that the student would have possibly ended up completing their sitting after 9 pm. The Examinations Office did not specify if they had a time point where students should not be required to be on campus but one can imagine that it would not be too late as some students may require public transportation, making late ending times unsafe and inconvenient. Feedback is also a form of data and, ideally, the app would have been developed in tandem with the Examinations Office to customise it to their specific needs.

8.3 Problem Formulation

Constraints, costs, and objective functions are highly subjective and depend entirely on the problem at hand. The feasibility of a solution can change as those elements change. This makes identifying the solution space and feasibility region an important step towards automation. The formulation shown in Problem Formulation contains a very general objective function and has little to no UCT-specific constraints and costs. The University of Cape Town is an established tertiary institution so it is likely that it keeps its constraint and costs (relatively) constant for examination timetabling. The 2019 UCT Examinations Policy & Procedures Manual details guidelines that the exam schedulers should follow. However, these guidelines could not be adhered to as they require a lot more information than was made available. The Examinations Office may also have certain criteria they follow which the manual does not explicitly state. These criteria would take the form of soft constraints or costs. More communication with the

Examinations Office would be required for a full formulation of the UCT examination scheduling problem.

8.4 Algorithm

Two algorithms were tested during the automation process. Unfortunately, the results from the Nelder-Mead algorithm were less than satisfactory but still provided valuable insight. When time is not a limiting factor, an algorithm needs to be robust enough to traverse the solution space in order to discover a feasibility region. An issue with using a simplex is that each step in the algorithm creates a solution linked to the simplex and this can be problematic when the entire simplex is situated in a locally minimal region.

The tabu search was able to avoid early convergence with the use of differing neighbouring solution generations. The first type of generation, *random*, creates solutions which do not have to be linked to the initial solution in any way, therefore rapid exploration is possible. The use of the second and third types of generation, *swapped* and *changed*, then help to refine solutions towards feasibility. Additionally, tabu searching provides efficient time usage with iterations of roughly the same duration. With the Nelder-Mead algorithm requiring differing amounts of time depending on the initial solution, it gives the user a non-intuitive understanding of setting the time limit since the user would not be able to estimate the number of iterations the algorithm would complete.

Tabu searching is not without its faults though. The results sections has shown that tabu searching can have difficulty moving from a near-feasible to feasible solution. As tabu searching is similar to simulated annealing and genetic algorithms, comparisons should be made to check if tabu searching is indeed a good choice. Perhaps the solution generation process could be adapted to include other techniques proven in literature or perhaps a different algorithm should be used altogether. A future consideration would be to automatically attain the best arguments for the algorithm and dynamically change them if necessary. Further testing should also be done on when to switch neighbouring solution generation types as Results hinted at non-optimal neighbourhood switching.

It should be noted that the run times of the algorithms are likely to decrease if a faster programming language is used. As R uses C/C++ and Fortran at its lower level, writing the algorithms in these languages directly will reduce overhead and thus should increase the speed at which an algorithm completes its iterations.

8.5 Graphical User Interface

It has been shown that an app can streamline the workflow of the UCT examination timetabling. While basic in its functionality, the app can successfully automate the scheduling process and provide the necessary output. However, the app is not without its issues. Proper fool-proofing has not yet been administered as it is possible to break the app in many ways. The app also lacks helpful tips and guidelines to steer users towards correct app usage. This all assumes the user can launch the app in the first place. The app requires R software and multiple other packages in order to execute certain functions and a user would have to open an R script and subsequently run the code to actually launch the app. This may prove difficult as individuals are accustomed to clicking an icon on their desktop or in their start menu. A workaround exists where an executable (*.exe* file) can be created that will perform all these tasks for the user, including downloading and installing R and its packages, but this has yet to be attempted.

A key task that the app cannot perform is to allow users to change constraints/costs and manually adjust the solution. As mentioned previously, UCT creates provisional timetables before finalising their examination schedule. As new constraints are introduced, the current solution may become infeasible and require adjustments. Rerunning the entire algorithm may be unnecessary if an improvement is clearly visible to the user. Unfortunately, the user cannot interact with the app to update the constraints and solution. Changing the constraints would require the user to have knowledge of R so that they can re-code the constraints and as the solution is given in text file format, adjustment would require opening the file and changing the values of the solution. Unless the user understands the workings behind the algorithms, it would not result in the intended adjustments.

On the technical side, it is evident that running the algorithms through the app are slower than running them through the script. Naturally, the hardware that the app is run on will affect the speeds at which actions are completed but, because the app itself requires resources, less resources are available for the algorithms when needed. On a whole, the slow down is negligible but a clear difference in run times are visible when utilising the Nelder-Mead algorithm, in the *Nelder-Mead* or *Full Algorithm* tabs. All apps require resources to operate but creating the app through R (and shiny) may be more resource intensive than alternatives. Furthermore, the app code itself may also not be optimised which only worsens the slowdown effect. The problem areas with the app are predominantly (i) user experience and that (ii) feedback is needed to ascertain what future app developments should include. A meeting was held with the Examinations Office where the app was showcased, and they were in accord with the considerations discussed. In the meeting, they mentioned that, in its current state, if the app contained the constraints that UCT uses, it would be used as a secondary tool. Essentially, once an agreeable solution has been obtained by the Examinations

Office, it would be fed into the app for further improvement (if possible). Hence, if the app was expanded into a more substantial programme, the Examinations Office is in support of its use over their current system(s).

Bibliography

[1] Abdul Rahman, S., Bargiela, A., Burke, E. K., Mccollum, B., and Özcan, E. "A construction approach for examination timetabling based on adaptive decomposition and ordering". In: 2010, pp. 353–372.

[2] Abela, J. "A parallel genetic algorithm for solving the school timetabling problem". In: *Division of Information Technology, CSIRO*. Citeseer. 1991.

[3] Arogundade, O. T., Akinwale, A. T., and Aweda, O. M. "A genetic algorithm approach for a real-world university examination timetabling problem". In: *International Journal of Computer Applications* 12.5 (2010), pp. 0975–8887.

[4] Attali, D. *shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds*. R package version 2.0.0. 2020. URL: https://CRAN.R-project.org/package=shinyjs.

[5] Brailsford, S. C., Potts, C. N., and Smith, B. M. "Constraint satisfaction problems: Algorithms and applications". In: *European journal of operational research* 119.3 (1999), pp. 557–581.

[6] Brélaz, D. "New methods to color the vertices of a graph". In: *Communications of the ACM* 22.4 (1979), pp. 251–256.

[7] Britannica, The Editors of Encyclopaedia. *NP-complete problem*. Accessed: 20/03/2021. 2018. URL: https://www.britannica.com/science/NP-complete-problem.

[8] Burke, E. K., Bykov, Y., and Petrovic, S. "A multicriteria approach to examination timetabling". In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer. 2000, pp. 118–131.

[9] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. "A classification of hyper-heuristic approaches". In: *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[10] Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. "A graph-based hyper-heuristic for educational timetabling problems". In: *European Journal of Operational Research* 176.1 (2007), pp. 177–192.

[11] Burke, E. K. and Newall, J. P. "A multistage evolutionary algorithm for the timetable problem". In: *IEEE transactions on evolutionary computation* 3.1 (1999), pp. 63–74.

[12] Burke, E. K., Newall, J. P., and Weare, R. F. "A memetic algorithm for university exam timetabling". In: *international conference on the practice and theory of automated timetabling*. Springer. 1995, pp. 241–250.

[13] Burke, E. K., Kingston, J. H., and De Werra, D. "Applications to timetabling". In: *Handbook of Graph Theory* 445 (2004).

[14] Carter, M. W., Laporte, G., and Lee, S. Y. "Examination timetabling: Algorithmic strategies and applications". In: *Journal of the operational research society* 47.3 (1996), pp. 373–383.

[15] Chahal, N. and De Werra, D. "An interactive system for constructing timetables on a PC". In: *European Journal of Operational Research* 40.1 (1989), pp. 32–37.

[16] Chandrasekharan, R. C. and Wauters, T. "A constructive matheuristic approach for the vertex colouring problem". In: *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT*. Vol. 1. 2021.

[17] Chang, W., Cheng, J., Allaire, J. J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., and Borges, B. *shiny: Web Application Framework for R*. R package version 1.6.0. 2021. URL: https://CRAN.R-project.org/package=shiny.

[18] Charnes, A. and Cooper, W. W. *Management Models and Industrial Applications of Linear Programming*. Vol. 1. Management Models and Industrial Applications of Linear Programming. Wiley, 1961.

[19] Cheng, S., Liu, B., Ting, T. O., Qin, Q., Shi, Y., and Huang, K. "Survey on data science with population-based algorithms". In: *Big Data Analytics* 1.1 (2016), pp. 1–20.

[20] Clayson, D. E. and Haley, D. A. "Marketing models in education: Students as customers, products, or partners". In: *Marketing education review* 15.1 (2005), pp. 1–10.

[21] Coleman, T. F. and Moré, J. J. "Estimation of sparse Jacobian matrices and graph coloring blems". In: *SIAM journal on Numerical Analysis* 20.1 (1983), pp. 187–209.

[22] Cooper, T. B. and Kingston, J. H. "The complexity of timetable construction problems". In: *International conference on the practice and theory of automated timetabling*. Springer. 1995, pp. 281–295.

[23] Cowling, P., Kendall, G., and Soubeiga, E. "A hyperheuristic approach to scheduling a sales summit". In: *International conference on the practice and theory of automated timetabling*. Springer. 2000, pp. 176–190.

[24] Čupić, M., Golub, M., and Jakobović, D. "Exam timetabling using genetic algorithm". In: *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces*. IEEE. 2009, pp. 357–362.

[25] Dantzig, G. B. and Thapa, M. N. *Linear programming 1: introduction*. Springer Science & Business Media, 2006.

[26] De Jong, K., Fogel, D. B., and Schwefel, H.-P. "A2. 3 A history of evolutionary computation". In: *A1. 1 Introduction* (1997).

[27] Denzinger, J. and Fuchs, M. "High performance ATP systems by combining several AI methods". In: (1996).

[28] Diestel, R., Schrijver, A., and Seymour, P. "Graph theory". In: *Oberwolfach Reports* 7.1 (2010), pp. 521–580.

[29] Digital Science. *Dimensions*. Accessed: 19/03/2021. URL: https://app.dimensions.ai.

[30] Dorigo, M. and Di Caro, G. "Ant colony optimization: a new meta-heuristic". In: *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. Vol. 2. IEEE. 1999, pp. 1470–1477.

[31] Farmer, J. D., Packard, N. H., and Perelson, A. S. "The immune system, adaptation, and machine learning". In: *Physica D: Nonlinear Phenomena* 22.1-3 (1986), pp. 187–204.

[32] Flavell, R. B. "A new goal programming formulation". In: *Omega* 4.6 (1976), pp. 731–732.

[33] Foong, O. M. and Rahim, S. B. "Particle Swarm Inspired Timetabling for ICT Courses". In: *Applied Mechanics and Materials*. Vol. 263. Trans Tech Publ. 2013, pp. 2138–2145.

[34] Garnier, Simon, Ross, Noam, Rudis, Robert, Camargo, Pedro, A., Sciaini, Marco, Scherer, and Cédric. *viridis - Colorblind-Friendly Color Maps for R*. R package version 0.6.1. 2021. DOI: 10.5281/zenodo.4679424. URL: https://sjmgarnier.github.io/viridis/.

[35] Giagkiozis, I., Purshouse, R. C., and Fleming, P. J. "An overview of population-based algorithms for multi-objective optimisation". In: *International Journal of Systems Science* 46.9 (2015), pp. 1572–1599.

[36] Glover, F. "Tabu search—part I". In: *ORSA Journal on computing* 1.3 (1989), pp. 190–206.

[37] Huang, C., Li, Y., and Yao, X. "A survey of automatic parameter tuning methods for metaheuristics". In: *IEEE transactions on evolutionary computation* 24.2 (2019), pp. 201–216.

[38] Ignizio, J. P. *Introduction to linear goal programming*. Sage Beverly Hills, CA, 1985.

[39] Jones, D., Florentino, H., Cantane, D., and Oliveira, R. "An extended goal programming methodology for analysis of a network encompassing multiple objectives and stakeholders". In: *European Journal of Operational Research* 255.3 (2016), pp. 845–855.

[40] June, T. L., Obit, J. H., Leau, Y.-B., Bolongkikit, J., and Alfred, R. "Sequential constructive algorithm incorporate with fuzzy logic for solving real world course timetabling problem". In: *Computational Science and Technology*. Springer, 2020, pp. 257–267.

[41] Kassambara, A. *ggpubr: 'ggplot2' Based Publication Ready Plots*. R package version 0.4.0. 2020. URL: https://CRAN.R-project.org/package=ggpubr.

[42] Kempe, A. B. "On the geographical problem of the four colours". In: *American journal of mathematics* 2.3 (1879), pp. 193–200.

[43] Kendall, G. and Hussin, N. M. "A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology". In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer. 2004, pp. 270–293.

[44] Kingston, J. H. "The KTS high school timetabling system". In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer. 2006, pp. 308–323.

[45] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[46] Lewis, R., Paechter, B., and Rossi-Doria, O. "Metaheuristics for university course timetabling". In: *Evolutionary scheduling*. Springer, 2007, pp. 237–272.

[47] Marie-Sainte, S. L. "A new hybrid particle swarm optimization algorithm for real-world university examination timetabling problem". In: *2017 Computing Conference*. IEEE. 2017, pp. 157–163.

[48] Merlot, L. T. G., Boland, N., Hughes, B. D., and Stuckey, P. J. "A hybrid algorithm for the examination timetabling problem". In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer. 2002, pp. 207–231.

[49] Nelder, J. A. and Mead, R. "A simplex method for function minimization". In: *The computer journal* 7.4 (1965), pp. 308–313.

[50] Pillay, N. and Qu, R. "Assessing hyper-heuristic performance". In: *Journal of the Operational Research Society* (2020), pp. 1–14.

[51] Pirlot, M. "General local search methods". In: *European journal of operational research* 92.3 (1996), pp. 493–511.

[52] Qu, R. and Burke, E. K. "Adaptive decomposition and construction for examination timetabling problems". In: *Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications* (2007), pp. 418–425.

[53] Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., and Lee, S. Y. "A survey of search methodologies and automated system development for examination timetabling". In: *Journal of scheduling* 12.1 (2009), pp. 55–89.

[54] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2017. URL: https://www.R-project.org/.

[55] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2019. URL: https://www.R-project.org/.

[56] Romero, B. P. "Examination Scheduling in a Large Engineering School: A Computer-Assisted Participative Procedure". In: *Interfaces* (1982), pp. 17–24.

[57] Rossi, F., Van Beek, P., and Walsh, T. *Handbook of constraint programming*. Elsevier, 2006.

[58] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, PBC. Boston, MA, 2020. URL: http://www.rstudio.com/.

[59] Salas-Molina, F., Pla-Santamaria, D., Garcia-Bernabeu, A., and Reig-Mullor, J. "A Compact Representation of Preferences in Multiple Criteria Optimization Problems". In: *Mathematics* 7.11 (2019), p. 1092.

[60] Schaerf, A. "Local search techniques for large high school timetabling problems". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 29.4 (1999), pp. 368–377.

[61] Schaerf, A. and Di Gaspero, L. "Local search techniques for educational timetabling problems". In: *Proceedings of the 6th International Symposium on Operational Research (SOR-01), Preddvor, Slovenia*. Citeseer. 2001, pp. 13–23.

[62] Shi, Y. and Eberhart, R. "A modified particle swarm optimizer". In: *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE. 1998, pp. 69–73.

[63] Silva, J. D. L., Burke, E. K., and Petrovic, S. "An introduction to multiobjective metaheuristics for scheduling and timetabling". In: *Metaheuristics for multiobjective optimisation*. Springer, 2004, pp. 91–129.

[64] Spendley, W. G. R. F. R., Hext, G. R., and Himsworth, F. R. "Sequential application of simplex designs in optimisation and evolutionary operation". In: *Technometrics* 4.4 (1962), pp. 441–461.

[65] Storn, R. and Price, K. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[66] Thompson, J. M. and Dowsland, K. A. "Variants of simulated annealing for the examination timetabling problem". In: *Annals of Operations research* 63.1 (1996), pp. 105–128.

[67] Torres-Ovalle, C., Montoya-Torres, J. R., Quintero-Araujo, C., Sarmiento-Lepesqueur, A., and Castilla-Luna, M. "University course scheduling and classroom assignment". In: *Ingeniería y Universidad* 18.1 (2014), pp. 59–76.

[68] Van Hentenryck, P. "Constraint and integer programming in OPL". In: *INFORMS Journal on Computing* 14.4 (2002), pp. 345–372.

[69] Vrielink, R. A. O., Jansen, E. A., Hans, E. W., and Hillegersberg, J. van. "Practices in timetabling in higher education institutions: a systematic review". In: *Annals of operations research* 275.1 (2019), pp. 145–160.

[70] Welsh, D. J. A. and Powell, M. B. "An upper bound for the chromatic number of a graph and its application to timetabling problems". In: *The Computer Journal* 10.1 (1967), pp. 85–86.

[71] White, G. M. and Xie, B. S. "Examination timetables and tabu search with longer-term memory". In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer. 2000, pp. 85–103.

[72] Wickham, H. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

[73] Wickham, H. *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.4.0. 2019. URL: https://CRAN.R-project.org/package=stringr.

[74] Wickham, H. and Bryan, J. *readxl: Read Excel Files*. R package version 1.3.1. 2019. URL: https://CRAN.R-project.org/package=readxl.

[75] Wickham, H., François, R., Henry, L., and Müller, K. *dplyr: A Grammar of Data Manipulation*. R package version 1.0.2. 2020. URL: https://CRAN.R-project.org/package=dplyr.

[76] Wu, C.-C. "Parallelizing a CLIPS-based course timetabling expert system". In: *Expert Systems with Applications* 38.6 (2011), pp. 7517–7525.

Appendix A

Additional Results

A.1 Tables

A.1.1 Tabu search

Table A.1: Improvement from tabu search with a 30-minute time limit on random initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Mins)
First	3607.779	2738	-184.5	30.0881
Second	5359.056	4067	1407.7	30.0457
Third	5866.390	4452	1943.3	30.0125
Fourth	4320.687	3279	661.6	30.0767
Fifth	5414.392	4109	1300.4	30.0196

Table A.2: Improvement from tabu search with a 1-hour time limit on random initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Hours)
First	6114.076	4640	1297.4	1.0013
Second	5112.718	3880	2605.9	1.0015
Third	5892.732	4472	1731.7	1.0016
Fourth	5112.688	3880	2073.7	1.0004
Fifth	4837.248	3671	1245.3	1.0011

Table A.3: Improvement from tabu search with a 2-hour time limit on random initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Hours)
First	3933.368	2985	2012.4	2.0000
Second	4685.797	3556	2696.0	2.0008
Third	5862.481	4449	2721.3	2.0005
Fourth	6355.227	4823	1595.8	2.0004
Fifth	5100.818	3871	1863.9	2.0001

A.1.2 Nelder-Mead

Against tabu search

Table A.4: Improvement from Nelder-Mead with a 30-minute time limit on random initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Mins)
First	699.687	531	40.3	34.2233
Second	1478.429	1122	-4.7	32.3064
Third	994.847	755	62.7	35.4845
Fourth	884.162	671	65.4	32.0511
Fifth	560.010	425	-18.7	34.0193

Table A.5: Improvement from Nelder-Mead with a 1-hour time limit on random initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Hours)
First	3143.977	2386	164.1	1.0738
Second	1436.265	1090	24.0	1.0426
Third	1013.295	769	73.4	1.0003
Fourth	1770.951	1344	-29.9	1.0619
Fifth	1557.488	1182	-21.8	1.2176

Table A.6: Improvement from Nelder-Mead with a 2-hour time limit on random initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Hours)
First	1651.047	1253	52.1	2.2025
Second	1608.880	1221	21.4	2.1163
Third	1933.028	1467	24.4	2.0098
Fourth	2037.125	1546	34.2	2.1512
Fifth	2412.668	1831	147.6	2.0968

Ensembled with tabu search

Table A.7: Improvement from Nelder-Mead with a 30-minute time limit on tabu search initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Mins)
First	64.568	49	30.2	42.4789
Second	39.530	30	2.5	42.8798
Third	140.991	107	-1.0	41.5034
Fourth	77.743	59	-2.5	30.0153
Fifth	27.666	21	-87.7	42.1306

Table A.8: Improvement from Nelder-Mead with a 1-hour time limit on tabu search initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Hours)
First	25.036	19	6.4	1.1516
Second	23.716	18	-39.3	1.2239
Third	3.954	3	19.5	1.1494
Fourth	15.808	12	-81.4	1.1523
Fifth	23.731	18	226.9	1.1697

Table A.9: Improvement from Nelder-Mead with a 2-hour time limit on tabu search initial solutions

Run	Total Cost	Conflict Cost	Proximity Cost	Run Time (Hours)
First	6.587	5	-22.6	2.1079
Second	3.953	3	5.9	2.0000
Third	6.572	5	-287.8	2.0001
Fourth	9.224	7	5.2	2.1638
Fifth	23.715	18	-54.7	2.2269

A.2 Figures

A.2.1 Tabu search

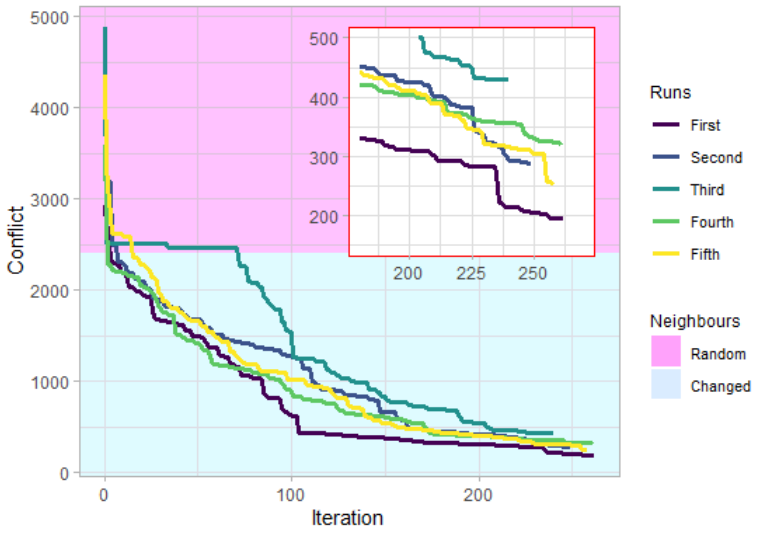

Figure A.1: Conflict value as best solutions change in tabu search runs with 30-minute time limits

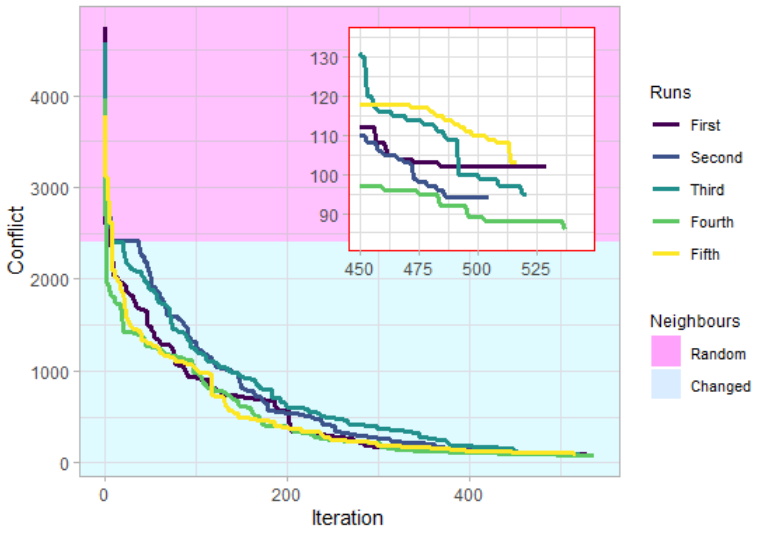

Figure A.2: Conflict value as best solutions change in tabu search runs with 1-hour time limits

A.2.2 Nelder-Mead

Against tabu search

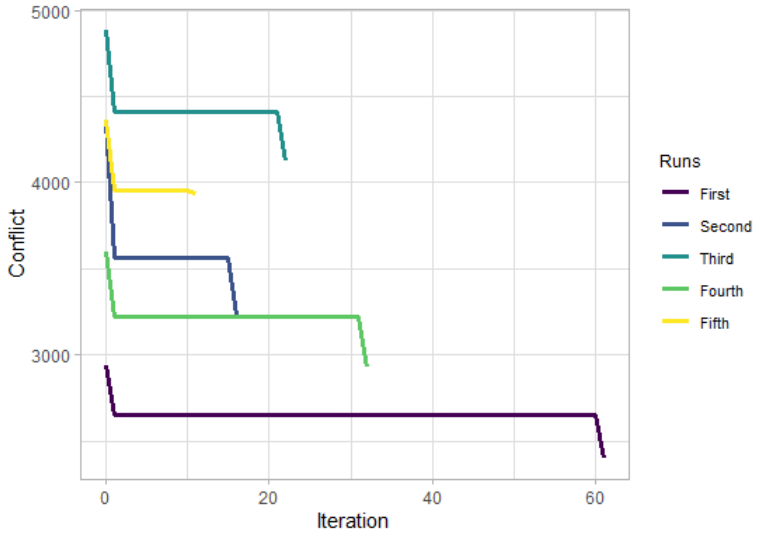

Figure A.3: Conflict value as best solutions change in Nelder-Mead runs with 30-minutes time limits

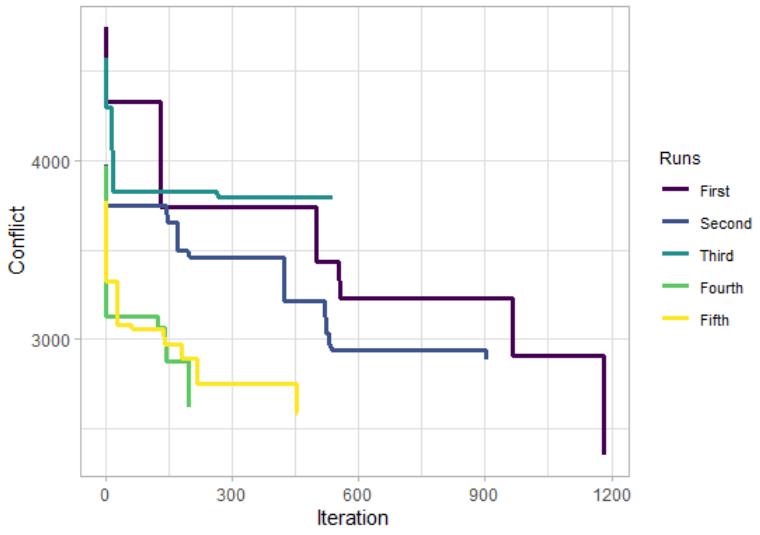

Figure A.4: Conflict value as best solutions change in Nelder-Mead runs with 1-hour time limits

Ensembled with tabu search

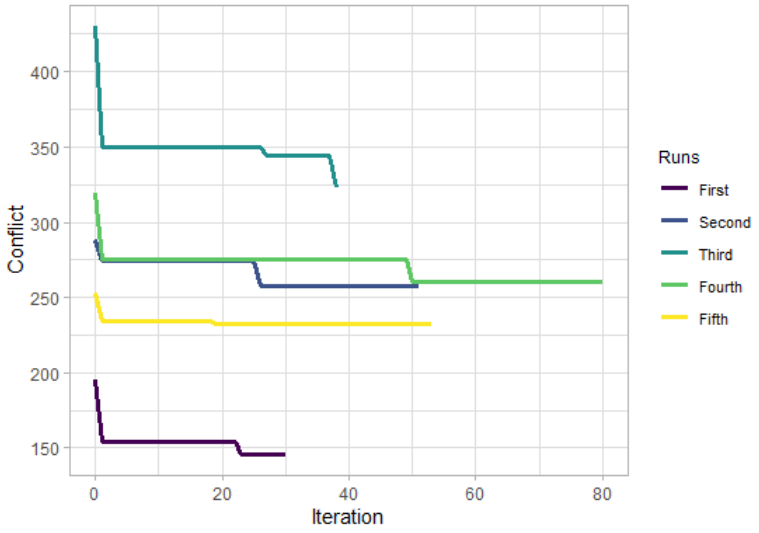

Figure A.5: Conflict value as best solutions change in Nelder-Mead runs with a 30-minute time limits on tabu search runs with 30-minute time limits

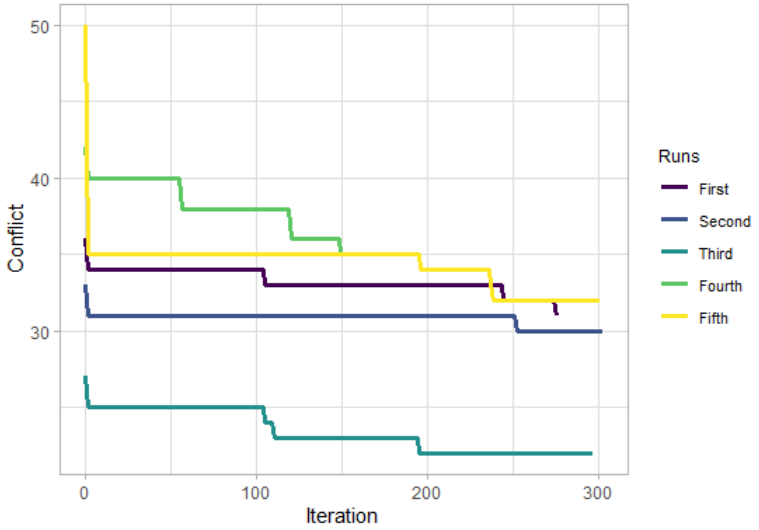

Figure A.6: Conflict value as best solutions change in Nelder-Mead runs with a 2-hour time limits on tabu search runs with 2-hour time limits