METAHEURISTICS FOR SOLVING REAL WORLD EMPLOYEE ROSTERING AND SHIFT SCHEDULING PROBLEMS

KENNETH N. REID

Doctor of Philosophy

Division of Computing Science and Mathematics

University of Stirling

July 2019

DECLARATION

I hereby declare that this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text and bibliography.

I also declare that this dissertation (or any significant part of my dissertation) is not substantially the same as any that I have submitted, or that is being concurrently submitted, for a degree or diploma or other qualification at the University of Stirling or similar institution.

I was admitted as a research student in May 2015 and a candidate for the degree of Doctor of Philosophy in 2016. This dissertation is a record of the work carried out at the University of Stirling between 2015 and 2019, under the supervision of Dr Jingpeng Li, Prof Edmund Burke, Dr Jerry Swan, Prof Amir Hussain, Dr Nadarajen Veerapen and Dr Alexander Brownlee.
I have read, and adhered to, the University's plagiarism policy, as detailed at:
`http://www.plagiarism.stir.ac.uk/`

Stirling, July 2019

Kenneth N. Reid

ABSTRACT

Optimising resources and making considerate decisions are central concerns in any responsible organisation aiming to succeed in efficiently achieving their goals. Careful use of resources can have positive outcomes in the form of fiscal savings, improved service levels, better quality products, improved awareness of diminishing returns and general output efficiency, regardless of field.

Operational research techniques are advanced analytical tools used to improve managerial decision-making. There have been a variety of case studies where operational research techniques have been successfully applied to save millions of pounds. Operational research techniques have been successfully applied to a multitude of fields, including agriculture, policing, defence, conservation, air traffic control, and many more. In particular, management of resources in the form of employees is a challenging problem — but one with the potential for huge improvements in efficiency.

The problem this thesis tackles can be divided into two sub-problems; the personalised shift scheduling & employee rostering problem, and the roster pattern problem. The personalised shift scheduling & employee rostering problem involves the direct scheduling of employees to hours and days of week. This allows the creation of schedules which are tailored to individuals and allows a fine level over control over the results, but with at the cost of a large and challenging search space. The roster pattern problem instead takes existing patterns employees currently work, and uses these as a pool of potential schedules to be used. This reduces the search space but minimises the number of changes to existing employee schedules, which is preferable for personnel satisfaction.

Existing research has shown that a variety of algorithms suit different problems and hybrid methods are found to typically outperform standalone ones in real-world contexts. Several algorithmic approaches for solving variations of the employee scheduling problem are considered in this thesis. Initially a Variable Neighbourhood Search (VNS) approach was used with a Metropolis-Hastings acceptance criterion. The second approach utilises Evolutionary Ruin & Stochastic Recreate (ER&SR) controlled by the Exponential Monte-Carlo Acceptance Criterion (EMCAC), which has only been used in the field of exam timetabling, and has not before been used within the domain of employee scheduling and rostering. ER&SR was then hybridised with our initial approach, producing ER&SR with VNS. Finally, ER&SR was hybridised into a matheuristic with Integer Programming and compared to the hybrid's individual components.

A contribution of this thesis is evidence that the algorithm ER&SR has merit outside of the original sub-field of exam scheduling, and can be applied to shift scheduling and employee rostering. Further, ER&SR was hybridised and schedules produced by the hybridisations were found to be of higher quality than the standalone algorithm. In the literature review it was found that hybrid algorithms have become more popular in real-world problems in recent years, and this body of work has explored and continued this trend. Problem formulations in this thesis provide insight into creating constraints which satisfy the need for minimising employee dissatisfaction, particularly in regards to abrupt change.

The research presented in this thesis has positively impacted a multinational and multibillion dollar field service operations company. This has been achieved by implementing a variety of techniques, including metaheuristics and a matheuristic, to schedule shifts and roster employees over a period of

several months. This thesis showcases the research outputs by this project, and highlights the real-world impact of this research.

ACKNOWLEDGMENTS

A PhD is self-evaluation on your mental resilience to stress and endurance. It is an assessment of your ability to spin a dozen proverbial plates, while new acquaintances ask you "So — will you get a *real* job after this?"

Without hyperbole - completing a PhD is an arduous undertaking. Imposter syndrome mixed with a number of weekly working hours that — if we were dubbed actual employees — would be illegal in most first world countries. This journey is a cerebral trial by fire, which is fair to say after even glancing at the mental health statistics for PhD students [1] [2] (and, to be fair, it seems most academic employees also suffer from mental and physical issues largely due to a culture of — somehow socially acceptable — hazardous work-life balance [3]). A student undertaking a PhD will tend towards solitude. Resilience against this pull was, at least for me, the key to maintaining a healthy psyche during my studies. My other advice would be to consider your work and free time like yin & yang. It's expected to work in your free time, and to procrastinate at work, but aim for a healthy balance.

The last discouraging fact I feel obliged to share: if you enter academia through a PhD, you will likely be paid less than your undergrad colleagues even though you are more highly qualified, and you are most likely to join

1 Levecque, Katia, et al. "Work organization and mental health problems in PhD students." Research Policy 46.4 (2017): 868-879.

2 Evans, Teresa M., et al. "Evidence for a mental health crisis in graduate education." Nature biotechnology 36.3 (2018): 282.

3 Sang, Katherine, et al. " 'Being an academic is not a 9-5 job': long working hours and the 'ideal worker' in UK academia." Labour & Industry: a journal of the social and economic relations of work 25.3 (2015): 235-249.

them in industry with 3-6 years less experience [4]. Regardless of the challenges a PhD presents, I find great difficulty in dissuading wide-eyed undergraduate students from following this path. It is a distinctly rewarding way to spend half a decade. A PhD is a path of self-discovery, self-discipline and a sure method of contributing to the pooled collection of all human knowledge — which in my opinion make every challenge throughout well worth it.

Family, friends and pets are the often-unsung support who replenish your spirits when low (be they metaphorical or consumable). They provide you warmth and kindness, and you'll often return this with apologies for missing social events (your corrections need completing, and you can't let your students wait any longer for their essay marks). And so...

Thank you to my wife, Alexis, for being with me at the peaks and the dips of this journey, as a constant inspiration and support. My parents (both biological and in-laws) for their empathy and succour. My close friends, Graeme & Erin, Wayne, Alexis & Creag for their support and understanding (how many times have I said no to social events? How many shows did I forgo?). And of course my sisters, thank you Aslynd, Eilidh and Skye.

For my close friends that I met (or became close to) during my PhD: Jason, Paul, Saemi and Sarah. You helped me through this sine-wave-like bipolar-symptom inflicting rollercoaster, and *somehow* made it enjoyable. We are a team, a unit, a support network, coffee addicts anonymous and debaters of all topics. I look forward to seeing you all at various conferences, while our own PhD students follow us around and wonder why we get on so well.

To undertake a PhD is to become an apprentice scholar. You follow the guidance of your supervisor and you find allies amongst your colleagues. I learned much from my supervisor (thank you, Dr Jingpeng Li), who gave me irreplaceable advice and space to flourish. My indispensable so-called

4 Moran, Karsten. https://www.nytimes.com/2016/07/14/upshot/so-many-research-scientists-so-few-openings-as-professors.html

'secondary' and 'external' supervisors (Dr Jerry Swan, Dr Nadarajen Veerapen, Dr Sandy Brownlee, Prof Edmund Burke) who went above and beyond any reasonable expectation to support me. A special thank you to Sandy for putting in weekly meetings with me, as well as all the additional support (helping with my CV, employing me for 2 months, wise counsel regarding all things research, not once moaning about knocking on his door with yet another 'quick' question!). Dr Jason Atkin also spent much of his free time assisting me with the solver implementation in Chapter 6, which I'm so very thankful for. As well as my academic supervisors I give my respect and thanks to Dr Gilbert Owusu, Alistair McCormick and Dr Mathias Kern for their support and counsel all the way from Ipswich (and Wales!). Dr Deepayan Bhowmik - thank you for employing me during the run up to my viva, even though I was a very distracted employee — and afterwards, too. I'm also grateful to the staff of CS&M, Stirling, for the friendly support - especially the girls in the office - Grace, Gemma, Linda. Thank you to Dr John Woodward for his support even though he isn't with Stirling anymore!

In aiding the retention of my sanity thank you to all the kindly people I haven't mentioned specifically: musical artists who helped block out distractions (Andy Mckee!); the otherwise welcome distractions of Reddit; baristas and other distributors of my favourite legal substance across the UK, and the not-technically-people, Tank & Brutus.

I'd like to extend gratitude to the following funders: British Telecommunications Plc and the UK's EPSRC DAASE project (grant no. EP/J017515/1).

"All we have to decide is what to do with the time that is given us."

—Mithrandir, 3019

LIST OF PUBLICATIONS

In satisfying the requirements outset in the thesis declaration: the chapters below are based on publications output during this PhD. Each row indicates the chapter and the relevant publication:

Chapter #	**Publication**
3	**Reid, K.N.**, Li, J., Swan, J., McCormick, A. and Owusu, G., 2016, December. *Variable Neighbourhood Search: A case study for a highly-constrained workforce scheduling problem.* In Computational Intelligence (SSCI), 2016 IEEE Symposium Series on (pp. 1-6). IEEE.
4	**Reid, K. N.**, Li, J., Veerapen, N., Swan, J., McCormick, A., Kern, M., & Owusu, G. (2018, September). *Shift Scheduling and Employee Rostering: An Evolutionary Ruin & Stochastic Recreate Solution*. In 2018 10th Computer Science and Electronic Engineering (CEEC) (pp. 19-23). IEEE.
5	**Reid, K.N.**, Li, J., Brownlee, A., Veerapen, N., Swan, J., Kern, M. and Owusu, G. 2019, July. *A Hybrid Metaheuristic Approach to a Real World Employee Scheduling Problem*. In GECCO'19: The Genetic and Evolutionary Computation Conference 2019. ACM, 201

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

aco Ant Colony Optimisation

al Apprenticeship Learning

alns Adaptive Large Neighbourhood Search

asp Answer Set Programming

aspccgtk Answer Set Programming Combinatory Categorial Grammar Toolkit

cpu Central Processing Unit

cp Constraint Programming

emcac Exponential Monte-Carlo Acceptance Criterion

er&sr Evolutionary Ruin & Stochastic Recreate

fso Field Service Operations company

ga Genetic Algorithm

gi Genetic Improvement

hc Hard Constraint

hcs Hard Constraints

hthsa Hybrid Taguchi Harmony Search Algorithm

ike Insider Knowledge Engine

ip Integer Programming

ipa Iterative Pattern Approach

mip Mixed Integer Programming

milp Mixed Integer Linear Programming

or Operational Research

pso Particle Swam Optimisation

rcpsp Resource Constrained Project Scheduling Problem

rp Roster Pattern

rps Roster Patterns

sa Simulated Annealing

sc Soft Constraint

scs Soft Constraints

smbo Sequential Model-Based Optimization

smac Sequential Model-based Algorithm Configuration

ui User Interface

vns Variable Neighbourhood Search

vnsh Variable Neighbourhood Search-Based Heuristic

vrp Vehicle Routing Problem

vrptw Vehicle Routing Problem with Time Windows

1

CHAPTER 1 - OVERVIEW

1.1 INTRODUCTION

In the field of Operational Research (OR), the optimised allocation of resources to activities is of widespread interest. Resources in this context can be machines, vehicles, raw materials, employees and so forth. Whenever resources are not utilised or organised correctly there is an irrevocable associated loss of efficiency. The most obvious loss in this perspective is economic loss, but other concerns are commonplace and often more important. Examples include more environmental damage, from say, additional carbon released into the atmosphere from non-optimised routing patterns for motor vehicles, or excessive plastic wrapping on mass produced goods. Whether it be economical, environmental, tactical improvements in military environments, time efficiency for airports or meeting various objectives in designing a product, OR is the search for techniques and algorithms which can yield high quality results with minimal cost. Such techniques include: mathematical optimisation and metaheuristics.

This thesis concerns a problem provided by a real-world telecommunications company seeking more efficient means of scheduling employees. The bulk of the content in this thesis is derived from works previously published in various conferences and is in use in some manner by said company. This chapter will provide an introductory insight into the objectives this thesis was

written to meet, initial hypothesis, research questions, and an overview of the structure of this thesis.

1.2 HYPOTHESIS

The main hypothesis of the work described within this thesis is as follows:

Thesis Hypothesis *By using a variety of known metaheuristics and Integer Programming (IP) techniques, a real-world shift scheduling & employee rostering problem can be solved to feasibility and a higher level of quality than the baseline, as provided by a field service operations company.*

This hypothesis is challenged by addressing the following research questions.

1. Massive search spaces make finding optimal solutions in a realistic time frame infeasible. Using evolutionary algorithms, is it possible to find a solution of improved quality within the *ideal* time of "5 to 10" seconds, or within the "maximum runtime" of 10 minutes? [1]

2. The metaheuristic ER&SR has previously only been used in exam timetabling. Can this be used to solve the employee scheduling problem which has a much larger search space?

3. Hybrid metaheuristic / matheuristic algorithms have been gaining in popularity for real-world problems. Can a hybrid algorithm tackle the employee scheduling problem provided to a higher quality than the component algorithms composing the hybrid algorithm?

4. One of the greatest difficulties in OR, and management in general, is finding methods to mitigate employee dissatisfaction in regard to change.

1 The time limits are requirements by the company providing the employee and problem data — during testing of algorithms in the various chapters, more time is used than this, but consideration is also always given to these requirements.

Are there constraints which can be put in place to minimise change while still improving quality in terms of overall objective targets and fitness?

1.3 THESIS STRUCTURE

In order to provide a structurally sound volume of work which can both be read sequentially as well as used as reference material with the table of contents, this thesis will be structured as follows:

- Chapter 1 — In this chapter the purpose of research is stated in the form of a hypothesis and an introduction.
- Chapter 2 — An overview of the problem itself, and the distinction between the sub-problems, is provided.
- Chapter 3 — The literature in the field is reviewed and showcased to solidify the purpose of this research and highlight the niche wherein the research written in this thesis lies. The research questions answered in this thesis are compared to other work in the field.
- Chapter 4 — Personalised Scheduling & Rostering is one of the main problem areas tackled with this research. In this chapter a case study regarding the initial problem is defined and solved with VNS to provide preliminary results of this problem and investigate the usage of a meta-heuristic approach to the problem.
- Chapter 5 — A continuation of work in the previous chapter, the problem is redefined to be a more closely aligned with the Field Service Operations company (FSO)s real-world problem. This is then solved with ER&SR.

- Chapter 6 — Roster Pattern scheduling is another problem tackled in this thesis. In this chapter the problem is defined and tackled with a new hybrid algorithm of ER&SR with VNS.

- Chapter 7 — A continuation of work in the previous chapter, the Roster Pattern (RP) problem is redefined to be more closely aligned with the FSOs real-world problem. This is then solved with a new matheuristic which, through hybridisation, combines the strengths of ER&SR with IP.

- Chapter 8 — This penultimate chapter gives a brief insight into the real-world applications that utilise the algorithms described in this thesis.

- Chapter 9 — The final chapter of this thesis refers to research questions introduced in Chapter 1 and answers them, evaluates the hypothesis and describes future work.

Additionally the appendices are as follows:

- Appendix A
  - Other efforts made to solve the problems in this thesis are shown here. These are not research in the strictest sense, and don't quite fit into other chapters. These approaches may provide insight into the problem and how the problem is initially probed.

- Appendix B
  - All graphics (such as charts, plots, etc.) in this thesis are displayed in this appendix in full page format for easier viewing.

2

CHAPTER 2 - PROBLEM DESCRIPTIONS

This chapter forms the first contribution chapter of this thesis. Further, this chapter provides both an overview of the problems tackled in this thesis as well as mathematical formulations of the problems, as interpreted from the descriptions and data provided by the FSO.

The problems tackled throughout this thesis are all personnel scheduling problems. Over the four content chapters (Chapters 4 through 7) there are two sub-problems tackled, which are termed the personalised scheduling problem, and the roster pattern scheduling problem. In this thesis, Chapters 4 and 5 describe methods for solving an instance of the personalised scheduling problem, and Chapters 6 and 7 describe methods for solving an instance of the roster pattern scheduling problem.

The structure of this chapter is as follows: in section 2.1 the terminology used throughout this thesis regarding the sub-problems of personalised scheduling and roster pattern scheduling are defined. Section 2.2 then describes the personalised scheduling problem in mathematical detail, followed by a discussion of the problem. Next, section 2.3 follows a similar structure with an integer programming subsection to define the roster pattern scheduling problem, followed by a discussion.

2.1 PROBLEM TERMINOLOGY

In this section the terms 'personalised scheduling' and 'roster pattern scheduling' are described, to make sense of the two overarching problems tackled across chapters 4 to 7.

2.1.1 *Personalised Scheduling*

The term "personalised scheduling" is one used internally at the FSO. This is considered an idealistic future of scheduling within the company, where employee schedules are flexible and malleable without the limitations of Roster Patterns (RPs).

The flexibility of personalised scheduling means that any shift can be modified in any respect: length, start time, end time, or skill requirement. Personalised scheduling has many useful qualities: it allows flexibility in meeting demand requirements, and theoretically allows employees to change shift preferences easily (e.g. if an employee wants fewer Saturday shifts, or wants to start later on Fridays). However, this approach is unlikely to generate schedules where employees have a standard routine, which RPs ensure. An example of what a personalised schedule could look like for a single employee compared to a roster pattern schedule can be seen in Table 2.1 (please note the colour contains no additional information in this table and is only used to emphasise structural differences). Personalised schedules are more likely to be without pattern or repeating structural elements, unlike roster patterns. This is the main reason for the two distinct problems - the personalised scheduling is considered a potential future use, but the roster pattern schedules are considered for a near-time replacement.

Personalised Schedule Example

	M	T	W	T	F	S	s
W1	9-5		8-4	10-6	9-5	9-5	
W2		9-5	8-4	8-4	7-3	9-5	
W3	8-4		9-5	9-5	8-4		
W4	8-4	9-5	9-5	9-5	9-5		

Roster Pattern Schedule Example

	M	T	W	T	F	S	s
W1	9-5	9-5	9-5	9-5	9-5		
W2	9-5	9-5	9-5	9-5		9-5	
W3	9-5	9-5	9-5	9-5	9-5		
W4	9-5	9-5	9-5	9-5		9-5	

Table 2.1: Single Employee Problem Type Example

2.1.2 *Roster Pattern Scheduling*

An RP is a set of shifts over days, weeks or months. These are not directly related to specific calendar dates, and instead is an abstracted period of shifts which can be applied to any calendar day, week or month. The characteristics of an RP include the set of shifts with daily shift start times, shift end times, shift length, but does not include any information about demand, skills required nor expected on these shifts.

It is not possible to modify any element of an RP. For example, in a scenario where fitness would improve if it were possible to have an employee begin at 10am instead of 9am, this is not allowed, unless another RP exists that the employee could be assigned to which begins at 10am. This constrained problem is a reflection of a real-world scheduling problem at the FSO.

While it is not possible to directly modify shift times for an employee, there are three other variables to consider. The below three variables can be shown clearly in Table 2.2.

1. Which RP to which an employee is assigned to can be modified. This can have drastic effects; from changing the number of days in which an employee works per week, to number of hours per week, day shift to night shift, etc. In order to maintain contractual considerations, there is a set of roster pattern preferences per employee. In some workplaces, any RP can be used, in others, preferences are required. This is taken into consideration in section 6.1. The number of RPs available can vary drastically from region to region. In Table 2.2, the leftmost column contains four example RPs with example shift times per day over two weeks. Changing RP can have large effects on the resulting schedule.

2. Of the weeks that exist in a pattern, the week the pattern begins on can be modified. So instead of, say, having weeks 0, 1, 2, 3, the employee

	M	T	W	T	F	S	s	M	T	W	T	F	S	s
RP1	9-5	9-5	9-5	9-5	9-5			9-5	9-5	9-5	9-5	9-5		
RP2	9-5	9-5	9-5			9-5		9-5	9-5	9-5	9-5			
RP3	8-4		8-4		8-4	8-4			8-4		8-4	8-4	8-4	
RP4	9-5	8-4	1-5			1-5			8-1		9-5	9-5	9-5	

Table 2.2: Example of Variable Data in Roster Patterns

could instead work 1, 2, 3, 0. Further, say an employee works a standard 9am-5pm shift (Monday to Friday) except every second week the employee works 10am-6pm on Mondays. If the employee's starting week is modified, it would change which hours they are working every Monday, possibly improving the overall solution and meeting objectives. In Table 2.2 the days are labelled in the first row, over two weeks. While RP1 would be unchanged by changing the starting week from the first week to the second, all the other RPs would be affected by this change.

3. The skills employees use on each shift are also mutable. Skills employees are assigned for can drastically affect the fitness of a roster. This is because for demand to be met, an employee needs to be assigned, regardless of stated skill levels. However, employees have a set of skill preferences which should be utilised as much as possible. So, in modifying which skills they use on each shift, there is increased likelihood that employees are working shifts they are either best suited for, or are using skills they wish to use more to improve their abilities. In Table 2.2, the skills are differentiated with colour. The names of the skills do not matter. Without changing which RP or starting week an employee is working, there are still a large number of changes that can be made to a schedule by simply manipulating which skills are used.

A combination of all three variables will be modified to improve rosters. However, there can be restrictions on these variables, defined by Hard Constraints (HCs) and Soft Constraints (SCs).

2.2 PERSONALISED SCHEDULING

2.2.1 *Integer Programming Model*

The following IP model refers to the problem tackled in Chapters 4 & 5.

Parameters	
I	Set of employees available
I_t	Subset of employees that work A1, A2 and A1/A2 shift patterns respectively. I_t where $t \in 1, 2, 3$ representing A1, A2 and A1/A2
J	Set of days representing days in a week (1-7, i.e. Monday - Sunday)
W	Set of all weeks in the scheduling period. $W = W_1 + W_2$, where W_1 represents the weeks with odd indices and W_2 the weeks with even indices
K	Set of shift types equal to 1 or 2, which stand for day shift and late shift respectively.
Z	Set of skills, in order of preference when used in reference to an employee, in order of demand priority when used in relation to days or weeks
α_{ij}	The maximum number of DAY types over the scheduling period for an employee
β_{ij}	The maximum number of DAY shift types over the scheduling period for an employee

γ_{ij}	The maximum number of LATE shift types over the scheduling period for an employee
ζ_{ij}	The minimum number of DAY types allowed over the scheduling period for an employee.
η_{ij}	The minimum number of DAY shift types over the scheduling period for an employee
μ_{ij}	The minimum number of LATE shift types over the scheduling period for an employee
l_{ik}	The number of occurrences of shift type k for each employee i over the scheduling period. $l_i \leqslant 5\|W\|$
k_i	The least common shift type for employee i: $k_i \in K$. $k_i = 1$ if $l_{i1} < l_{i2}$, 2 otherwise
R	Set of the day index for each occurrence of k_i for employee i over the scheduling period. $\|R\| = l_{ik}$
o_i	Ideal intermittence between the i-th employee's k_i: $o_i = \mathrm{int}(7\|W\|/l_{ik_j})$
p_{ij}	Number of shifts worked for employee i on day j. $p_{ij} = \sum_{k \in K} \sum_{w \in W} x_{ijkw}$, and $p_{ij} > 1$
m_{ik_i}	The number of occurrences of the least common shift type k_i for each employee i on day j. $m_{ik_i} < \|W\|$
q_i	Ideal day intermittence for employee i: $q_i = \lceil \|W\|/m_{ik_i} \rceil$, $q_i \in \mathbb{N}/\{0\}$
t_{jw}	Number of employees allocated per j across W:

	$t_{jw} = \sum_{i \in I} \sum_{k \in K} x_{ijkw}, \forall j \in J, w \in W$
u_{jw}	Daily allocated employee spread fitness score per j across W, $u_{jw} \in \{0, 1\}$
v_j	Mean allocated employees per j, $v_j = \left\{ \sum_{i \in I} \sum_{k \in K} \sum_{w \in W} x_{ijkw} / \lvert W \rvert, \forall j \in J \right\}$
$v_j \pm \delta$	Acceptable upper / lower bounds of v_j per j across W. $\delta \geqslant 0$, δ is an input to the algorithm, which depends on business preferences at the time.

Decision variable x_{ijkw} is 1 if an employee i is assigned shift type k for day j in week w, 0 otherwise, for all employees on day shifts on all weeks on all shifts, defined as:

$$x_{ijkw} = 0 \text{ or } 1, \forall i \in I, j \in J, w \in W, k \in K \tag{2.1}$$

Slack / surplus variables are minimized and are used as the positive or negative deviations from individual goals, defined as:

$$s^1_{zjkw} \geqslant 0, s^2_{zjkw} \geqslant 0, \forall z \in Z, j \in J, k \in K, w \in W \tag{2.2}$$

$$s^3_{ijw} \geqslant 0, s^4_{ijw} \geqslant 0, \forall i \in I, j \in J, w \in W \tag{2.3}$$

$$s^5_{ti} \geqslant 0, \forall r_t \in R, i \in I \tag{2.4}$$

$$s^6_i \geqslant 0, \forall i \in I \tag{2.5}$$

$$s^7_j \geqslant 0, \forall j \in J \tag{2.6}$$

Subject to:

$$\text{HC1} \quad \sum_{k \in K} x_{ijkw} \leqslant 1, \forall i \in I, j \in J, w \in W \tag{2.7}$$

$$\text{HC2} \quad \sum_{k \in K} \sum_{w \in W} x_{ijkw} \leqslant \alpha_{ij}, \forall i \in I, j \in J \tag{2.8}$$

$$\text{HC3} \quad \sum_{k\in K}\sum_{w\in W} x_{ijkw} \geqslant \zeta_{ij}, \forall i \in I, j \in J \tag{2.9}$$

$$\text{HC4} \quad \sum_{w\in W} x_{ij1w} \leqslant \beta_{ij}, \forall i \in I, j \in J \tag{2.10}$$

$$\text{HC5} \quad \sum_{w\in W} x_{xij1w} \geqslant n_{ij}, \forall i \in I, j \in J \tag{2.11}$$

$$\text{HC6} \quad \sum_{w\in W} x_{ij2w} \leqslant \gamma_{ij}, \forall i \in I, j \in J \tag{2.12}$$

$$\text{HC7} \quad \sum_{w\in W} x_{ij2w} \geqslant \mu_{ij}, \forall i \in I, j \in J \tag{2.13}$$

$$\text{HC8} \quad \sum_{j\in J}\sum_{k\in K} x_{ijkw} = A_1, \forall i \in I_1, w \in W \tag{2.14}$$

$$\text{HC9} \quad \sum_{j\in J}\sum_{k\in K} x_{ijkw} = A_2, \forall i \in I_2, w \in W \tag{2.15}$$

$$\begin{aligned}\text{HC10} \quad &\sum_{j\in J}\sum_{k\in K} x_{ijkw} = A_1, \forall i \in I_3, w \in W \\ &\sum_{j\in J}\sum_{k\in K} x_{ijkw} = A_2, \forall i \in I_3, w \in W\end{aligned} \tag{2.16}$$

$$\text{SC1} \quad \sum_{i\in I} x_{ijkw} + s^1_{zjkw} - s^2_{zjkw} = d_{zjkw}, \forall z \in Z, j \in J, k \in K, w \in W \tag{2.17}$$

$$\text{SC2} \quad \sum_{k\in K} [x_{ijkw} + x_{i(j+1)kw}] + s^3_{ijw} - s^4_{ijw} = 0, \forall i \in I, j \in J, w \in W \tag{2.18}$$

$$\text{SC3} \quad r_{t+1} - r_t + s^5_{ti} \leqslant o_j, \forall r_t \in R, i \in I \tag{2.19}$$

$$\text{SC4} \quad [(\sum_{p=1}^{p-\epsilon} w_{(p+\epsilon)_{ij} w_{p_{ij}}}) + 1]/p_{ij} + s^6_i \leqslant q_i, \forall i \in I \tag{2.20}$$

where $p + \epsilon$ is next p_{ij} after current p_{ij} across W

$$\text{SC5} \quad (\sum_{w\in W} u_{jw})/|W| + s^7_j = 1, \forall j \in J \tag{2.21}$$

$$u_{jw} = 1 \ \text{ if } \ t_{jw} \in [v_j - \delta, v_j + \delta], \text{ else } u_{jw} = 0$$

2.2.2 *Problem Description*

In this subsection, the personalised scheduling problem is defined in further detail. There are two chapters in this thesis which solve an instance of the personalised scheduling problem: Chapter 4 and Chapter 5. There are slight differences in the personalised scheduling problem, as it was refined by the FSO over time. As such, subsection 2.2.2.1 refers to both chapters, and 2.2.2.2 refers to the refinements in Chapter 5.

2.2.2.1 *Personalised Scheduling Problem Description*

The term 'shift' refers to a length of time within a 24-hour period in which a single employee can be allocated. Shifts can only exist within a single day, and cannot span between two days (that is, a shift cannot breach the midnight barrier). There is also an additional facet to this problem, in that as well as rules about how employees are assigned shifts, there are requirements about shift types. These are known as shift rules, and govern rules for the generation of all shifts, regardless of which employee is allocated to each.

The constraints for this problem consider both employee preferences and demand requirements, which are input by representatives of employee unions. This means that the schedules produced are tuned to meet company requirements as well as the wants and needs of employees. This is an approach which is not implemented in the workforce yet — but this research is being used for simulation purposes and for considering future goals in the industry, and is a part of a suite of tools offered by the research team to internal and external clients in the FSO.

There are many employee scheduling examples which are found to be NP-Hard problems [1]. Knowing that the optimal solution to the problems presented cannot be found in polynomial time, a 'good enough' quality solu-

tion is acceptable, if also feasible. The challenge is therefore to find the best quality solution within a time constraint. The FSO requested "ideally 5 seconds run time" of the algorithm to generate employee schedules, which is a very short time frame for such a large-scale problem.

The data used for this study included around 25,000 employees, however the actual number needing scheduled at any one point was up to around 200. This is due to the shifts being scheduled at a regional level, and employees (for the purpose of this study) never leave a region. The FSO selected a region which was deemed typical for demand requirements and resource characteristics, which was used for this study.

Below are the key characteristics of the problem, defined in partnership with the FSO:

1. Due to the real-world data being provided by a UK based FSO, UK Employment Law must be adhered to.

2. The objective of meeting demand includes not simply providing manpower to cover number of employees required per shift, but skills required per shift, and per day, while still meeting HCs.

3. The shift creation process is structured by shift rules. Shift rules contain parameters and requirements.

The problem is most easily defined with a set of HCs and SCs. HCs are a set of requirements which must be clearly defined, as they are the rules by which a generated schedule is deemed accepted or rejected. As such the nature of these constraints are Boolean. If a single Hard Constraint (HC) fails, then the schedule is deemed infeasible. A list of HCs for this problem can be found in Table 2.4. In this list, 'day type' means a day of week which exists one or more times in a schedule, for example all instances of Monday, or all instances of Tuesday. 'DAY' shift refers to a simplified and undefined shift of working

hours. 'LATE' shifts refer to shifts with later starting and end times than 'DAY' types.

Hard Constraints	
HC1	Employees must have a either 0 or 1 shifts per day.
HC2	Employees have a maximum number of day types (Monday - Sunday) in the scheduling period. A possible minimum is no restriction or an integer which represents the enforced lower threshold (> 0).
HC3	Employees have a minimum number of day types (Monday - Sunday) in the scheduling period. A possible minimum is no restriction or an integer which represents the enforced lower threshold (> 0).
HC4	Employees has a maximum number of DAY shift type in the scheduling period. A possible maximum is no restriction, no DAY shift type allowed (0) or an enforced upper threshold (> 0).
HC5	Employees has a minimum number of DAY shift type in the scheduling period. A possible minimum is no restriction, or an integer which represents the enforced lower threshold (> 0).
HC6	Employees has a maximum number of LATE shift type in the scheduling period. A possible maximum is no restriction, no LATE shift type allowed (0) or an enforced upper threshold (> 0).
HC7	Employees has a minimum number of LATE shift type in the scheduling period. A possible minimum is no restriction, or an integer which represents the enforced lower threshold (> 0).
HC8	Employees with the A1 day shift pattern type must be allocated to A1 shifts every week.

HC9	Employees with the A2 day shift pattern type must be allocated to A2 shifts every week.
HC10	Employees with the A1/A2 day shift pattern type must be allocated to A1 shifts every second week and A2 shifts every other week.

Table 2.4: Hard Constraints for Personalised Scheduling & Rostering

SCs are used to describe a schedules quality, as opposed to its feasibility. Technically any schedule can be assessed for quality, but it is only worthwhile to consider the quality of a feasible schedule. When an algorithm is running and attempting to decide which of various schedules to keep or to improve upon, the SCs are used to judge each solution by a set of fitness values. The SCs for this problem are defined in Table 2.5.

Soft Constraints	
SC1	Shifts should cover the maximum amount of demand requirement they possibly can. For example, if a shift is 8 hours long, then ideally the shift would cover $8 * 1$ hours' worth of demand.
SC2	Two consecutive rest days for employees every week.
SC3	Employees least common shift type (DAY or LATE shift) should be spread as evenly as possible over the scheduling period to allow a high as possible mean number of days between the least common shift type allocations.

SC4	Employees least common day types (Monday - Sunday) should be spread as evenly as possible over the scheduling period to allow a high as possible mean number of weeks between the less common day type allocations. Not used if employee is scheduled for one, less than one, or the maximum number of day type(s), as ideal intermittency has already been achieved.
SC5	The number of employees allocated to each day type (Monday - Sunday) should be spread as evenly as possible over the scheduling period.

Table 2.5: Soft Constraints for Personalised Scheduling & Rostering - Variable Neighbourhood Search

2.2.2.2 *Personalised Scheduling Problem Refinements*

The problem refined below is tackled in Chapter 5. First, the objectives were rewritten into more focused points:

1. Due to the real-world data being provided by a UK based FSO, UK Employment Law must be adhered to.

2. The 'meeting demand' objective now considers skills required per shift and per day, as well as a sufficient number of employees to satisfy shift requirements.

3. The shift creation process is structured by shift rules. Shift rules contain parameters and requirements.

The HCs and SCs have been modified in collaboration with the FSO. The changes were made to better suit the needs of the company, and removing peripheral constraints which were deemed either unimportant or potentially negative. Of particular note is the removal of the concept of day shifts and

late shifts. In this chapter, the shifts can be scheduled to any hour of a single 24-hour day (without overlap between days). This has both a positive and negative effect: the search space has increased in size, but this also provides more flexibility for potential employee allocation slots, meaning there is more potential for improved schedules.

The HCs are as follows:

- HC1: Employees 0 or 1 shifts per day.
- HC2: Employees have a minimum and maximum number of allocations per day of week (Monday - Sunday).
- HC3: Employees can have shift patterns A, B or A/B. A/B means in one week they must do A shifts, in the next B shifts. Shift pattern is synonymous with the number of shifts an employee works in a 7 day week, Monday to Sunday.
- HC4: Employees must be allocated to shifts equal to or after their shift time minimum, and before or equal to their shift time maximum.
- HC5: Employees must be allocated to shift lengths longer than or equal to their minimum shift length allowed, and shorter than or equal to their maximum shift length allowed.
- HC6: Shifts must begin within their shift rules designated start and end times.
- HC7: Shift rules specify a minimum requirement of number of shifts across the scheduling period. A possible minimum is no restriction or an enforced lower threshold (> 0). Every shift rule also specifies a maximum requirement of shifts across the scheduling period. A possible maximum is no restriction, no shift rules of this type (0) or an enforced upper threshold (> 0).

- HC8: Shifts must be greater than or equal to the shift rules minimum shift length and be lesser than or equal to its shift rules maximum shift length, if specified.

The SCs are as follows:

- SC1: Shifts should cover the maximum amount of demand requirement they possibly can.
- SC2: Employees should be allocated to shifts that meet their first skill preference.
- SC3: Employees should be allocated to shifts that they meet the skill requirement for.
- SC4: Two consecutive rest days for every employee on every week.

2.3 ROSTER PATTERN SCHEDULING

2.3.1 *Integer Programming Model*

In this section a mathematical model is provided to clearly define the problem constraints of the roster pattern scheduling problem in a precise manner. The following IP model refers to the problem tackled in Chapters 6 & 7.

Please note HC3 is handled solely by the metaheuristic part of the hybrid algorithm in Chapter 7, not in the IP section, and is therefore not described in this section.

Parameters	Description
I	Set of employees available
T	Set of all possible roster patterns $t \in T$
t_{jk}	Hour and days allocated to employees assigned roster pattern t
$\|t\|$	Cardinality of hours across all days in roster pattern t
J	Set of days representing all days in a schedule $j \in J$
K	Set of hours $k \in \{0, ..., 23\}$.
B	Set of all skills
F	Set of all original roster allocations $f_{it} \in F$ where i is an employee and t is the original roster pattern
n	Number of roster pattern changes that have occurred. This is calculated as: $n = \sum_{f_{it} \in F} \sum_{t \in T} y_{it} \ \ where \ \ t \neq f_{it} \forall i \in I$
n_{max}	The maximum number of roster pattern changes that are allowed to occur. If there is no parameterised maximum, $n_{max} = \|I\|$
G	Set of all roster preferences $g_{it} \in G$
q_c	Weighting of each Soft Constraint (SC) where $0 \leqslant q_c \leqslant 1, c \in \{1, 2, 3, 4\}$ $\sum_{1}^{4} Q_c = 1$
D_i	Set of skill preferences $d_{ib_s} \in D_i$ where s is the skill preference index for employee i $1 \leqslant s \leqslant 5$

	$\|b_1\| = 1$ $\|b_2\| \geqslant 0$ $\|b_3\| \geqslant 0$ $\|b_4\| \geqslant 0$ $\|b_5\| \geqslant 0$
e	Original total demand as input. Count of all hours and skills where there is demand.

Decision variable w_{jkib} is 1 if an employee i is assigned to hour k on day j and skill b, 0 otherwise, for all employees on day shifts on all weeks on all shifts, defined as:

$$w_{jkib} = 0 \text{ or } 1, \forall j \in J, j \in K, i \in I, b \in B \tag{2.22}$$

Decision variable x_{dkb} is equal to a value of 0, or higher if there is demand currently required on hour k and day d for skill b. This exists for all hour, day and skill combinations:

$$x_{dkb} = 0 \text{ or } 1, \forall d \in D, k \in K, b \in B \tag{2.23}$$

Decision variable y_{it} is 1 if an employee i is assigned roster pattern t for all employees on day shifts on all weeks on all shifts, 0 otherwise. This is defined as:

$$y_{it} = 0 \text{ or } 1, \forall i \in I, t \in T \tag{2.24}$$

Decision variable z_{jki} is 1 if an employee i is available for potential allocations on hour k for day j, 0 otherwise, for all employees, all weeks on all shifts, defined as:

$$z_{jki} = 0 \text{ or } 1, \forall j \in J, k \in K, i \in I \tag{2.25}$$

Target function:

$$\text{Min} \quad ((SC1 * q_1) + (SC2 * q_2) + (SC3 * q_3) + (SC4 * q_4)) \tag{2.26}$$

Subject to:

$$\text{HC1:} \sum_{t \in T} y_{it} = 1, \forall i \in I \tag{2.27}$$

$$\text{HC2:} \quad n \leqslant n_{max} \tag{2.28}$$

$$\text{HC4:} \sum_{g_{it} \in G} y_{it} = 1, \forall i \in I \tag{2.29}$$

$$\text{HC5:} \sum_{b \in B} w_{jkib} \leqslant 1, \forall j \in J, k \in K, i \in I \tag{2.30}$$

$$\text{HC6:} \sum_{b \in B} w_{jkib} \geqslant y_{it_{jk}}, \forall i \in I, t_{jk} \in t \tag{2.31}$$

$$\text{HC7:} \sum_{k \in K} \sum_{b \in B} \sum_{j \in J} w_{jkib} = \sum_{t \in T} \sum_{|t| \in |T|} y_{it} * |t|, \forall i \in I \tag{2.32}$$

$$\text{HC8:} \sum_{b \in B} w_{jkib} = z_{jki}, \forall j \in J, k \in K, i \in I \tag{2.33}$$

The following SCs are used in the aforementioned target function (Eq 2.26):

$$SC1 = \sum_{j \in J} \sum_{k \in K} \sum_{b \in B} x_{dkb} \tag{2.34}$$

$$SC2 = \sum_{j \in J} \sum_{k \in K} \sum_{i \in I} \sum_{d_{ib_s} \in D_i} w_{jkib} \tag{2.35}$$

$$SC3 = \sum_{j \in J} \sum_{k \in K} \sum_{i \in I} \sum_{d_{ib_s} \in D_i} \sum_{b \in B} w_{jkib} * s \tag{2.36}$$

$$SC4 = \sum_{i \in I} \sum_{b \in B} \left(1 - \sum_{j \in J} \left| \frac{\frac{\sum_{k \in K} w_{jkib}}{e} - \frac{\sum_{j \in J} \sum_{k \in K} w_{jkib}}{e}}{|J|} \right| \right)^* \tag{2.37}$$

* Note $|J|$ is cardinality of days, while the other usage of the vertical lines denotes absolute value

2.3.2 *Problem Description*

In this subsection, the roster pattern scheduling problem is defined in further detail. There are two chapters in this thesis which solve an instance of the roster pattern scheduling problem: Chapter 6 and Chapter 7. There are slight differences in the personalised scheduling problem, as it was refined by the FSO over time. As such, subsection 2.3.2.1 refers to both chapters, and 2.3.2.2 refers to the refinements in Chapter 7.

2.3.2.1 *Roster Pattern Scheduling Problem Description*

Below are the key characteristics of the problem, defined in partnership with the FSO:

1. Due to the real-world data being provided by a UK based FSO, UK Employment Law must be adhered to. This means a maximum average of 48 hour working weeks.

2. The objective of meeting demand includes not simply providing manpower to cover number of employees required per shift, but skills required per shift, and per day, while still meeting HCs.

3. The shift creation process is structured by shift rules. Shift rules contain parameters and requirements.

The HCs are as follows:

- HC1: Employees must have exactly one RP.

- HC2: An optionally specified maximum number of RP changes can occur.

- HC3: An optionally specified maximum number of starting week changes can occur.

- HC4: If RP preferences are provided, only RPs preferred by each employee can be explored.

HC1 is hard-coded; an employee cannot be given more than or less than one RP outside of a single iteration of ER&SR or VNS (meaning that employees do have RPs removed, but they are then given another RP, or potentially the same one again).

HC2 and HC3 can optionally have the maximum number of changes set to no maximum, zero (meaning no changes of this type allowed) or a specific number. An example could be maximum of ten RP changes allowed. When ten have occurred, only those modified RPs can be changed. If one of these is returned to the original pattern, then the current number of changes is reduced to nine, and another employee's pattern could potentially be changed. In this chapter HC2 and HC3 are set to infinite, meaning they are always satisfied.

HC4 can be set to on or off. If set to off, any employee could potentially be given any RP. This is generally not realistic in many regions as certain RPs are created for special circumstances, specific contractual requirements, etc. It would also mean employees could be given the incorrect number of working hours per week. However, in the case that available RPs are interchangeable, then this is not an issue.

For most regions, employees have a number of preferred RPs, and HC4 would be toggled on. The preferred patterns will always meet contractual obligations. These patterns are agreed upon by both employee and regional managers. Every employee has at least one preferred RP which is their current pattern. There is no maximum number of preferred RPs, other than the number available in the pool, which is dependent on how many have been created by managers in the past. In the region used for this chapter there are around 400 potential RPs employees could have in their preferred patterns list.

The quality of a solution is judged with SCs. This is undertaken by considering elements that are important for the quality of life for engineers, for meeting demand and for meeting customer satisfaction. These are important, but do not break any laws or contractual requirements if not met. The following SCs are considered to judge the quality of a solution:

- SC1: The total demand requirement met by all shifts should be maximised
- SC2: The number of employees assigned shifts that require one of their skills should be maximised
- SC3: The number of shifts using employees preferred skills should be maximised
- SC4: Variation in the percentage of demand met per hour, day and skill should be minimised.

2.3.2.2 *Roster Pattern Scheduling Problem Refinements*

The characteristics of the roster pattern scheduling problem are refined below. The problem refined below is tackled in Chapter 7.

1. Due to the real-world data being provided by a UK based FSO, UK Employment Law must be adhered to. This includes maximum working hours allowed per week, number of rest hours in between shifts, etc.
2. The employees described in the data have a variety of skills. Demand is provided for the problem in skill-based manner. Demand may fluctuate greatly across the scheduling period. The objective of meeting demand includes not simply providing manpower to cover number of employees required per shift, but also skills required per shift (and per day) while still meeting HCs.

The refined HCs are as follows:

- HC1: Employees must have only one roster pattern.
- HC2: An optionally specified maximum number of roster pattern changes can occur.
- HC3: An optionally specified maximum number of starting week changes can occur.
- HC4: If roster pattern preferences are provided, only roster patterns preferred by each employee can be explored.
- HC5: Employees can use a maximum of one skill per hour per shift.
- HC6: Employees must be allocated to shifts which are described within the roster pattern they are allocated to.
- HC7: Employees must only work the number of hours described by the roster pattern they are allocated to.
- HC8: Employees must only be assigned to hours where they are not already assigned.

The refined SCs are below:

- SC1: Employees should be assigned to shifts which have the highest demand coverage in the set.
- SC2: Employees should have the skill required on each shift to which they are allocated.
- SC3: Employees skill preferences should be met when possible.
- SC4: Variation in the percentage demand met per day, per skill, should be as small as possible.

Combined weighted constraint violations are used to judge solution fitness. Specific weightings are provided by the FSO; in this problem all weightings are equal (25% per SC).

3

CHAPTER 3 - LITERATURE REVIEW

This chapter identifies the niche this research fills in the literature. First is an explanation of metaheuristics to set the scene for the methodologies chosen to tackle problems in this thesis. After the explanation of metaheuristics and significant literature relating to their real-world application in Section 3.1, the related works to the problems tackled in this thesis — the shift scheduling & employee rostering problem — are reviewed in Section 3.2. Penultimately there is an overview of the tuning algorithms used in this thesis and how they have been used in some other related works in Section 3.3. This chapter ends with concluding remarks in Section 3.4, placing work explored in this thesis neatly into the field.

3.1 INTRODUCTION TO METAHEURISTICS

Before delving into an explanation of metaheuristics directly, first it is prudent to define key terminology. An optimisation problem is one where it may be fairly simple to produce a feasible output (that is, one which meets all crucial requirements), but is difficult to find an output which improves quality (quality is very much problem-specific, but can include minimisation of resources used, efficiency of output, time taken to solve, surplus, etc.). The term 'optimisation problem' is synonymous with the term 'search problem', as finding an optimal solution to a complex problem is a search through many, and potentially all, possible solutions. The various possible outcomes from applying a heuristic to a problem create a search space. A different algorithm applied to the same

problem creates a different search space. 'Solution' is a phrase used in the literature which depending on context can have two meanings — one being the heuristic being used, the other being the output of a heuristic. When a solution (in this case meaning output from a heuristic) is being modified and fitness values considered, nearby solutions are known as neighbours, and less similar solutions are considered more distant neighbours. Fitness is a term used to mathematically describe the quality of a solution (such as profit, meeting demand, preventing waste, etc.). Finally, constraints are generally bundled into two groups - HCs and SCs. HCs are Boolean, and absolutely required for a solution (output from a heuristic) to be considered viable, or feasible. If even a single HC fails out of thousands then a solution is deemed infeasible. HCs are often implemented to adhere to laws, contracts or economic requirements. SCs are used for measuring quality of feasible solutions.

The term 'metaheuristic' was first coined by Glover [2] in the celebrated paper introducing tabu search. According to Genreau et al [3], metaheuristics are high-level procedures designed to solve the problem of entrapment within local optima when exploring complex solution spaces, particularly spaces that employ one or more neighbourhood structures to define admissible moves when transitioning between solutions. Metaheuristics are detached from problem definitions: they are general solvers that can be applied to a variety of distinct problems, spanning theoretical and real-world settings.

Instead of requiring a new heuristic for every problem, metaheuristics can be used to create solutions to a variety of problems. Burke et al suggest metaheuristics are used when standard heuristic procedures have failed, or would take too long (or expensive hardware) to run [4]. While alternative approaches, such as mathematical programming or exhaustive search will ultimately find the global optimum to a given problem, they will often fail to do so in a practical length of time with the finite resources of a typical machine (this

depends on the problem and resources available, but can tend into thousands of years run-time). Many mathematical programming methodologies struggle to provide good quality feasible solutions in short time bursts, similarly with exhaustive search. However, the major drawback of metaheuristics is their inability to definitively give optimal solutions, or reduce the search space [5], so, the main use of metaheuristics is to provide fairly good quality solutions fairly quickly.

There are thousands of derivations of metaheuristics. For example, a metaheuristic framework for Simulated Annealing (SA) can be customised for different problem domains (e.g. Travelling Salesperson, Vehicle Routing, Scheduling etc.) via the provision of domain-specific heuristics for solution neighbourhood and solution quality. Popular metaheuristics include SA [6], VNS [7], tabu search [2], genetic algorithms, memetic algorithms [8] and many others.

3.1.1 *Variable Neighbourhood Search*

VNS is a methodology which is also commonly used in optimisation problems, first proposed by Mladenovic et al [7]. VNS considers the incumbent solution and its neighbours, and only moves if improvements are made. VNS first searches for the local optimum, then when no further improvements can be found locally, a perturbation occurs to escape the local search space, and begin anew. This continues until a predetermined termination criterion is met. There have been a huge number of variations made upon the original VNS metaheuristic, frequently with the purpose of customising to specific problem characteristics and required run times.

VNS has been successfully utilised for a variety of real-world problems. Many such papers have been classified and categorised in the thorough literature review by Hansen et al in 2008 [9], followed by an updated review in 2010 [10].

However, no such real-world focused literature review of VNS exists in almost decade hence. As such, a select number of the more recent and prominent papers will be described here.

Both Vehicle Routing Problem (VRP)s and Vehicle Routing Problem with Time Windows (VRPTW)s have been tackled by VNS as well as a variety of algorithms in the literature, and are an important focus of research, as efficient logistical schedules and pathways for transportation can lessen our carbon footprint; save money for all involves parties and improve demand satisfaction. Binhui et al present a VRPTW problem is tackled using VNS with Compound Neighbourhoods [11]. The main difference between this and traditional VNS approaches is that the independent operators have different lengths of exchange segments, as well as the traditional operation position. The results show that local search tends to be less effective on the considered datasets, and overall promising results when compared to the Solomon benchmarks described in [12]. Another example of a VRP problem being solved using VNS can be found in a paper by Drake et al [13] — however, the methodology is entirely different. By using a grammatical evolution hyper-heuristic to generate a construction heuristic, this approach was found to, in the best cases, find the optimal number of vehicles required. These tests were also run on the aforementioned Solomon benchmarks [12].

A recent paper described a Variable Neighbourhood Search-Based Heuristic (VNSH) approach to solve a theoretical problem modelled after real-world instances of Resource Constrained Project Scheduling Problem (RCPSP). In this paper it was found that traditional schedule generation schemes (including serial & parallel) were ineffective compared to the enhanced activity swapping among varied neighbourhood that is accomplished with the VNSH method for single resource breakdown, and similar results were found for multiple resource breakdown.

A prominent paper in the literature by Pisinger et al [14] established a derivative algorithm known as Adaptive Large Neighbourhood Search (ALNS). The proposed algorithm works on structurally differentiating neighbourhoods, while standard VNS algorithms (such as basic VNS, best improvement VNS, reduced VNS) operate on one type of neighbourhood with variable depth. ALNS was proposed to tackle five different variants of the VRP, which is possible due to the metaheuristic nature of ALNS.

3.1.2 *Evolutionary Ruin & Stochastic Recreate*

ER&SR was first described in Li et al [15] then further explored as a theoretical framework [16]. ER&SR is derived from the traditional Ruin and Recreate algorithm by Schrimpf et al [17]. Ruin and recreate functions as described by the name - first destroying some or all of a solution, then rebuilding the solution. What is ruined and how depends on the problem encoding, and the problem itself. Additionally, what is reconstructed and how depends greatly on the same information. The flexibility of this algorithm is what makes it a metaheuristic. ER&SR is an extension upon Ruin and Recreate, which is differentiated by the addition of traditionally 'evolutionary' techniques. Specifically, the ruin phase includes both a selection and mutation subphase, and the recreate phase has a deterministic rebuild which attempts to somewhat stochastically reintroduces solution components, pending approval from the underlying improvement heuristic.

Prior to work described in this thesis, ER&SR had only been used to solve a real-world complex exam timetabling problem by Li et al [16]. In this thesis ER&SR is used on an employee scheduling problem for the first time (excluding papers declared in the publications section of the beginning content of this

thesis). This is described in Chapter 5, and ER&SR is also implemented in Chapter 6 and Chapter 7.

No other works, other than those described in this thesis, have yet been published on ER&SR, which makes a case for future work on this algorithm.

3.1.3 *Simulated Annealing*

SA is often described as a classical technique in the field of metaheuristics. SA is a metaheuristic which is named after the metallurgical technique of annealing. When moulding steel, iron, glass or another material, metallurgists or glass smiths heat the substance to a high temperature, making the material malleable. Similarly, in SA, at a high temperature exploration of the search space is more likely, and at a lower temperature exploitation of the current neighbourhood is more likely. This allows a customisable approach to exploring a search neighbourhood, based on the parameters of cooling rate and temperature.

This metaheuristic is first described in 1953 by Metropolis et al [18] as an adaptation from the metropolis-hastings algorithm, which itself is best described in a later paper by Hastings et al [19].

SA has been used to solve a variety of real-world problem instances. For example, in the 1992 paper by Teodorovic et al [20] an SA algorithm is implemented to solve an instance of the VRP. While this paper does not concentrate on a real-world problem, the authors do refer to potential real-world VRP problems using SA, including planning the collection of materials such as plastics, glass and textiles in urban areas. The authors also imply that the lengthy run time of this algorithm could be an issue if used in real-time for VRPs, however, if being planned in advance and a longer runtime is not an issue, SA can provide satisfactory routing plans.

SA has also been applied to the academic problem of project allocations. In this instance by Chown et al [21] there are a number of projects that students can indicate a preference for. There is a maximum workload per staff member, and SA is used to maximise student project allocation preferences, while ensuring no single staff member is overworked.

Efficient mobile network routing is a pressing real-world problem for most countries across the globe. In a paper by Saha et al [22], the authors describe the purpose for research in this area as being for lower latency, higher message delivery rates, and a lower routing overhead. The authors explain that in many scenarios a greedy methodology is utilised, causing entrapment in local optima. Thus the authors introduce a new routing protocol which they dub 'SeeR'. SeeR is based on an SA algorithm, in order to improve the aforementioned issues. It was found that SeeR either was near-equivalent in performance to other protocols tested, or better (in particular latency was improved).

SA is used in Chapter 6 and Chapter 7 as a useful comparison to the results produced by the other described algorithms.

3.1.4 *Hybrid Approaches*

As well as standalone algorithms being used for real-world problems, there are many examples of hybridisation of algorithms. Hybridised algorithms are generally preferred when multiple features from various component algorithms are desired. Such features can include solving speed, finding local optima, escaping local optima, or transforming the search space. Hybrid algorithms are somewhat ill-defined in the literature, and can be viewed liberally (all algorithms contain some operators or components which are used in other algorithms), or conservatively (only named algorithms which are combined

can be defined as hybrid algorithms). In order to minimise uncertainty, the conservative definition is preferred in this thesis.

A well-known paper in the VRP literature was written by Osman et al [23]. In this paper, SA is used in hybridisation with Tabu search. This hybridisation is useful, as both SA and Tabu search have the advantage of being able to leave local optima in further search of the global optimum, but Tabu search prevents returning to already known solutions. This means that this hybridisation allows further exploration with SA, while reducing computational runtime with Tabu search. Further, there is a freeing strategy in place, so that if enough time has passed, it is possible to return to know solutions to continue searching. This ensures that the search does not halt in an inescapable solution, and does not needlessly retrace steps. This hybrid was tested on seventeen standard problems from the VRP literature. At the time of publication in 1993, this hybrid algorithm performed significantly better than other published solutions in both solution quality and number of vehicles required.

According to Yildiz et al [24], hybrid algorithms have shown outstanding efficiency and reliability in regard to real-world engineering optimisation problems. In [24], the authors combine harmony search (inspired by the search a jazz musician undertakes seeking greater musical harmony) with the Taguchi method (which has a proven track record in engineering optimisation). This hybrid is dubbed Hybrid Taguchi Harmony Search Algorithm (HTHSA), wherein the strengths of each approach can be fed back and forth - that is, the harmony search can assist the Taguchi method, and vice versa. There is a highly promising result from this paper, this hybrid approach outperforms all other tested methodologies (including standalone Genetic Algorithm (GA) and standalone harmony search) in both cost minimisation, maximisation of profit, and in minimising iterations of function evaluations until convergence.

Hybrid algorithms have also been utilised in military environments. The authors, Lee et al, hybridise Ant Colony Optimisation (ACO) with a genetic algorithm to improve weapon-target systems [25]. The authors found this approach to have the best performance when compared to existing search algorithms for this problem, in simulations.

3.1.5 *Matheuristics*

Matheuristics are a special type of hybrid algorithm, where a metaheuristic is hybridised with a mathematical programming algorithm. Matheuristics can be defined as an intersection between deterministic and non-deterministic solvers. Implementations vary, but often a metaheuristic will provide a partially complete solution and allow the exact solver to complete the rest of the solution. As the size of problems exact solvers can complete in an acceptable time frame are small to medium size search spaces, a matheuristic can exploit the strengths of both metaheuristics to traverse vast search spaces, and exact solvers to quickly find local optima, and ideally the global optimum.

Applications of matheuristics to VRPs have been compiled into two literature reviews, as of writing: the 2010 Rich VRP paper [26] and the more thorough 2014 paper by Archetti et al [27]. This paper claims it is likely that due to more accessibility in programming languages which support matheuristic approaches; the frequency of matheuristic solutions to VRPs are likely to increase.

The parallel machine scheduling problem has also been tackled with a matheuristic approach by Fanjul et al [28]. Central to this problem is scheduling a scarce resource among a number of machines. This paper adapts a Mixed Integer Linear Programming (MILP) program originally described in [28] into 3 separate matheuristic approaches, based on successful strategies in the

literature. The mathematical programming solver used in this paper was IBM ILOG-CPLEX 12.6. The matheuristic approaches successfully improved the number of jobs solved by the machines. Further, using matheuristic approaches showed a clear improvement from the standalone CPLEX solver.

More recently, in 2017, a real-world irregular bin packing problem based in Spain was solved using a matheuristic approach. Combining CPLEX with various procedures for solving the assignment of pieces to bins. There are also strategies in place for instances of infeasibility. The non-deterministic solutions include Bin Packing with Greedy Decisions, First Fit algorithm and Partial Bin Packing. Computational result highlights include lower computational effort and better performance on average among the various implementations (though some had higher costs related with better results).

Matheuristics are of relevance to this thesis, as Chapter 7 utilises IP with ER&SR to solve an instance of a real-world scheduling problem. This literature shows that hybrid algorithms, in particular matheuristics, have a proven track record in a variety of real-world problems and further investigation is warranted.

3.2 shift scheduling & employee rostering problems

OR is a discipline in which real-world problems are first modelled numerically or programmatically. Then mathematical heuristics are applied to the model to find solutions which can be used in the real-world. In this section a variety of these problems, and their solutions, are showcased and discussed. As the field of OR has been active for many decades, originating in World War II in order to aid in military endeavours. This section will discuss the aspects most applicable to this thesis, and focusing on the most predominant techniques used to solve employee scheduling problems in recent history.

Examples of employee rostering & shift scheduling problems come from a variety of sectors and industries, including telecommunications technician scheduling [29], airline crew scheduling [30], military personnel scheduling [31], nurse scheduling [5], telephone call centre scheduling [32] and many others [33].

The field of employee rostering has been extensively surveyed: one of the first surveys of the state of the art was produced by Baker [34] in 1976. In 2004 Ernst et al produced a review of applications, methods and models [33], which not only successfully highlighted some of the most influential research in the field, but also correctly predicted future trends. Such trends include employee scheduling in airlines, which has since been further examined by several reports [35] [36], and the need for more robust frameworks to tackle the complexity of these problems. The sub-topic of evolutionary scheduling was reviewed by Hart et al in [37], providing thorough coverage of the years 1985 - 2004. The extensive 2013 review 'Personnel Scheduling: A Literature Review' by Van Den Bergh et al [38], gives an exploration of the field, in which papers are categorised by criteria such as personnel characteristics, decision types, shift flexibility, coverage constraints (both hard and soft), whether or not under- and over- staffing is permitted, by skills, and more. More recently Arturo Castillo-Salazar et al produced a workforce scheduling and routing problems literature review [39]. There have been several other notable domain specific literature reviews, such as home care routing and scheduling by Fikar et al [40] and physician scheduling by Erhard et al [41].

Scheduling vast numbers of employees while taking demand, location, skill sets, preferences and contract types into consideration is a complex task which cannot be completed optimally (or even close to optimally) manually for a large real-world workforce. It is often impossible to exhaustively search for an optimal solution in a timely fashion [4]. As such other approaches have been

developed by operational engineers, computer scientists and employee managers to generate optimal or close-to-optimal solutions, e.g. using evolutionary computation approaches [42]. Improving labour costs by only a few percent could prove very beneficial, since this is often the main expenditure for companies according to Van den Bergh [38]. In the field of employee rostering there is much motivation for obtaining high quality employee rosters. Managing the shifts of a large number of employees is a complex task — optimisation of this process is a necessity for efficient use of resources.

Rather than the expensive (and often impossible) alternative of analysing every potential roster, an evolutionary algorithm can sample the solution space for a result which is often at least close-to-optimal, within a reasonable time.

There are many examples of evolutionary algorithms and nature-inspired approaches to solving real-world employee scheduling problems, from a variety of industries.

3.2.1 *Variable Neighbourhood Search*

With a proven record of usage for over two decades as of writing, VNS is a trusted and frequently used algorithm for real-world employee scheduling problems across a variety of sectors.

In home healthcare, for example, VNS has been successfully applied by Pinheiro et al [43]. This particular scheduling problem is also a routing problem, provided by an industrial partner of the authors. This VNS instance works from an initial solution and searches locally within multiple neighbourhoods in order to find improved solution quality. Then, to escape local optima, VNS shakes the solution, possibly incurring a worse solution, at the end of every iteration. Four configurations of VNS were considered during testing, with various local search mechanisms and differing numbers of shaking neighbour-

hoods. The results showed that the hill-climbing local search variant of VNS performed worse than the proposed algorithm. Further, the proposed VNS algorithm reached the optimal solution, whenever the optimal solution was known.

Solos et al take on a different shift scheduling problem [44], in that tank trucks as well as their drivers are scheduled using a version of VNS known as Effective Stochastic VNS. This algorithm introduces three new swap mechanisms. Two of the new swap functions are based on what is known as a core swap method, and themselves are named sorted successive row swap and random row swap. The final novel swap is collapsible window stochastic swap. In total, there are nine potential swaps that the implementation can utilise, three of the nine are novel. The study concludes that this methodology successfully found solutions which reduce costs to the oil company by reducing overtime, while minimising changes to driver schedules.

Nurse rostering is a well-studied problem with important impacts on the real-world. Maenhout et al apply VNS to an instance of the nurse rerostering problem to solve sub-problems to optimality [45]. The subproblems are divided into nurse schedule, day roster and combination of days. However, the final roster quality is dependent on the order in which the sub-problems are solved, and a dynamic guiding order is recommended in order to add consideration to the holistic solution as well as the local. This concept of restructuring the problem into sub-problems is not explored thoroughly in the literature, and this example was found to improve upon current (in 2011) state of the art heuristics.

VNS is implemented in works described in Chapter 4, controlled by Metropolis-Hastings Acceptance. VNS is also used in a hybridised state with ER&SR, described fully in Chapter 6.

3.2.2 *Simulated Annealing*

SA was described in Section 3.1. In this section examples of shift scheduling and employee rostering problems solved by SA are described. SA has been used less in for solving personnel scheduling in recent years. However, it has often been used as an acceptance criterion [46][47]. The cases where it has been used for employee scheduling are discussed in this subsection.

In 2004 a US-based patent was filed for what is dubbed a "Dynamic Workforce Scheduler" [48], which considers schedules a workforce while considering employee preferences, job skills, minimum and maximum hours per employee, rules regarding employment of minors, break times, union contracts, and more. At the core of this patent is an SA algorithm which provides consideration to the various constraints while scheduling employees.

Brusco et al aim to minimise cost by creating only as many shifts as are required in [49]. One of the novelties of this paper is the constraint relaxations: up until this point, most of the literature assumed certain restrictions were required, which increased caused an excess of labour, and cost. A novel 'intelligent improvement routine' is implemented which prevents unnecessary lingering run-time. Near-optimal solutions were found for all problems tested.

SA is used in Chapter 5 and Chapter 6 as a useful comparison to the results produced by the other described algorithms.

3.2.3 *Hybrid Approaches*

Hybridised approaches have much potential and a proven track record in employee scheduling problems. This is due to employee scheduling problems often being found to be NP-Hard (as described by Brucker et al in [50]), and hybrid approaches have component algorithms selected which can swiftly

converge and explore. Blum et al [51] note that hybrid approaches are loosely defined, of which critics disapprove. However, the authors also state that this yet ill-defined term provides space for research into previously unexplored directions.

An example of a hybrid algorithm for shift scheduling can be found in the 2018 study by Pour et al [52]. A preventive signalling maintenance crew scheduling problem is solved using a Constraint Programming (CP) / Mixed Integer Programming (MIP) framework, where the CP implementation allowed a feasible starting point for the MIP application to improve upon.

In an example of a nurse rostering problem, Rahimian et al [53] utilise VNS in hybridisation with IP. A greedy heuristic is employed to tackle the initial problem and create a base solution, which is successfully improved upon iteratively with VNS deep-embedded with IP.

In the third edition of the Handbook of Metaheuristics by Raidl et al [54], there is a useful chapter on metaheuristic hybrid techniques and the various approaches explored in the literature. Additionally, as found in this literature review, the authors have also found hybrid approaches to be increasingly popular in regard to real-world problems. Another useful survey on hybrid approaches was written in 2011 by Blum et al [51], which can give historical context to the now more popular methodologies which come under the moniker of 'hybrid metaheuristics'.

In the Burke et al 2008 paper [55], VNS is hybridised with heuristic order, and found that this hybrid approach significantly outperformed a commercial genetic algorithm on the same data-set. In another paper of Burke et al [5], VNS was again hybridised but with IP instead of heuristic ordering. However, this new approach was compared to the previous hybridisation and found improvements. This was, in part, the inspiration towards hybridisation of VNS with ER&SR in Chapter 5 and the hybridisation of ER&SR with IP in Chapter 6.

3.2.4 *Matheuristics*

Metaheuristics are generally viewed as black-box optimisers, but there are a set of optimisers which come under the label of grey-box optimisers, which are a hybridisation of mathematical programming (such as MIP) and metaheuristics, known as matheuristics.

In [56] the authors solve a nurse rostering problem based in an Italian private hospital using a VNS based matheuristic, combined with CPLEX and separately combined with XPRESS. The solutions showed that the proposed MathVNS procedure were equal to or better quality than those produced by the XPRESS solver standalone. Moreover, this was found to be a solution of satisfactory quality for use within the hospital.

A real-world data set problem provided by the French Operations Research Society is solved by Chen et al [57]. The presented algorithm is known as 'Forward Approximate Dynamic Programming Algorithm', which uses a mathematical algorithm in R in conjunction with the dynamic programming component to produce a schedule for technicians. Interestingly, R is capable of providing near instantaneous solutions, the dynamic programming element requires longer run-time. However, the main focus of this study is not to improve demand on a day-by-day basis, but to manoeuvre the workforce into a better position for future customer service requests. This Approximate Dynamic Programming based approach outperforms the so-called 'myopic approach' across all variants including the baseline problem, workforce disruption and new task type introduction.

Matheuristic approaches appear to be increasing in popularity in recent years, which is unsurprising due to the improved quality of quick solutions, when compared to base metaheuristics.

In this thesis the problems solved by the FSO are unlike most scheduling problems due to the two main reasons. First, the FSO is involved in ongoing negotiations with the employees union to ensure any changes made are fair and meet expectations and requirements of the employees. As such, solutions must provide a flexibility constraint to ensure change can be limited. Secondly, due to the FSO having tens of thousands of existing employees with existing contracts, all contracts must be respected as they currently exist. This means that tens of thousands of constraints must be implemented as is. The simplest mechanism for this is a direct import of RPs, which contain all shifts, days per week, and so on. This provides a novelty to this research, as the unique large set of constraints require creative considerations to provide new solutions to optimising employee rosters, and problems of this nature with these specific constraints have not been found within the literature.

3.3 PARAMETER TUNING

Due to a substantial impact on the quality of solutions produced by algorithms of which have their parameters 'tuned' (the metaphor refers to guitar, piano, or other stringed instruments having tension altered to correct tones), parameter tuning has been rightly thoroughly investigated in a variety of real-world fields and employee scheduling. There are two broad approaches: statistical analysis and search-based optimisation. In the paper by Eiben et al [58], the authors state various benefits of parameter tuning from their research, including salient superior parameter values, insight into algorithmic performance, information about the importance and interactions between parameters. They note that automated parameter tuning is the transition from competitive to scientific testing.

3.3.1 *Taguchi Methods*

The Taguchi methods are a series of statistical tests and methodologies, designed by Genichi Taguchi [59] originally for use in improving the quality of manufactured products. In recent years, the Taguchi methodology has been used in a variety of industries and research disciplines to solve a variety of problems, including marketing, search design, machining problems, and so on. The Taguchi method of design has also been applied to various search problems, some of which are described below.

In a publication by Wang et al [60], the Taguchi design methodology was used to tune a PSO algorithm applied to a motor design problem. It was found that when Particle Swam Optimisation (PSO) is tuned by the Taguchi design method, optimal solutions can be found in benchmark functions, and the authors state that generally throughout testing, parameter design with Taguchi method on PSO obtains the global optimum and is more robust.

Both SA and VNS are tuned using the Taguchi method in Maleki et al's paper [61] to solve an assembly flow-shop scheduling problem. The results of this experiment show that the VNS implementation outperforms the SA algorithm in terms of solution quality, but not in terms of CPU time to solve. When using a tuner to compare two algorithms like this paper proposes, the tuner provides an additional level of fairness to experimentation, as these algorithms have stochastic elements, where default parameterisation may hinder the search greatly depending on how direct the influence of parameters are on each algorithm. Allowing each algorithm to run with closer to optimal run-time parameters provide a comparison of best-case scenarios for each algorithm, allowing a fairer comparison of goodness.

3.3.2 *irace Software Package*

The renowned software package `irace` [62] allows automatic configuration of parameters, regardless of implementation language or operating system. Due to the ease of use, the well documented user guide and automatic tuning nature of the software, there has been much attention given to this tuner in the literature since its release. `irace` is based on the f-race algorithm, introduced in [63][64].

One such example of `irace` being used as a parameter tuner for a search problem can be found in Mascia et al's paper [65]. This approach implements a grammar-based generation of stochastic local search heuristics using `irace`. The authors found that replacing the evolutionary algorithm in the grammatical evolution with `irace` provided an improved result.

The automated design of algorithms is a fairly recent field of study which includes generating mutation operators for GAs by Woodward et al [66] and for creating human competitive programs alongside Genetic Improvement (GI) by Haraldsson et al [67]. A niche use for `irace` can be found in the automated design of algorithms in Yarimcam's 2014 paper [68]. In this problem items are packed into bins, and require immediate decisions of where to go to prevent overflow. As this implies, an optimal solution is unimportant, the best solution available in the short time frame is the most important. `irace` is used to train an algorithm which produces heuristics to solve the problem. Generally `irace`, in the literature, is used for tuning algorithms which produce solutions directly, not for tuning a method which produces heuristics itself. Two classes of train-test sessions are considered in this study, categorised into *small* ($N = 10^4$) and *large* ($N = 10^5$). These two sessions are compared to one another, as well as to a GA and an Apprenticeship Learning (AL) algorithm.

Policies produced by `irace` tuning are found to be effective and produce competitive results against all but the GA method for policy production.

3.3.3 *Other Approaches*

In Rangel et al's paper [69] an SA algorithm is tuned for an employee timetable scheduling problem for call centres using covering arrays. A covering array is a combinatorial object which is often used in software and hardware testing. It is a method employed to find a fair set of test cases which cover a variety of potential situations or parameters that the hardware or software will encounter in deployment. Covering arrays have been reused for parameter tuning algorithms, due to the nature of considering a wide variety of inputs. In the aforementioned study, the covering array allows consideration of some parameters configurability in relation to other input variables. The authors recommend further usage of covering arrays for parameter tuning, however, there is a strong case to be made that covering arrays are substandard.

Statistical inference methods have also been used as parameter and operator tuning tools in Petrovski's publication [70]. It was found that factor tuning has several disadvantages, including mistakes making sub-optimal or even unsuccessful runs; trial and error approaches are impractical for real-world scenarios due to factor interaction and optimal settings for one problem are not necessarily the best for another. However, the methodology introduced in this paper show that in situations where these methodologies are 'worth the effort' the aforementioned problems can be addressed.

Hutter et al [71] produced a parameter tuner known as Sequential Model-based Algorithm Configuration (SMAC) [1], which is considered state-of-the-art for model-based automatic configuration according to Caceres et al [72]. SMAC

1 http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

is based on Sequential Model-Based Optimization (SMBO) and builds a random forest (a machine learning algorithm, itself built on the principles of regression and classification) model to predict optimal parameters or settings. SMAC has been successfully used as a surrogate models for aiding `irace` configurations [72], tuning the specialised Answer Set Programming (ASP) toolkit known as Answer Set Programming Combinatory Categorial Grammar Toolkit (AspCcgTk) to provide linear improvements in run-time by Buddenhagen et al [73], improving quality of pathing algorithms in a robotics study by Burger et al [74], and hundreds of other uses.

However, an alternative 2013 study by Arcuri et al [75] with over a million experiments across various parameter tuning settings and various problems has shown that there is not always a benefit to parameter tuning. As the no free lunch theorem (first described in [76]) implies, there is a direct cost to any algorithm. Arcuri et al state, and through various experiments can fairly argue, it may not be worth the effort tuning the parameters of an algorithm. Instead, simply using the default values may be in the long run more efficient. This is particularly true for test data generation, but otherwise there is an implied 'grey area'. That being said, if an algorithm is to be repeatedly used on similar datasets, tuning is viable and certainly worthwhile. In this thesis various real-world problem instances are solved, and created with the intention of future use by the FSO. As such, parameter tuning is worthwhile.

Parameter tuning is a methodology investigated in this thesis. In Chapter 5 the Taguchi method is investigated as a parameter tuning tool for use with ER&SR. In Chapter 6 as well as in 7, following best practice, `irace` is used to tune parameters and is found to be a useful and powerful software for this purpose. For further reading, the 2019 paper by Huang et al provides a recent insight into trends regarding automatic tuning of metaheuristics [77].

3.4 CONCLUSION

This thesis contributes to the field of operational research by investigating the use of metaheuristics and hybrids incorporating them for optimising employee schedules for use within telecommunications. This chapter has provided an overview of recent as well as critical developments in the field of employee scheduling in Section 3.2, an overview of how algorithms implemented in this thesis are used in the literature for other real-world problems in Section 3.1 and an overview of how the tuning algorithms utilised in this thesis have been used in other problems in Section 3.3.

This literature review highlights the importance of hybrid approaches for finding state of the art results for real-world problems, and notes the increase in number of hybrid approaches for employee rostering problems. This thesis focuses on using a combination of previously unseen characteristics in OR literature to solve a real-world employee scheduling problem, and compares them to the individual components as standalone tools for this problem.

4

CHAPTER 4 - PERSONALISED SCHEDULING & ROSTERING - VARIABLE NEIGHBOURHOOD SEARCH

This chapter provides preliminary results from the personalised scheduling problem defined in Chapter 2. The purpose of this chapter is to 1) ascertain whether the shift scheduling & employee rostering problem can be solved using a known metaheuristic and 2) provide insight into further research to be taken in proceeding chapters. This chapter first describes the implementation of a Greedy Algorithm and VNS used to tackle the problem, initially described in Chapter 2, is then described in section 4.1. Following this are the results and a conclusive discussion.

4.1 IMPLEMENTATION

The order of events for this section follow the steps of the algorithm. First, the Greedy Algorithm is explained, which provides an initial solution for the algorithm to work upon. Then the VNS implementation is defined, which is controlled by a Metropolis-Hastings Acceptance criterion, that is also described. The flow of events is visualised in Fig 4.1.

4.1.1 *Greedy Algorithm*

The first step of this solution is to create a set of neighbourhoods which satisfy the HCs using a Greedy Algorithm. This will likely produce a poor quality

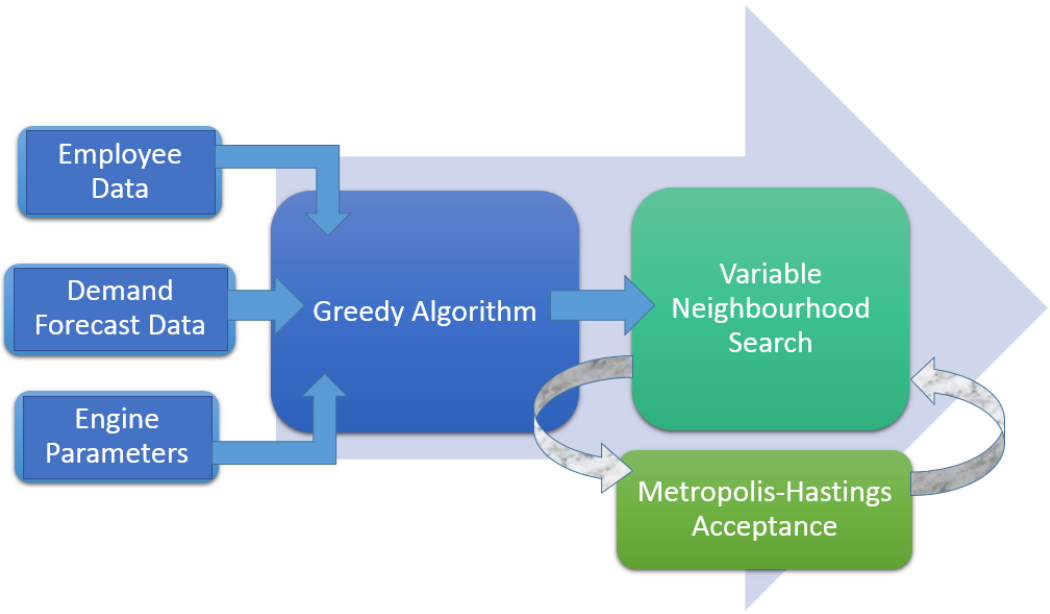


Figure 4.1: Diagrammatic Overview of Algorithm Flow

solution in terms of meeting SCs, but this is only to act as a starting point for the VNS algorithm. This process is demonstrated in Algorithm 4.1. The counter c represents the current number of shifts still required to meet all employees' contractual obligations and therefore ensuring that HC3 is satisfied. The statement "next highest priority demand" refers to a shift where the highest amount of demand could be satisfied. This is so that some level of quality can come from the Greedy Algorithm, and if the high-quality SC placements repeatedly fail, eventually employees can be allocated to shifts where there is no demand requirement as employees must work even if there is no demand (contractual requirements are HCs).

4.1.2 *Variable Neighbourhood Search*

The VNS requires a solution, comprised of neighbourhoods, as a parameter. A benefit of using the VNS is that, while the Greedy Algorithm limits the scope of local optima, the VNS allows some movement around the solution space,

Algorithm 4.1 Greedy Algorithm

```
1: infeasible ← true
2: while infeasible do
3:     k ← getRandomShiftAmongstHighestPriorityDemand
4:     e ← getRandomEmployeeWithHighestSkillPreference(k)
5:     if allocating e to k is feasible then
6:         Allocate e to k
7:     end if
8:     if all contractual obligations are met then
9:         infeasible ← false
10:    end if
11: end while
```

providing a higher chance of reaching the global optimum - or at least closer to it.

The solution the VNS receives is a set of weeks with employees allocated throughout. The VNS works in three steps as follows.

Firstly, by looking at a random day within the solution which has at least two employees on different shifts (day shift and late). Then an HC check is run, ensuring that swapping these two employees is acceptable. If they can safely be swapped, the algorithm then analyses the current solution in terms of fitness; how well the SCs are met. The swap occurs, and a second fitness test is run. If the fitness has worsened, the swap is reverted, otherwise the change is accepted. This intra-day swap mechanism can be viewed in Fig 4.2. Please note L is short for 'late', D is short for 'day' and 'e' is short for 'employee'.

The percentage chance of the next phase occurring is dependent on Metropolis-Hastings acceptance, which is described in section 4.1.2.1. In this phase, a more disruptive change occurs: instead of swapping employees on a single day, an employee is moved shifts from one day to another in the same week. This is

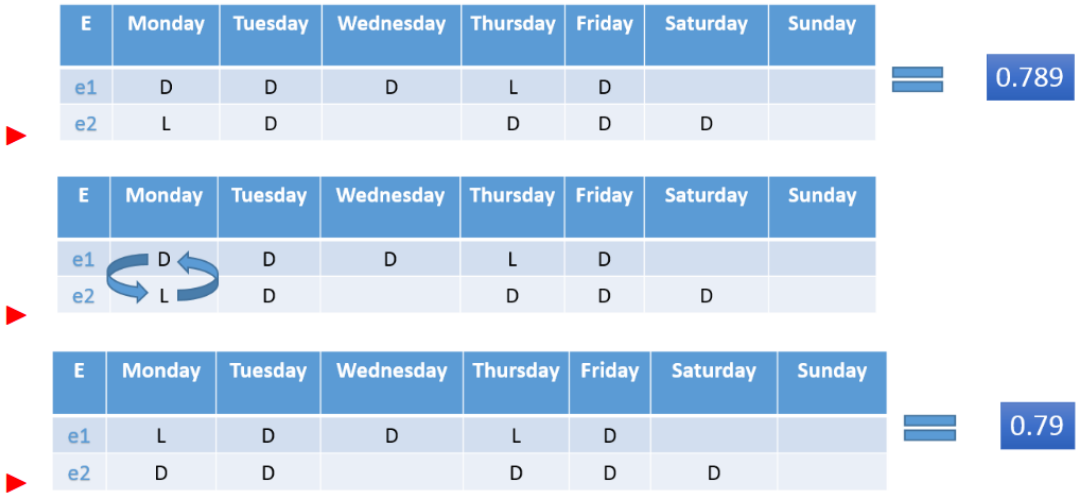


Figure 4.2: Intra-day Swap Mechanism

dependent on whether the HCs allow this swap (e.g. if an employee cannot contractually work Sundays, they will never be given a Sunday). Similarly to before, this change will occur, but only if fitness improves will the change remain as part of the solution. This inter-day swap mechanism can be viewed in Fig 4.3.

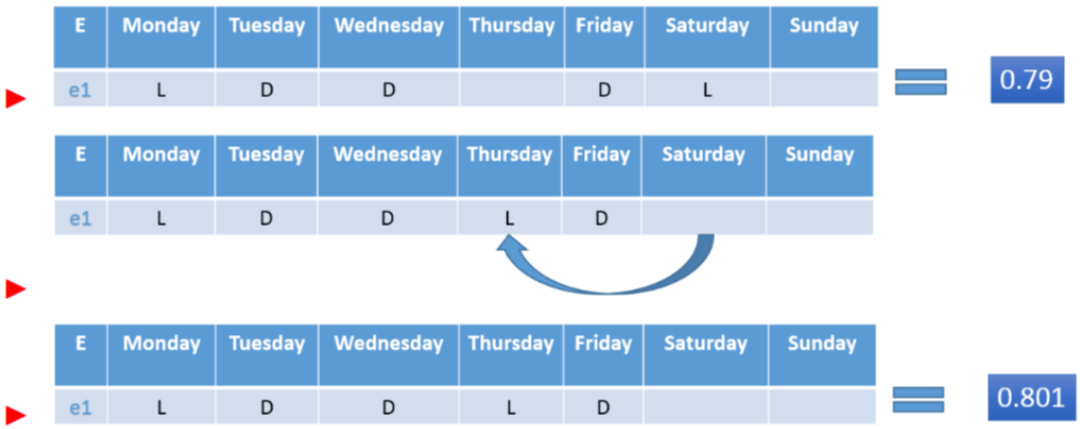


Figure 4.3: Inter-day Swap Mechanism

The final phase of the VNS is the least likely to occur as it is the most destructive, and is also controlled by Metropolis-Hastings acceptance, preventing this from occurring as frequently towards the end of run-time. In this phase a single employee has a week of allocations removed. The employee is then

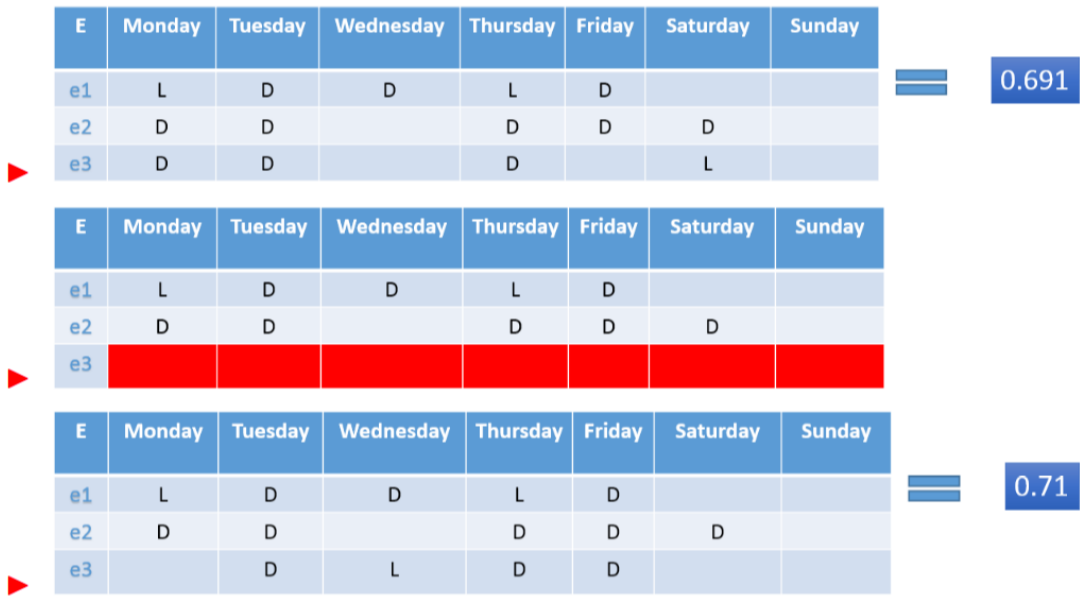


Figure 4.4: Delete & Regenerate Mechanism

given a new set of shifts which meet HCs, and to an extent SCs. Regardless of improvement of fitness or not, this change remains. This is intentional, in order to leave the current neighbourhood and potentially reach higher fitness levels. This is visualised in Fig 4.4.

4.1.2.1 *Metropolis-Hastings Acceptance*

In this instance the solution utilises the probabilistic acceptance criterion of Metropolis-Hastings acceptance, similar to exponential Monte-Carlo or Simulated Annealing, and embedding it within the VNS.

Unlike in more traditional methods, in this approach Metropolis-Hastings acceptance is used to attempt to find a global maximum. Specifically, this method of controlling the frequency of an event occurring over time is used to reduce the likelihood of disruption the closer the algorithm is to ending. The VNS should produce a better solution as time moves forward, but if the VNS is not controlled by Metropolis-Hastings Acceptance there is increased chance of losing a more highly rated fitness solution.

The goal of the Metropolis-Hastings Acceptance is to reduce the likelihood of unnecessarily altering a good solution later in run-time, and allow exploration of the solution space earlier in run-time. The rates mentioned henceforth are used during testing but are parameterised and therefore malleable as the user of the algorithm prefers (or as tuning tools or test design cases set). During the inter-day swap mechanism, the probability of this going ahead is first $p = 100\%$, but this decreases by $0.99 * p$ each iteration. The chance of a drastic change through the delete & regenerate method is at first 10%, and this reduces by 50% of the current likelihood every 10% of run-time (i.e. 10% at start, then 5%, then 2.5%, etc.) This means it's still possible a change will happen later in run-time but becomes much less likely, thus increasing likelihood of retaining a good solution.

4.2 RESULTS

Whether a solution can be accepted as feasible or infeasible is determined by the HCs being achieved, which ensure that legal and contractual requirements

Algorithm 4.2 Variable Neighbourhood Search

```
1: s ← greedyAlgorithmSolution()
2: r ← parameterisedRuntime()
3: d ← parameterisedDays()
4: d′ ← randomDay()
5: iDS ← 1.0 //Inter-day swap probability
6: dRG ← 1.0 //Delete & regenerate probability
7: while getCurrentTime() < r do
8:   if intra-day swap is feasible (d′) then
9:     e ← randomEmployee()
10:    f ← getFitness()
11:    swap(d′)
12:    f′ ← getFitness()
13:    if f > f′ then
14:      swap(d′) //undo
15:    end if
16:  end if
17:  if iDS > randomDouble() then//Inter-day swap
18:    e ← getRandomEmployee()
19:    f ← getFitness()
20:    d′ ← getRandomDayEmployeeIsWorking(e)
21:    d″ ← getDifferentRandomDayEmployeeIsWorking(d′)
22:    swap(d′, d″)
23:    f′ ← getFitness()
24:    if f > f′ then
25:      swap(d′, d″) //undo
26:    end if
27:  end if
```

```
    iDS *= 0.99
    if dRG > randomDouble() then //Delete & regenerate
        e ← getRandomEmployee()
        w ←getRandomWeek()
        removeAllEmployeeAllocationsForThisWeek(e, w)
        generateFreshAllocations(e, w)
    end if
    dRG *= 0.5
end while
```

are met. The quality of the feasible solutions is measured by how well the SCs are met. Solutions produced by this algorithm always meet HCs, and meet SCs to some extent. SCs are assumed to be of equal priority during testing, however it should be noted that these can be modified before run time so certain SCs can take a higher precedence and the final fitness results are weighted as such.

Fitness results are obtained from fitness functions, which each return a value pertaining to the SC they represent. The fitness results are then averaged (this can be parameterised to other weightings) to return an overall fitness value for the current solution. The results in this section were tested on an oracle enterprise Linux 6 OS server with a 12 core Xeon CPU for 1 hour. The tests used real employee data and real demand forecast data, however it should be stated that the number of shift types the employees work was increased as the data provided was for an area where engineers worked a very small number of late shifts per quarter.

There were 50 tests conducted, and the results are shown in Fig 4.5. These results show the mean fitness of the solution after being processed by the Greedy Algorithm compared to running variable neighbourhood search processes for one hour. These values are calculated by taking the mean of all fitness values carried out from the SCs. In practice the SCs can be weighted so

the engine can be used to produce solutions which cater to specific intrigues, for example increasing the weighting of SC1 being met.

The mean of all Greedy Algorithm fitness results is 0.681 and the mean of all VNS fitness results is 0.784, which indicates that over fifty tests running at 1 hour each, there is an average of 10.3% increase in fitness using the VNS algorithm after the Greedy Algorithm sets up an initial solution. Metropolis-Hastings acceptance is the technique used in order to prevent a closer to optimal solution from being disrupted later in the run-time.

Linear regression analysis was conducted to analyse the relationship between the Greedy Algorithm solution and the resulting VNS solution. The coefficient of determination R^2 is shown in Fig 4.5. This statistic gives useful information about the goodness of fit. An R^2 value of 1 would indicate the regression line perfectly fits the model. In this instance the R^2 value is calculated at 0.83, this is a good indicator of positive correlation between the results of the VNS algorithm and the Greedy Algorithm.

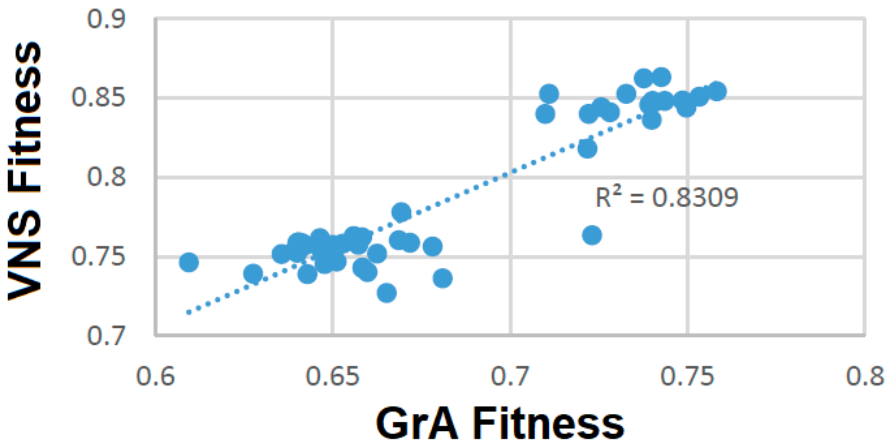


Figure 4.5: Mean VNS Fitness Scatter Plot

Further analysis was conducted on the normality of data. First, Shapiro-Wilk tests were performed and found the normality of the data was rejected (values can be found in Table 4.1).

	W	p
Greedy Algorithm	0.726	2.484e-08
VNS	0.754	9.002e-08

Table 4.1: Shapiro-Wilk Tests

This prompted the need to visualise the data to find a reasoning for the lack of normality. Two Q-Q plots can be seen — the Greedy Algorithm Q-Q plot in Fig 4.6 and the VNS Q-Q plot in Fig 4.7. These visualisations show that the non-normal distribution of data is due to outliers.

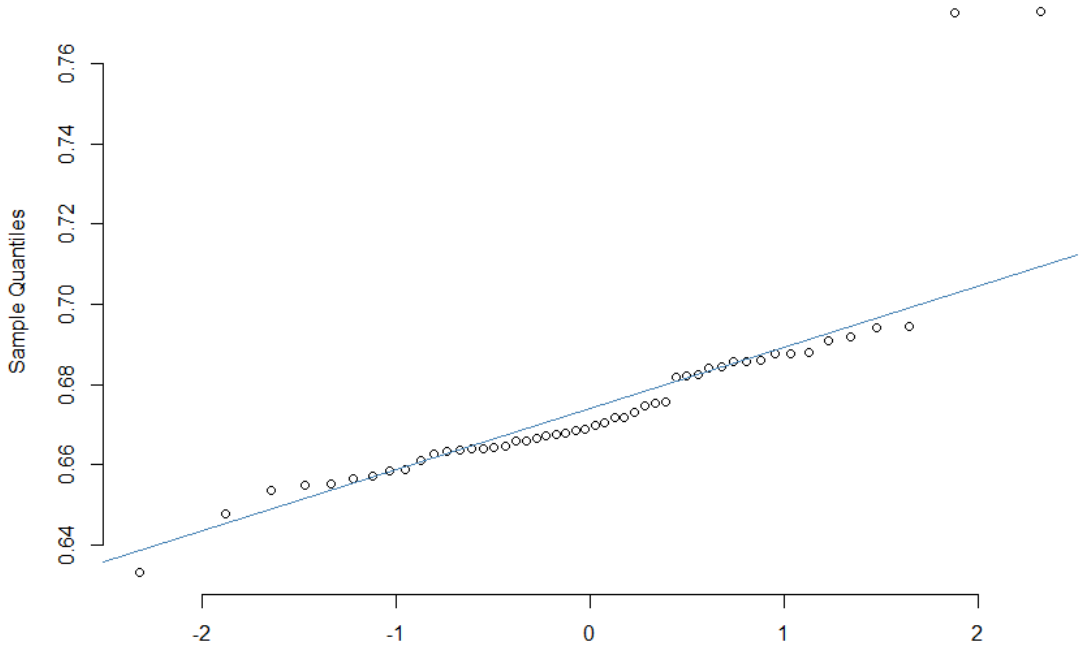


Figure 4.6: Greedy Algorithm Q-Q Plot

After removing outliers, a normal distribution of data emerges. For the Greedy Algorithm tests this can be visually confirmed in Fig 4.8, and for VNS in Fig 4.9.

With the outliers removed, it is also wise to visualise the spread of values for both datasets. This can be viewed in Fig 4.10. Additionally, the Shapiro-Wilk

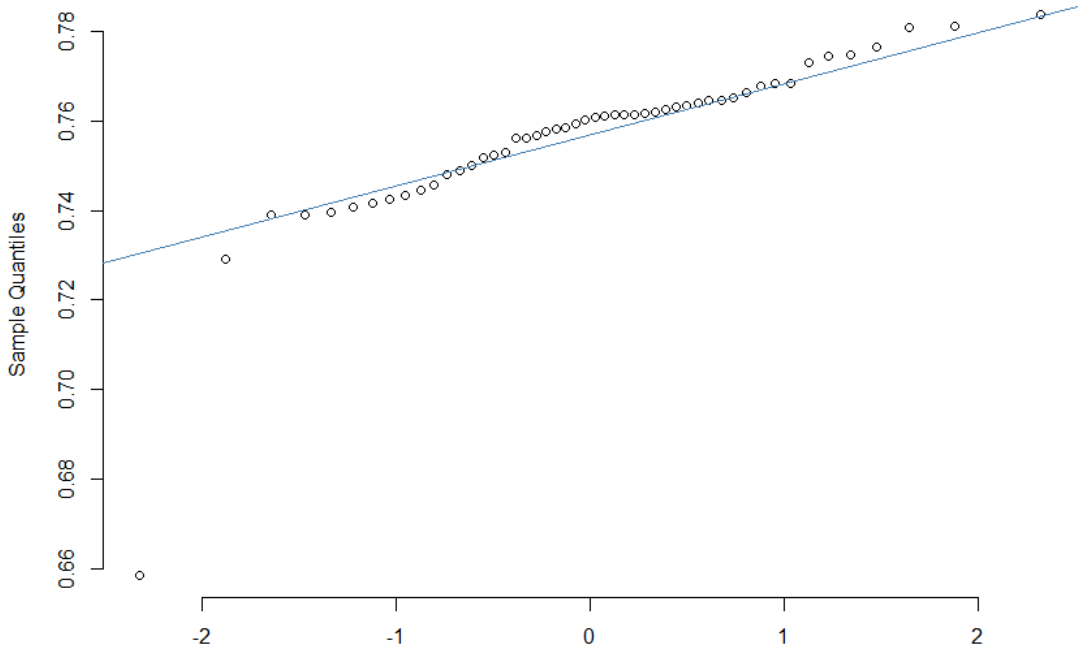


Figure 4.7: VNS Q-Q Plot

	W	p
Greedy Algorithm	0.96546	0.1675
VNS	0.97789	0.4939

Table 4.2: Shapiro-Wilk Tests (Outliers Removed)

tests were run again to confirm normality after removing outliers. The results can be viewed in Table 4.2. With the p value in both instances being above an α level of 0.05, the Shapiro-Wilk tests now confirm normality in the data. The null hypothesis — that the data are normally distributed — cannot be rejected.

While testing in this section allows the algorithm to run for 1-hour, in testing with a 5 second upper limit (as a requirement of this algorithm by the FSO), feasible solutions with fitness values much lesser than the 1 hour run-time are produced (on the same employee region and demand set). However, these still provided improvements upon the baseline, on average 3.7% improvement

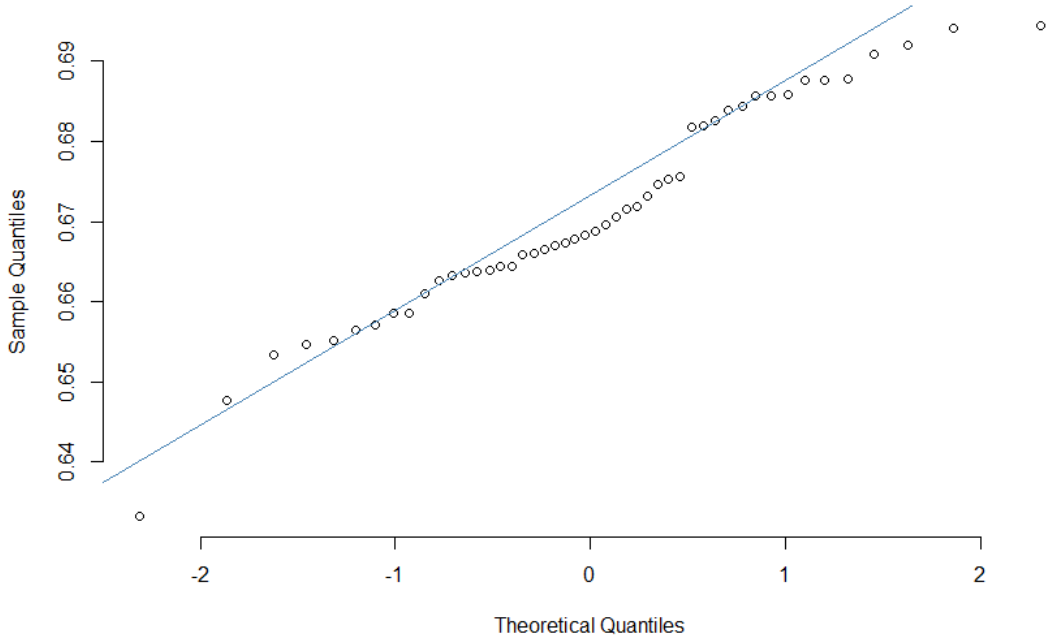


Figure 4.8: Greedy Algorithm Q-Q Plot with Outliers Removed

upon the greedy algorithm result, showing that the additional run-time given to VNS is potentially worthwhile.

4.3 DISCUSSION

This chapter details a case study of using a variety of metaheuristics to solve a highly constrained real-world shift scheduling and employee rostering problem. This was achieved using VNS, controlled with a probabilistic acceptance criterion of Metropolis-Hastings acceptance, upon an initial solution generated by a Greedy Algorithm.

This chapter has taken a real-world problem and used a combination of algorithms to improve upon the current standard at the FSO (of manual rostering) by producing a set of feasible schedules of good quality within 5 seconds, as required by the FSO.

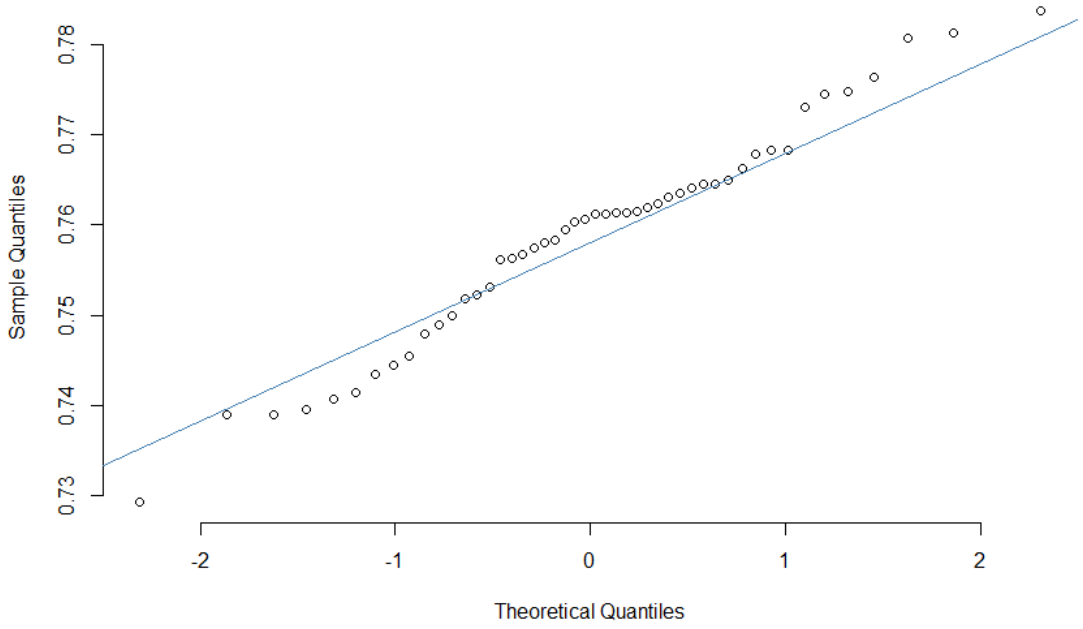


Figure 4.9: VNS Q-Q Plot with Outliers Removed

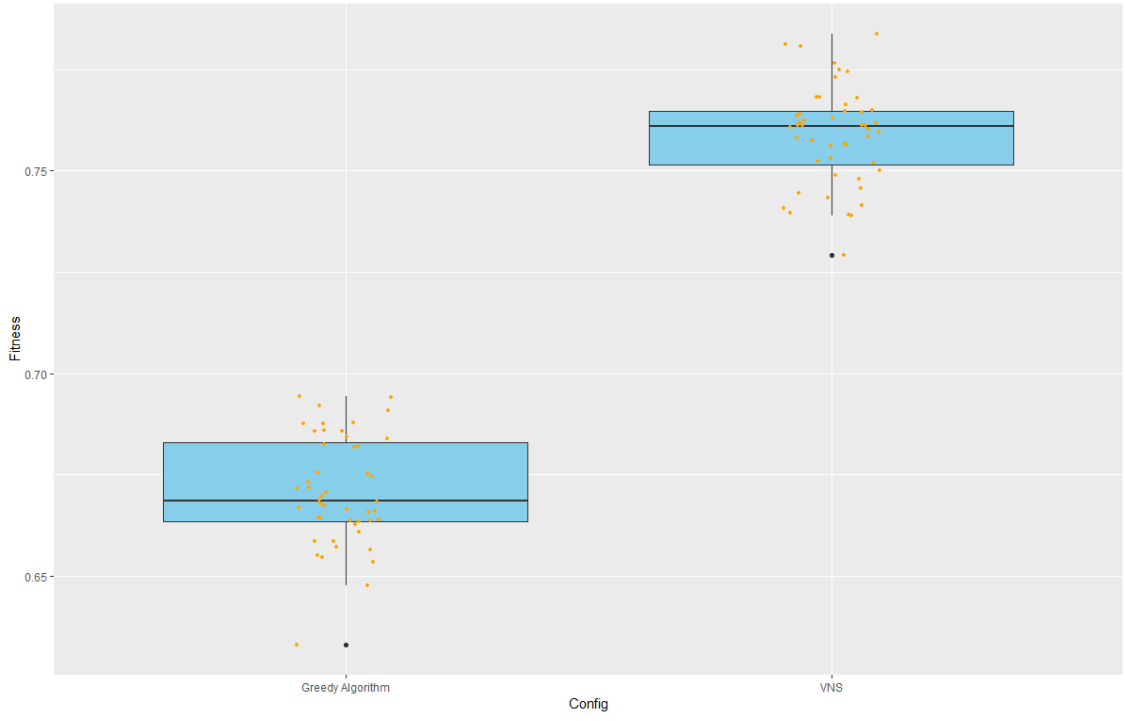


Figure 4.10: Greedy Algorithm vs VNS Boxplot

Potential future work could be to extend the Greedy Algorithm with extra functionality which will provide a starting set of shift patterns which already meet some HCs. This head-start should improve the time it takes for the algorithm to run, as well as potentially improving fitness results since closer to optimal patterns can be preset with this method. Alternatively, entirely replacing the Greedy Algorithm with a more powerful heuristic could provide a better starting point for the VNS. There is also the option of furthering analysis on which days least meet fitness criteria, and attempting to focus on those days in particular — this may improve fitness more quickly by solving problem areas. Other acceptance criterion algorithms are available which could be explored to provide further exploration and exploitation within the limited 5 second time frame.

Gratitude is extended to the FSO for providing this data and providing much information during the constraints capture phase of this work.

5

CHAPTER 5 - PERSONALISED SCHEDULING & ROSTERING - EVOLUTIONARY RUIN & STOCHASTIC RECREATE

This chapter solves the refined personalised scheduling & rostering problem as described in Chapter 2.

5.1 IMPLEMENTATION

This section describes implementation — in particular, the changes to the algorithm presented in Chapter 4 required to meet the specific requirements of the problem. The order in which the algorithm is implemented to operate has been changed from that of the theoretical framework [16] to meet the requirements of the FSO problem. As suggested by Li et al [16], a different control mechanism (EMCAC) is used to allow a halting operation; to prevent overt disruption in the final iterations of the search; allow exploration of the search space earlier in run-time; and if parameterised correctly to prevent early convergence. A pseudocode overview of the modified algorithm is given in algorithm 5.1, and a flow diagram can be viewed in Fig 5.1.

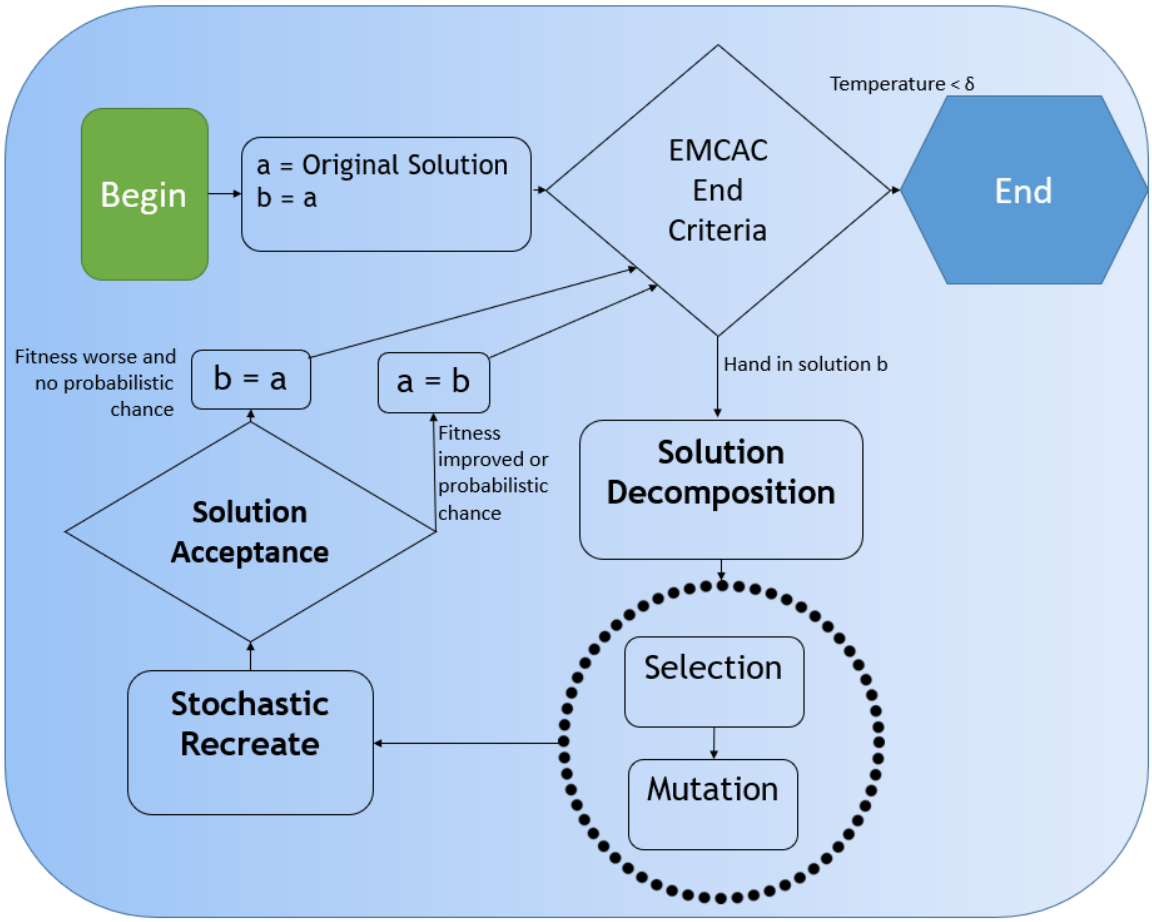


Figure 5.1: Flow Diagram

Algorithm 5.1 Evolutionary Ruin & Stochastic Recreate

```
1: t ← getParameterisedTemperature()
2: δ ← getParameterisedDelta()
3: cr ← getParameterisedCoolingRate()
4: sr ← getParameterisedSelectionRate()
5: mr ← getParameterisedMutationRate()
6: a ← greedyAlgorithmSolution() //Initialisation complete.
7: while t > δ do //Exponential Monte-Carlo Acceptance
8:     b = a.copy()
9:     f ← decomposition(W) //W are parameterised weightings. Solution de-
   composition complete.
10:    a = selection(a, sr)
11:    a = mutation(a, mr) //Evolutionary ruin complete.
12:    a = rebuild(a) //Stochastic recreate complete.
13:    p ← getAcceptanceProbability()
14:    if p < r then //Random int
15:        a = b.copy()
16:    end if//Solution acceptance complete.
17:    t *= cr
18: end while
```

5.1.0.1 *Phase 0 - Initialisation*

The first phase initialises the problem, including creating every potential shift. This means after this phase has run, there is a shift of every potential length, with every potential start and end time, for every particular skill requirement, per shift rule. It is important to note that during initialisation, shifts which are impossible are not created (for example no employees work Sundays, so no Sunday shifts are created). This is important as it allows

the algorithm to explore the search space without having to recreate shift objects repeatedly, saving on potential processing costs. After initialising, this phase also creates an initial solution by Greedy Algorithm. The Greedy Algorithm in this section is simplistic, and calls on the fitness functions of the Solution Decomposition phase to calculate preferred employee allocations. See algorithm 5.2 for pseudocode. Having a greedily constructed solution means that the second phase of evolutionary ruin has allocations to ruin during the first cycle. After the initial solution creation, the algorithm cycles through subsequent phases until the end criterion is met — specifically, until the temperature is equal to or less than the delta.

Algorithm 5.2 Greedy Algorithm

```
1: infeasible ← true
2: while infeasible do
3:     k ← getRandomEmployeeShiftAllocationAmongstHighestFitness()
4:     if allocating e to k is feasible then
5:         Allocate e to k
6:     end if
7:     if all contractual obligations are met then
8:         infeasible ← false
9:     end if
10: end while
```

5.1.0.2 *Phase 1 - Solution Decomposition*

The solution decomposition consists of Employee Shift Allocation (eSA) objects as the componential elements for analysis. These objects consist of a shift and an employee. Each of the eSA objects across the scheduling and rostering period are assessed for their fitness, which is derived from each objects ability to satisfy the constraints. The calculation by which the overall object fitness is

derived is dependent on the parameterised weightings per constraint. The SC weightings are parameterised so that users of the algorithm can modify constraints to be worth more or less than other constraints, for use in real-world scenarios (for trying different preferences, techniques etc.). This feature was implemented for a more realistic business use case. By default the HC fitness is worth 95% of the total and the final 5% is between the weighted SCs. Due to the nature of this problem there is a separation of holistic constraints and lower level constraints, which are described in the ER&SR algorithm. This is due to the holistic constraints requiring knowledge of the whole solution, including all shifts and employee allocations. The holistic constraints mentioned are HC2, HC3, HC7 & SC4. In this implementation there is a deviation from the algorithm presented in Chapter 4 to allow consideration of these holistic constraints. This is achieved by measuring these constraints at a schedule-wide perspective, then modifying the fitness values of individual eSA objects. If an object is found to break these constraints (for example, scheduling an employee to work a 6-day week when contracted for 5), then fitness of all offending eSA objects is probabilistically modified. The schedule thus far is unchanged from the initial solution construction, or since the previous iteration as this solution decomposition phase only provides analysis of solution fitness.

5.1.0.3 *Phase 2 - Evolutionary Ruin*

The next phase is the evolutionary ruin section where every eSA object is evaluated and employees are probabilistically removed from shifts. This phase was named "Evolutionary Ruin" by Li et al [15] due to the introduction of the selection and mutation operators. With all the fitness values obtained from the Solution Decomposition phase, each eSA object is evaluated first in the selection mini-phase. The selection mini-phase first calculates an acceptance rate:

$$P = r * x_1 \tag{5.1}$$

Where P is the probability of acceptance, r is a random number between 0 and 1, and x_1 is a parameter to tune the fastidiousness of the selection mini-phase. Then P is compared to the fitness of the object: if the fitness is higher than the random number, continue to the next eSA. If instead the fitness is lower than P, then the employee is removed from this eSA.
After the selection mini-phase is the mutation mini-phase, which will probabilistically remove an employee from every eSA object without regard to fitness. This introduces an additional level of stochasticity by probabilistically deallocating employees from eSA objects, allowing additional exploration of the solution space. This phase solely targets the components which previously were selected for retention in the selection phase (or rather, not selected for deallocation). This proceeds without regard for fitness values. The ruin phase causes the occasional removal of an employee from an individual shift. Careful parameterisation of this phase is important as excessive ruination can hinder exploitation. The chance of the ruination occurring is calculated as follows:

$$P = r * x_2 \tag{5.2}$$

Where P is the probability, r is a random number between 0 and 1, x_2 is the parameter used to increase or decrease probability, and the subscript is the parameter to control the rate of removal.

5.1.0.4 *Phase 3 - Stochastic Recreate with Solution Acceptance*

This now partially complete solution (which may even be empty if enough components are destroyed) is reconstructed during the stochastic recreate phase. This phase executes repeatedly until a full solution is available. First, the list of employees is randomly ordered to prevent giving preference to the first employees in the list repeatedly. An employee is taken from the list and

the number of shifts worked per week is calculated. Every week where the employee is not working enough shifts is allowed to rebuild. This is conducted by selecting a random day in the week the employee is not currently working then selecting a random shift that exists on this day, and the employee is placed onto this shift. The fitness of this component is then analysed, and a random floating-point number between 0 and 5 is generated. Due to the HCs consisting of 95% of the fitness, the generated number becomes the allowed SC fitness. If the allocation is above the acceptable limit ($95\% + r$) then the allocation is accepted. The recreation method repeats for all employees and all weeks until a full schedule has been generated.

The stochastic recreate with solution acceptance phase is controlled by EMCAC acceptance criterion, a frequently used methodology derived from statistical thermodynamics. In this circumstance however it is used to control the ruining capabilities of ER&SR as well as this phase. The acceptance probability is as follows [6]:

$$P = exp((o - n)/t) > r \tag{5.3}$$

Where P is the probability of accepting a worsened state, o is the original solution energy cost, n is the energy cost, t is the temperature and r is a random number between 0 and 1. This allows extensive exploration in the early stages of run-time but turns to more exploitative measures later in run-time. If the solution is accepted then the previous solution is overwritten, and the algorithm enters the next iteration or ends if the temperature $t < \delta$ (where δ is the parameterised end criterion). If the solution is rejected, the previous solution is preferred, and the algorithm enters the next iterations or ends if the temperature $t < \delta$. Due to the nature of EMC acceptance criterion, the best solution found is not necessarily the final solution at the end of run-time. As such, this implementation saves the best solution found for actual use.

The results in this section are derived from tests which ran on an oracle enterprise Linux 6 OS server with a 2.3GHz Xeon CPU. Parameters were first tuned using the Taguchi method, some of the results of these tests are also presented in this section. The parameters which provided the best overall fitness (as an unweighted average) are then discussed.

The effectiveness of metaheuristics is largely dependent on well-tuned parameters [4]. There are many experimental designs which can be used for tuning the parameters, such as grid search, random search, or more popular recently; hyper-heuristics. However, the Taguchi method has not been used in personnel scheduling, but has much success in the experimental design of engineering methods [78], electrical engineering [79], designing test cases for a open routing vehicle problem using neural networks [80], and others.

The Taguchi design of experiments evolved from orthogonal arrays and Latin squares [81]. This design of tests allow experimentation with a variety of parameters to find a combination of parameters to produce a high fitness value with ER&SR, and reduce the testing time (which is deemed important to the FSO).

The cooling rate is used as the EMCAC control mechanism which controls the number of iterations. t is set at 10000 for all tests, and the testing included 6 potential cool rates, from 0.09 - 0.59.

An L16 (4^2) orthogonal array was designed consisting of 2 factors with 4 possible values and 16 runs - however, as having 0 selection and 0 mutation rate is redundant, this was removed (so only 15 runs). This consisted of the following parameters: the Selection rate (SR) is used during the evolutionary ruin phase as is the mutation rate (MR).

Table 5.1: Top Five Mean Taguchi Test Results

SC1	**SC2**	**SC3**	**SC4**	**SR**	**MR1**	**MR2**	**Mean Fitness**	σ
0	**1**	**0**	**1**	**0.33**	**1**	**0**	**96.983**	**0.038**
1	0	1	0	1	0.66	0	96.975	0.034
0	0	1	0	1	1	0	96.973	0.087
0	0	1	1	0.33	1	0	96.972	0.055
0	1	1	0	0.33	0.33	0	96.971	0.054

This is combined with an L8 (2^4) orthogonal array consisting of 4 factors with 2 possible values and 8 runs. These parameters are simply toggles to turn SCs (SC1 - SC4) on or off. Two tables were used due to the limitations in the number of Taguchi designs that exist. It is likely other problems would require different parameters from those given in Section 5.1, as these parameters are tuned for this problem specification and dataset. The top five fitness results from Taguchi testing are shown in table 5.1. Cooling rate is not shown because all of the 5 results shown had a cooling rate of 0.09.

Testing included running each above Taguchi test 10 times on a demand set of 1 month for close to 200 employees. Fig. 5.2 highlights a sample of the testing conducted (6 configurations out of 600 total configurations), showing that there is a significant difference in the distribution of fitness levels obtained using different parameter configurations. ANOVA testing also confirmed this. However there was little difference between the top 5 results as shown in Fig. 5.3.

The best-case scenario can be seen in more detail in Table 5.1. The parameters for the highest fitness are noteworthy, as it seems to be generally preferable to have a maximum amount of selection rate and about 66% of mutation rate. It also highlights that it is easier to forge a solution where SC2 and SC3 are met

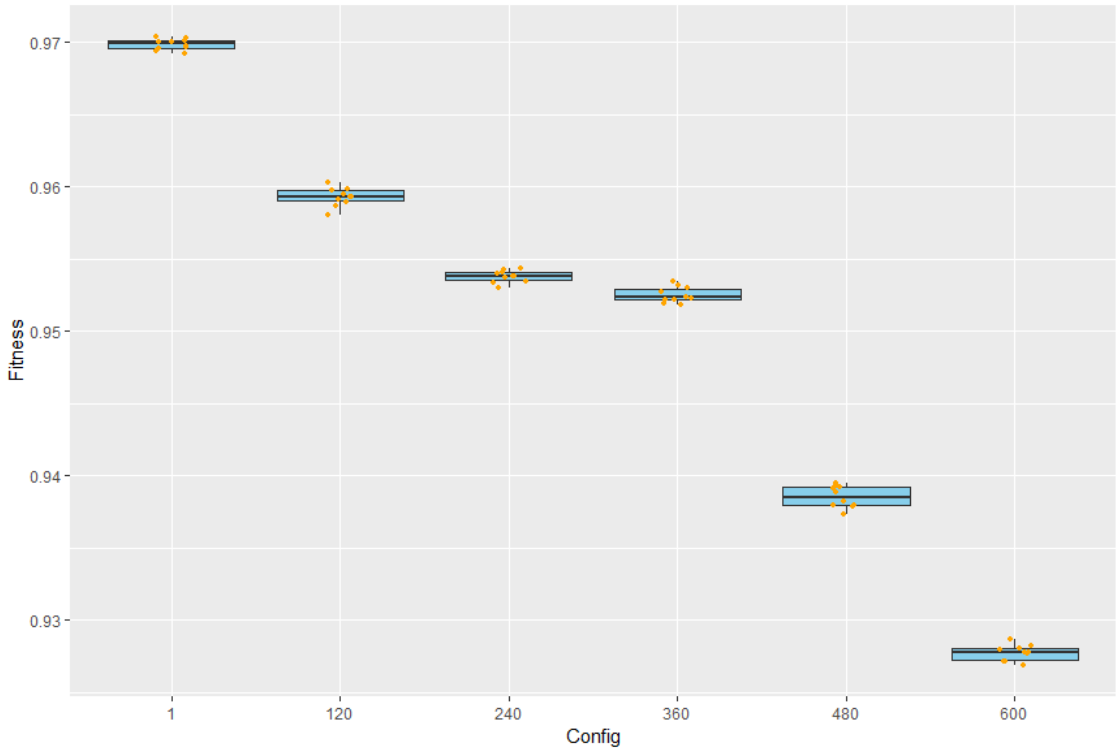


Figure 5.2: Box-Plot of Fitness for 6 Sample Parameter Configurations

rather than SC1 and SC4. Our recommendation to the FSO in this instance is to test parameterising the SCs in a weighted format to find a balance that most satisfies the company and employee requirements. An example of the fitness throughout a single test with a single set of parameters can be viewed in Fig. 5.4.

This solution improves upon the current implementation of manually generated rosters. The improvements are found across all SCs when the best case solutions of both the manual and the ER&SR implementations are compared.

5.3 CONCLUSIONS

In this chapter a revised implementation of the ER&SR algorithm is presented to tackle a real-world employee scheduling and employee rostering problem. This approach is novel as it has not previously been used to tackle real-world employee scheduling and rostering problems. The results have shown that this

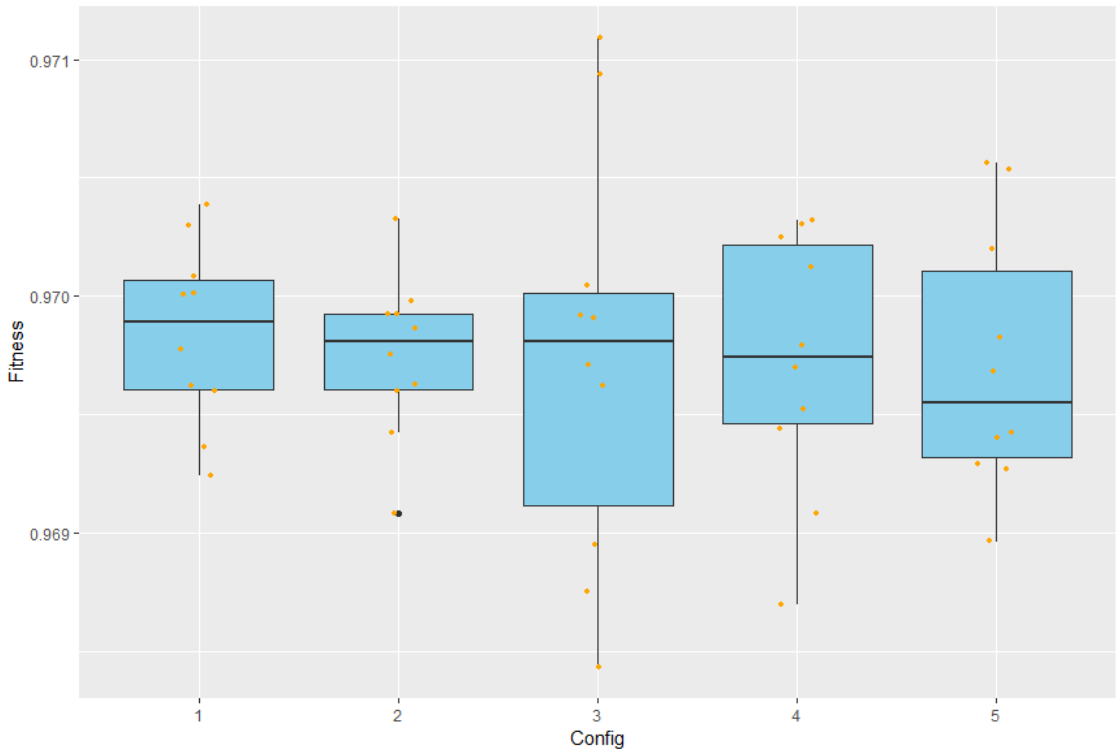


Figure 5.3: Box-Plot of Fitness for the Best 5 Parameter Configurations

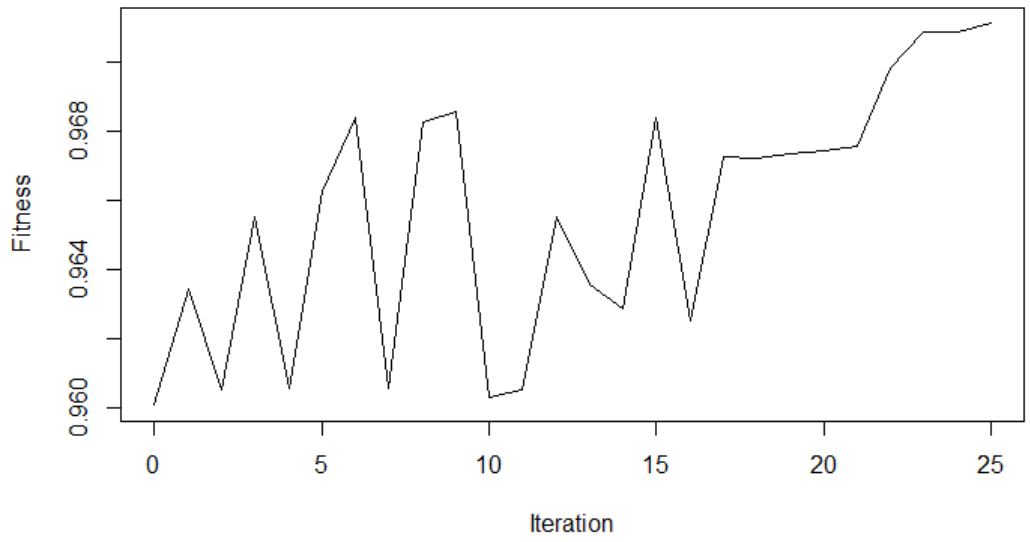


Figure 5.4: Fitness per Iteration of a Single Test.

approach is an improvement upon the current implementation for generating employee rosters and schedules, and that ER&SR is capable of generating near-optimal solutions for this type of problem.

However, there is scope for further research. In particular using modified ER&SR algorithms to personnel scheduling and employee rostering problems when compared to industry solvers such as CPLEX and state of the art solutions. There is also scope for further research into alternative test case designs and parameter tuners such as `irace` [62].

6

CHAPTER 6 - ROSTER PATTERNS - EVOLUTIONARY RUIN & STOCHASTIC RECREATE HYBRIDISED WITH VARIABLE NEIGHBOURHOOD SEARCH

In this chapter the sub-problem known as the RP scheduling problem is tackled. The problem is different to the personalised scheduling problem tackled in the previous two chapters. Both problem types are described in Chapter 2. Given the additional constraints this problem places on the search, an extension to the ER&SR approach is proposed and introduced using VNS. This chapter begins by introducing the hybrid algorithm, before giving experimental results with a real-world case study.

6.1 IMPLEMENTATION

6.1.1 *Evolutionary Ruin & Stochastic Recreate*

This chapter proposes a hybrid extension of the algorithm described in Chapter 5. ER&SR was used to solve not only an employee rostering but also a shift scheduling problem, without VNS in 4, however in this instance ER&SR is hybridised with VNS to solve a real-world employee rostering problem, not a shift scheduling problem.

The implementation was designed to be modular, with a VNS component which can be called at any stage of the algorithm. This allows consideration of a variety of approaches and how to best exploit the local search advantages

from VNS in conjunction with the more exploratory ER&SR. As such the following hybridised algorithms have been implemented:

(i) ER&SR with VNS running sequentially after ER&SR stopping criterion (referred to as 'ER&SR with VNS Sequential')

(ii) ER&SR with VNS running after accepting a modified schedule (referred to as 'ER&SR with VNS Internal')

(iii) ER&SR with both the above (referred to as 'ER&SR with both VNS internal and sequential')

Pseudocode of ER&SR with VNS can be seen in Listing 6.1. Lines 19 and 25 reference VNS, of which is described more fully in subsection 6.1.2, with VNS pseudocode available in Listing 6.2.

The following revisions to the algorithm have been made to tailor it to this specific sub-problem. This enumeration refers to line numbers within the aforementioned pseudocode in listing 6.1 in the format of [Listing Name - Line Number].

1. The initial set up phase initialises the software and reads the current hand-written solution in place by the FSO [6.1 - 1-9].

2. The EMCAC control mechanism is initialised with the parameterised inputs, controlling the number of times ER&SR loops, as are the VNS iterations parameterised.

3. Phase 1 - solution decomposition no longer considers HCs. This is due to RPs which ensure contractual and legal requirements cannot be breached. More precisely, the HCs considered in this chapter can be temporarily breached during the evolutionary ruin phase, but are all then restored during stochastic recreate. SCs are considered in the solution decomposition [6.1 - 12], and the fitness evaluations can be

Algorithm 6.1 Evolutionary Ruin & Stochastic Recreate

```
1: t ← getParameterisedTemperature()
2: δ ← getParameterisedDelta()
3: cr ← getParameterisedCoolingRate()
4: sr ← getParameterisedSelectionRate()
5: mr ← getParameterisedMutationRate()
6: vnsInt ← getVnsInternalIterations()
7: vnsSeq ← getVnsSequentialIterations()
8: a = originalSolutionFromFSO()
9: t′ = t
10: while t′ > δ do //Exponential Monte-Carlo Acceptance
11:    b = a.copy()
12:    f ← decomposition(W) //W are parameterised weightings. Solution de-
       composition complete.
13:    a ← selection(a, sr)
14:    a ← mutation(a, mr) //Evolutionary ruin complete.
15:    a ← rebuild(a) //Stochastic recreate complete.
16:    p ← getAcceptanceProbability()
17:    if p < r then //Random int
18:       a = b.copy()
19:       if vnsInt > 0 then
20:          beginVNS(vnsInt)
21:       end if
22:    end if//Solution Acceptance complete.
23:    t′ *= cr
24: end while
25: if vnsSeq > 0 then
26:    beginVNS(vnsSeq)
27: end if
```

completed by considering only the potential changes from the current state (referred to by [16] as 'componential' fitness). This is necessary as all four SCs consider the percentage of all individual shifts meeting criteria. The solution decomposition serves the purpose of analysing fitness during ER&SR run-time as well as after VNS swaps have occurred to decide whether to keep or revert changes.

4. Phase 2 - evolutionary ruin is mostly unchanged from the algorithm presented in Chapter 5. Instead of ruining individual shifts, both the selection and mutation elements now ruin employee RPs, by giving a null allocation. Selection [6.1 - 13] will destroy elements with the following probability:

$$P(s) = r * x_1 \tag{6.1}$$

where $P(s)$ is the probability of acceptance, r is a random number between 0 and 1, and x_1 is a parameter to limit the stochasticity of selection. If P is higher than the fitness of the employee allocated to their current RP, then the employee has their RP removed for a new one. After selection is complete, mutation [6.1 - 14] considers all non-affected employees, and randomly removes employees from RP. This adds an additional level of stochasticity to the search. Fitness is not considered during mutation. Mutation can be overly destructive and hinder exploitation, so careful parameterisation is important. The chance of ruination is calculated as follows:

$$P(b) = r * x_2 \tag{6.2}$$

where $P(b)$ is the probability, r is a random number between 0 and 1, x_2 is the parameter used to increase or decrease probability. This phase is likely to break a number of HCs before the schedule is reconstructed and made feasible in the next phase.

5. Phase 3 - stochastic recreate [6.1 - 15] no longer finds random shifts to allocate employees to, but instead allocates employees to random RPs and starting weeks. Employees to be allocated a new shift pattern are chosen randomly. This is so the search space can be explored if HC2 or HC3 are enabled. These HCs are considered with every RP allocation to ensure the maximum bound is not breached. If HC4 is toggled on, only RPs the employee has selected may be considered for allocation. Otherwise, any RP can be allocated.

6. Phase 4 - solution acceptance [6.1 - 16-22] will probabilistically accept the changes. This is unchanged from the above-mentioned previous work, and still uses the EMCAC. The acceptance probability is calculated as follows:

$$P = \exp((o - n)/t) > r \tag{6.3}$$

where P is the probability of accepting a worsened state, o is the original solution energy cost (modelled as inverse fitness), n is the energy cost of the new solution, t is current temperature and r is a uniformly generated random number between 0 and 1.

6.1.2 *Variable Neighbourhood Search*

In the same manner as the last subsection, code lines are referred to in the following manner: [6.2 - Line Number].

After the ER&SR algorithm has accepted a new solution, VNS is called (if parameters indicate to do so). The specific VNS method chosen is basic VNS [3]. While more complex varieties of VNS are available (such as other hybrid solutions [53], or with adaptive memory improvements [82]), ER&SR already provides a feasible close-to-optimal solution. Therefore, VNS is used to search

nearby neighbourhoods for minor improvements rather than building from the baseline.

Assuming iterations left is above zero, VNS first finds the fitness of the current schedule [6.2 - 7]. Each employee defines a neighbourhood. A random employee is selected, and so is their RP [6.2 - 8-9]. If HC4 is enabled, only RPs the employee has a preference for are considered for the next step, otherwise all RPs are considered [6.2 - 12]. A random RP from the top ranked RPs in the selected list is chosen [6.2 - 12 or 13] (for example, if there are 300 potential RP changes, and 7 have equal fitness potential, one of them is chosen randomly). Checking if no HCs are failed by a potential swap, then the swap goes ahead [6.2 - 15-21]. Then fitness of this new schedule is determined [6.2 - 22]. If the new fitness is lower than the original, then the schedule is reverted, otherwise it is kept [6.2 - 23 - 25].

6.1.3 *Order of Events*

After the internal VNS has completed, ER&SR repeats until the completion criterion is met. VNS can then run after ER&SR has completed, if parameterised. This would allow the best solution found by ER&SR to be exploited further by VNS. In this implementation VNS is run either a) during ER&SR (shortly after a solution has been accepted) for a number of parameterised iterations, b) after the ER&SR iterations have all completed, or c) both. This means this algorithm can be run with ER&SR only, ER&SR with VNS Internal, ER&SR with VNS Sequential, and ER&SR with both. These are compared as separate configurations, alongside others, in section 6.2.

Fig 6.1 provides a visual overview of the hybrid algorithm. In the diagram, the internal VNS (to the left) is shorthand for the same process to the right of the diagram. Both the internal VNS and the sequential VNS are only active if parameterised to run.

Algorithm 6.2 Variable Neighbourhood Search

```
1: k_max ← getKmax()
2: t_max ← getTmax()
3: W ← getWeightings()
4: while getCurrentTime() < t_max do
5:   k ← 0
6:   while k < k_max do
7:     f ← getFitness(W)
8:     e ← getRandomEmployee()
9:     s ← getRP(e) //RP is Roster Pattern.
10:    s′ ← null
11:    if isHC4toggled() then
12:      s′ ← randomRPfromBestRPsFromPreferences(e)//Only consider
       RPs that e can feasibly be allocated to.
13:    else s′ ← randomRPfromBestOfAllRPs() //Consider all RPs.
14:    end if
15:    if swapWouldBreakHC2() then
16:      continue
17:    end if
18:    if swapWouldBreakHC3() then
19:      continue
20:    end if
21:    giveEmployeeRP(e, s')
22:    f′ ← getFitness(scWeightings)
23:    if f′ < f then
24:      giveEmployeeRP(e, s)
25:    end if
26:    k = k + 1
27:  end while
28: end while
```

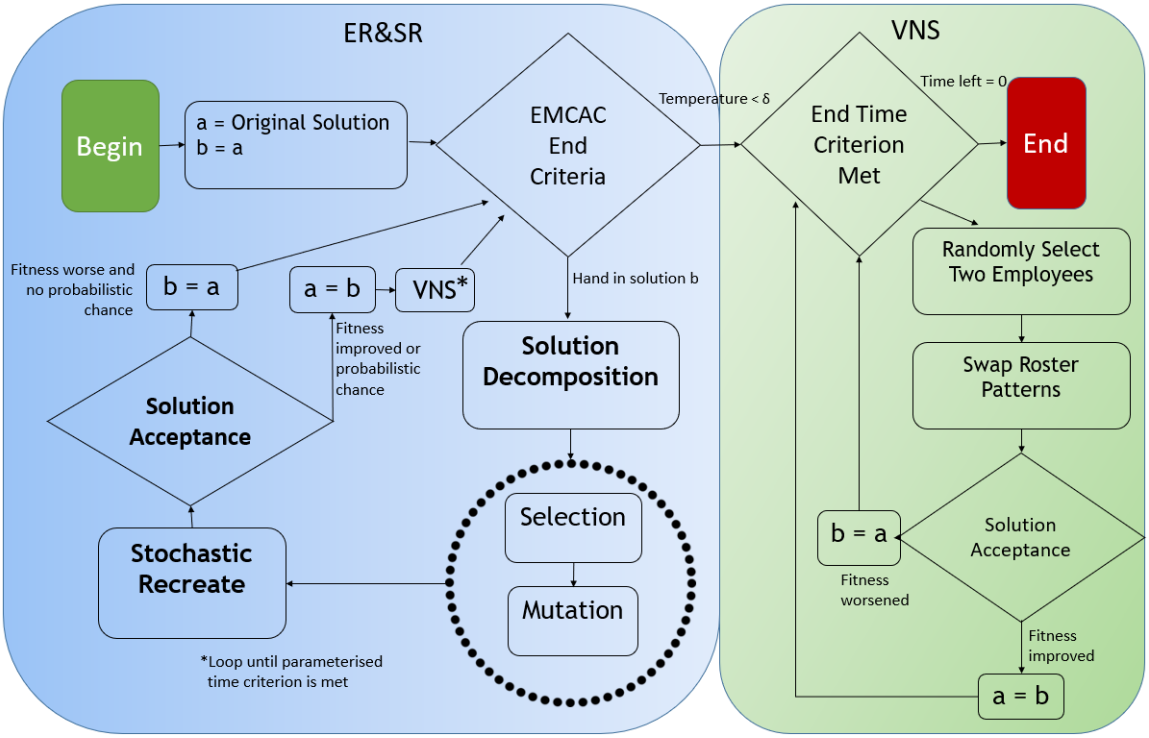


Figure 6.1: Diagrammatic Overview of ER&SR & VNS Hybrid

6.2 RESULTS

In this section the results are discussed from tests run on an Intel i5 2.49GHz CPU. The configurations tested were:

(A) ER&SR with both VNS Internal and VNS Sequential.

(B) ER&SR with VNS Internal.

(C) ER&SR with VNS Sequential.

(D) ER&SR only.

(E) VNS only.

(F) SA only.

(G) Baseline.

Parameter	Reference	Type	Range	Additional Parameters
temp	–a	r	(1,1000000)	
cr	–b	r	(0.0001,0.5)	
sr	–c	r	(0.0001,1.0)	
mp1r	–d	r	(0.0001,1.0)	
mp2r	–e	r	(0.0001,1.0)	
vnsint	–f	c	(“on")	
vnsintr	–g	r	(1.0,100.0)	| vnsint %in% c(“on")
vnsseq	–h	c	(“off")	
vnsseqr	–i	r	(1.0,2000.0)	| vnsseq %in% c(“on")

Table 6.1: Parameters to be Tuned by irace

The baseline is also shown in the results for comparison (the baseline is the fitness computed from the manually generated solution in place at the FSO). The baseline solution is always feasible.

The effectiveness of metaheuristics are largely dependent on well-tuned parameters [4]. This is particularly important in this instance as there are many parameters and ranges to select from. Thus, the parameters for testing were chosen by the `irace` parameter configuration tool [62], such automated parameter tuning being preferable to onerous manual configuration [83]. During `irace` testing, the testing of elite configurations was enabled, meaning that only the best configurations found during the run of `irace`, the final best, will be used in the testing phase. The first elimination test value was set to the default of five. The default statistical test configuration for `irace` was used (F-test).

Config[1]	Temperature	CR	SR	MP1	MP2	VNSi	VNSs	SA
A	**116621.6**	**0.045**	**0.434**	**0.041**	**0.353**	**35.951**	**743.145**	**N/A**
B	33904.68	0.221	0.707	0.045	0.597	81.053	N/A	N/A
C	711429.8	0.024	0.301	0.791	0.486	N/A	103	N/A
D	706984.6	0.007	0.881	0.721	0.696	N/A	N/A	N/A
E	N/A	N/A	N/A	N/A	N/A	N/A	1963	N/A
F	9189272	0.004	N/A	N/A	N/A	N/A	N/A	1
G	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 6.2: Tuned Parameters

Each configuration had 30 iterations within `irace`. The parameters tuned by `irace` are listed in Table 6.1: temperature, cooling rate, selection rate, mutation phase 1 rate, mutation phase 2 rate, VNS internal iterations, and VNS sequential iterations. All required parameters are tuned to each algorithm configuration: e.g. cooling rate does not apply to 'VNS only' or the baseline; `vnsintr` only applies when VNS Internal is enabled. Note that Table 6.1 shows the parameters for `irace` for a single test (specifically ER&SR with VNS internally), as an example.

These dependencies are reflected in Table 6.2, the tuned configurations found by `irace`. Here, 'N/A' is used when a variable is not tuned. The cooling rate was limited to a maximum of 0.5 due to previous observations during testing finding that too few iterations caused by a higher cooling rate caused poorer results. Similarly, the VNS internal number of iterations was capped at 100, as higher values caused a dramatic increase in run-time.

After the parameters were tuned with `irace`, the algorithms were then run thirty times with these parameters. The mean values for these tests can be seen in Table 6.3. A box-plot showing the range of fitness values can be viewed in

Config	ER&SR	VNS Internal	VNS Sequential	Simulated Annealing	Average Fitness
A	**1**	**1**	**1**	0	**0.55993**
B	1	1	0	0	0.54985
C	1	0	1	0	0.54227
D	1	0	0	0	0.51884
E	0	0	1	0	0.49834
F	0	0	0	1	0.44840
G	0	0	0	0	0.42987

Table 6.3: Average Fitness per Run Type

Fig. 6.2. The results shown in this chapter reference a single region and around 200 employees being scheduled. HC2 and HC3 have no maximum bound, allowing any number of changes — which both increases the complexity of the problem but also increases the chance of finding better SC compliant allocations. The demand forecast data used for testing is provided by the FSO and is calculated by an internal research team for the region the employees tested work.

The SA metaheuristic, given by Config F, has the most variability in terms of fitness provided by the SCs. Across all of the configurations that use ER&SR, interestingly there is no solution generated which is worse than the baseline, but that is likely due to the high number of iterations present in the run configuration parameters provided by `irace` (both directly by VNS iterations and indirectly via temperature & cooling rate values).

A non-parametric ANOVA test (Kruskal-Wallis) was applied to the results and returned a p-value of $p < 2.2e^{-16}$. The datasets were confirmed visually as normally distributed in histogram plots as well as in Q-Q plots, which is a plot of the quantiles of the two distributions. The Q-Q plot of Configuration

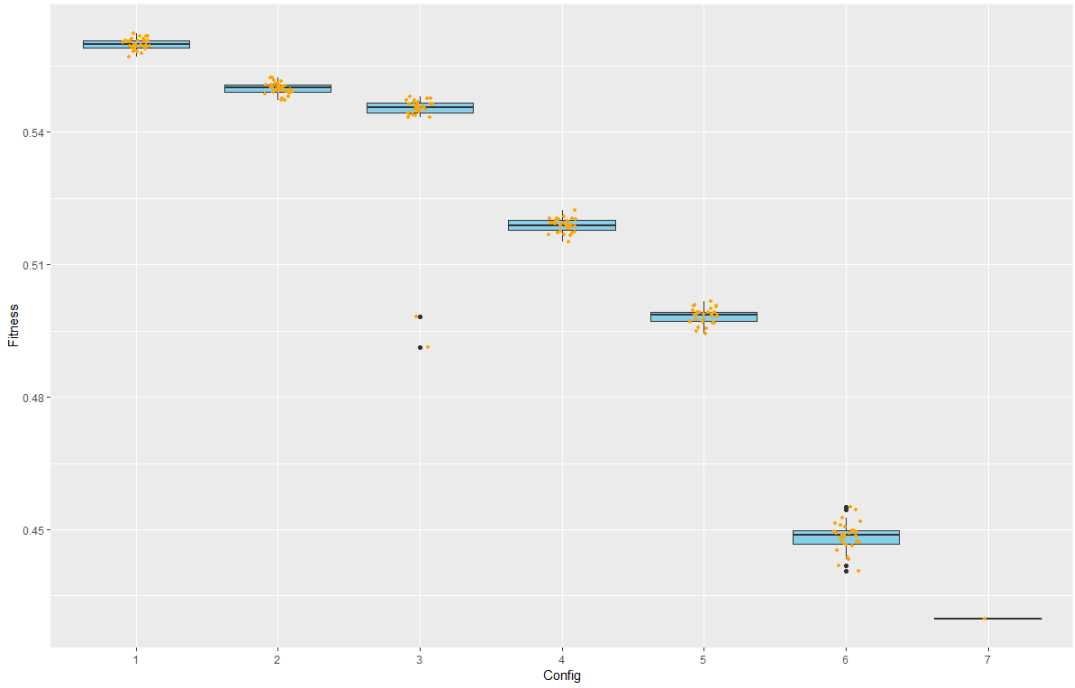


Figure 6.2: Box-Plot of Average Fitness (A-F) and Baseline Fitness (G)

A can be viewed at Figure 6.3. The null hypothesis (that the results for each configuration are drawn from the same distribution) can be rejected.

There is clear indication that ER&SR provides better schedules, and more so when using the VNS module at any tested stage, than the schedules created by SA or by VNS alone. The results show that the method this chapter provides is an improved alternative for generating employee schedules compared to the baseline solution for this dataset and demand forecast. The schedules are improved in terms of average SC fitness, which is how the quality of a solution is judged. View Table 6.4 for comparison of demand hours met.

The results also show that this algorithm can provide a small but relatively quick improvement upon standard ER&SR, where using both internal and sequential VNS have the highest rate of improvement on this dataset. The results show that demand met is improved on average, based on an estimated demand forecast from the FSO.

1 Table headers, in full, are as follows: Configuration - Temperature - Cooling Rate - Selection Rate - Mutation Phase 1 Rate - Mutation Phase 2 Rate - Variable Neighbourhood Search Internal - Variable Neighbourhood Search Sequential - Simulated Annealing

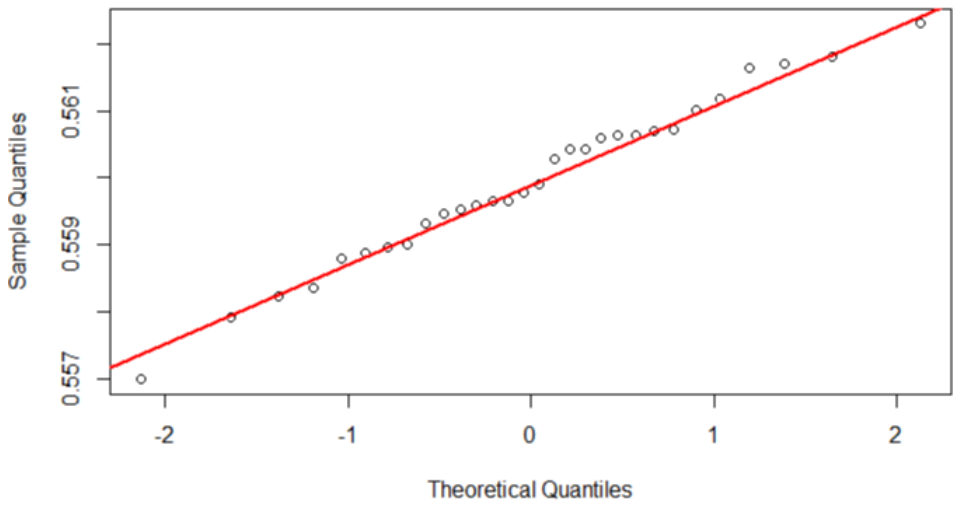


Figure 6.3: Q-Q Plot of Config. A

Config	Original Demand	Demand Hours Met	Demand Hours Not Met
A	**80305**	**55426**	**24879**
B	80305	55198	25107
C	80305	55180	25125
D	80305	54699	25606
E	80305	52980	27325
F	80305	51512	28793
G	80305	45373	34932

Table 6.4: Demand met per Config

6.3 CONCLUSIONS

This chapter presents a hybrid algorithm combining ER&SR with VNS and compares it to a SA approach and a basic VNS approach. This is applied to a real-world problem provided by an FSO company solving an instance of the employee rostering problem. The novelty of this approach lies in the improvement of the solution compared to standalone ER&SR. A further novelty is application of hyper-parameter tuning to ER&SR (via the `irace` algorithm configuration tool). This solution also provides an improved schedule for a real-world application when compared to the approach currently in place.

There is also scope for further research on ER&SR, in particular when hybridised with industry solvers such as CPLEX, which would include a mathematical formulation of the problem. There is also potential in replacing the stochastic rebuild section of ER&SR with a different mechanism, for example with an exact method with lowered bounds, other metaheuristics or matheuristics. While in this paper the number of VNS iterations are auto-configured by `irace`, an interesting future development may be a reverse EMCAC to control the depth of the search of the VNS so that more exploitative search can take place towards the end of run-time. This may be particularly useful as it was found during experimentation that the ruin method is very destructive early in run-time, negating any benefits of VNS. Additionally, future work should provide random seed trial search dynamics across methods tested to improve replicability of these stochastic algorithms.

7

CHAPTER 7 - ROSTER PATTERNS - EVOLUTIONARY RUIN & STOCHASTIC RECREATE HYBRIDISED WITH INTEGER PROGRAMMING

7.1 INTRODUCTION

In this chapter a novel matheuristic is defined and applied to a real-world instance of the employee rostering problem. The problem tackled is the roster pattern scheduling problem as refined in Chapter 2. The matheuristic is composed of ER&SR with IP. ER&SR was developed in [16] as an advancement from the original Ruin & Recreate algorithm [17], and IP is a mathematical optimisation technique derived from Linear Programming by Dantzig [84].

In Chapter 5 the personalised shift scheduling and employee rostering problem is solved using the ER&SR algorithm. Additionally ER&SR was hybridised with VNS in Chapter 6 with improved results than the standalone ER&SR. Continuing this path of research, this chapter implements ER&SR but replaces the *stochastic recreate* phase with an IP element. The precise data & specifications were provided by the FSO and specific problem details are intentionally omitted.

This collaboration with the FSO resulted in the design of precise constraints and fitness criteria. As this is a real-world problem, an empirically grounded view of the algorithm effectiveness is obtained in this study.

When aiming to enhance employee satisfaction, improve demand satisfaction and maximise the effectiveness of a workforce, optimising employee

schedules is an important consideration. If employee schedules are feasible but optimisation of the schedules is not considered, then an irrevocable loss of efficiency is likely, with employees working skills they do not prefer, and demand not being met.

Hybridisation of ER&SR with IP was chosen due to its proven record tackling employee rostering problems in the past. Recently, a mixed IP approach was utilised to tackle a skill mix and training schedule for aircraft maintenance [85]. Similarly, a two-stage stochastic IP approach was implemented to tackle integrated staffing and scheduling with application to nurse management [86].

Personnel scheduling is often used interchangeably with employee scheduling and workforce rostering. In this chapter the term 'employee rostering' is used to mean the allocation of an employee to a shift or set of shifts which are preset and immutable.

Employee rostering and personnel scheduling has been tackled with a variety of other evolutionary methodologies in the past. For example, VNS was used to tackle a field service operations employee scheduling problem in [87], and in [88] a memetic algorithm was employed to solve a personnel scheduling problem that maximises employee substitutability. Metaheuristic and evolutionary approaches are popular in personnel rostering problems due to their ability to quickly find comparatively good solutions to problems. It is also important to note that reaching an optimal solution is a worthwhile cause academically, in real-world problems slight percentage differences in fitness values are often not worth the computation time. Additionally, in the UK, an average worker takes 4.3 sick days per year [89], meaning solutions produced for employee schedules will rarely be worked as planned, especially in larger workforces. Therefore, a near optimal solution is a worthy goal for real-world applications. For further examination of IP and general hybrid solvers, there is a more in-depth examination of the field in Chapter 3.

Metaheuristic performance is heavily linked with well-tuned parameters [4]. As such the `irace` package [90] was utilised to search for the most appropriate parameters during testing.
The contributions this chapter presents are detailed below:

1. Propose the hybridisation of ER&SR with IP;
2. Analysis of this hybrid algorithm's application to a real-world scheduling problem;
3. Comparison of the base ER&SR algorithm to the proposed hybrid, and standalone IP.

This rest of this chapter is structured in the following manner: section 7.2 details the solution implementation, using the theoretical framework provided by [16] as a basis, and the unique elements of this implementation are described. The results of this approach are presented and analysed in Section 7.3. This chapter concludes with Section 7.4 where the research is summarised and recognition to funders and partners on this project is given.

7.2 IMPLEMENTATION

In this section the steps of ER&SR with IP are described. For a theoretical framework of ER&SR (which this algorithm was derived from), see [16]. A metaheuristic is well-placed to find a good result within the desired timeframe. ER&SR is an evolutionary algorithm with a proven track record to solve real-world scheduling problems. An IP rebuild method was implemented within the ER&SR algorithm to enhance results produced by utilising the power of an exact solver using IP. Before proceeding into written detail, a visual overview can be referenced in Figure 7.1.

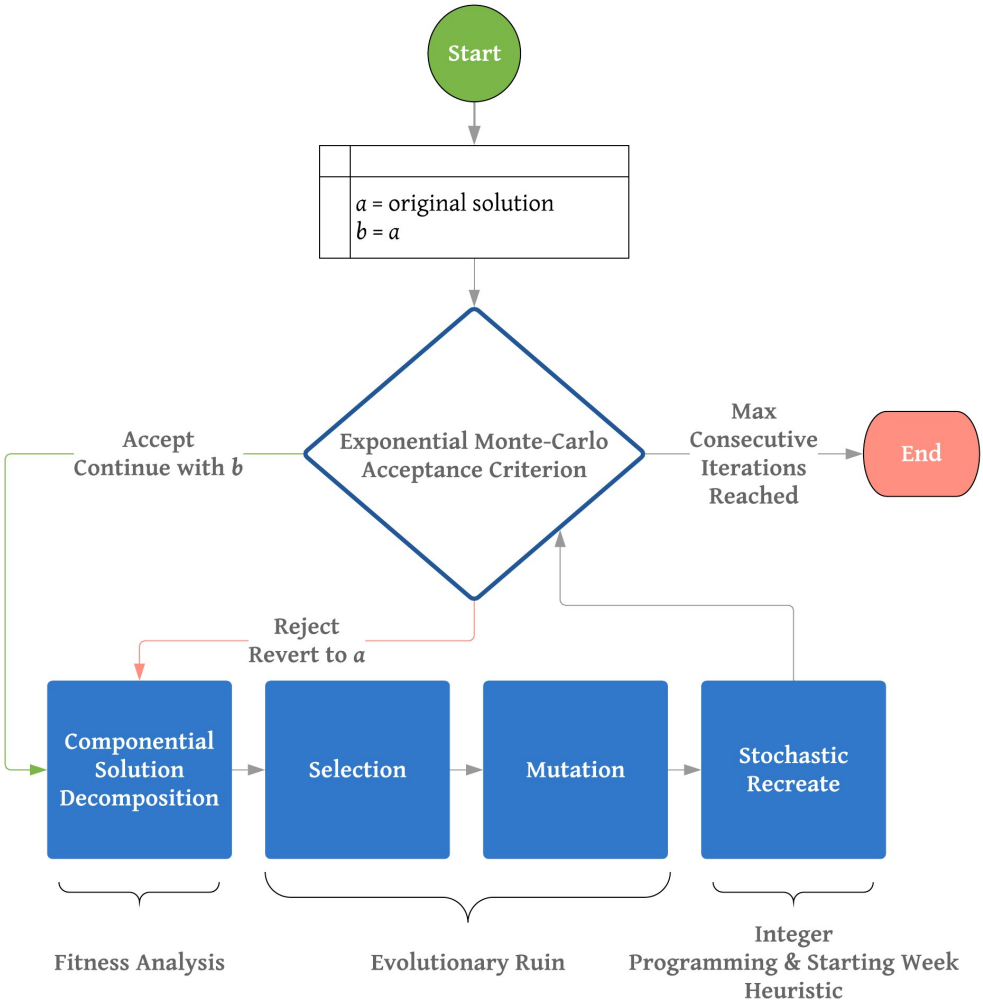


Figure 7.1: ERSR & IP Diagram

As described in Chapter 2, this problem requires selection of one RP per employee, from a set of RPs which each employee have indicated a preference for — or at least be willing to work. In order to maintain HC4, the search space is reduced to only ever allocate employees to these roster patterns, regardless of potential improvements from selected others.

As the algorithm is executed the number of RP changes (defined as number of employees not on their original RP), and the number of starting week changes (defined as number of employees still on their original RP but not starting the shift on the first week) increases. If this is capped (in order to

remain feasible and satisfy HC2 and HC3) only currently modified employee RPs and starting weeks can be again modified. This does not incur an additional change, as even if employee has moved from their original RP to another, then a third, the algorithm only counts number of employees not on their original patterns. This logic also applies to starting week changes. If the algorithm cannot escape a local optima due to this, or any other reason, the mutation operator compensates for this.

RPs can be of varying lengths - generally ranging from one week to fifty-two weeks in length. If an RP is too long for a scheduling period, for example say $|RP| = 52$ but $|demand| = 12$, RP length would be reduced to 12 also. Similarly, if the length of demand is for example $|demand| = 52$ but the $|RP| = 12$, then RP days would be repeated until the internal representation of the RP would be of length 52.

The rest of this section will refer to sections of pseudocode in algorithm 7.1 in the following manner: [7.1 - Line Number].

The initial schedule is the handwritten schedule provided by the FSO. This is imported into the algorithm along with the required parameters [7.1 - 1-6]. This original schedule comprises the original allocation for every employee, as provided by line managers [7.1 - 6].

The fitness object f is declared [7.1 - 8]. The solution is analysed before the acceptance criterion loop begins to prevent fitness of the same solution being analysed more than once - this would be most inefficient as decomposition of the schedule is the most costly segment of the metaheuristic. When fitness is calculated, individual employee shift allocations are analysed componentially. This information is stored for later use, and a weighted average of all SCs is calculated.

After initialisation, the acceptance criterion loop begins. In the testing conducted for this chapter, EMCAC is used (Li et al in [15] suggest that most known acceptance criterion methodologies should suffice). A copy of the current schedule is stored for instances when changes are not accepted and there is a need to revert to the previous solution.

Exploration for a new schedule begins by selection [7.1 - 12]. The probability of an employee being removed from an RP is as follows, where P is probability, s is selection, r is a random number between 0 and 1, and x_1 is a parameter used to control the rate of selection based on user or researcher preference:

$$P(s) = r * x_1 \tag{7.1}$$

Additionally, to the selection operator, mutation can also occur to ruin elements of the schedule [7.1 - 13]. The probability of an employee being removed from an RP is as follows, where m is mutation, r is a new random number between 0 and 1 and x_2 is a parameter allowing the user to control rate of mutation:

Algorithm 7.1 Evolutionary Ruin & Stochastic Recreate with Integer Programming

```
1: t ← getParameterisedTemperature()
2: δ ← getParameterisedDelta()
3: cr ← getParameterisedCoolingRate()
4: sr ← getParameterisedSelectionRate()
5: mr ← getParameterisedMutationRate()
6: a ← originalSolutionFromFSO()
7: t′ ← t
8: f ← decomposition(W) //W are parameterised weightings. Solution decomposition complete.
9: while t′ > δ do //Exponential Monte-Carlo Acceptance
10:    b ← a.copy()
11:    f ← decomposition(W) //W are parameterised weightings. Solution decomposition complete.
12:    a ← selection(a, sr)
13:    a ← mutation(a, mr) //Evolutionary ruin complete.
14:    a ← rebuildWithCplex(a)
15:    a ← startingWeekModifier(a) //Stochastic recreate complete.
16:    p ← getAcceptanceProbability()
17:    if p < r then //Random int
18:       a ← b.copy()
19:    end if//Solution Acceptance complete.
20:    t′ *= cr
21: end while
```

$$P(m) = r * x_2 \tag{7.2}$$

At this point in run-time the schedule will be infeasible [7.1 - 14]. The key novelty of this algorithm is in the implementation of the recreate phase. Due to the massive search space being infeasible to solve using an exact solver by an exact solver alone, or by exhaustive search, metaheuristics are generally accepted as a preferable alternative. This approach is commonly termed a matheuristic. The underpinning hypothesis for this approach is that using a metaheuristic to mostly solve the problem and using an exact solver to find the optimal solution to the sub-problem, would provide better solutions than the metaheuristic alone. However, during implementation testing it was found that running the solver for any more than 12 employees for a 3 month scheduling period took an amount of time greater than 3 days run-time. Considering the test data being around 180 employees, this was found to be an impractical approach. Thus, a parameter was introduced to prevent too many employees being scheduled by the exact solver portion of the matheuristic, and allowing the metaheuristic to solve the rest. It should be noted that if CPLEX is parameterised with a percentage of employees above 0%, it will always schedule at least one employee per iteration, by design. These results are described in detail in section 7.3.

In [7.1 - 14], CPLEX is called to solve the sub-problem consisting of the employees grouped in i. Being an exact solver, CPLEX returns the optimal solution to this sub-problem. Depending on the number of employees in the region being analysed, and on the parameter N, this step is the most computationally intensive (in terms of run-time and processor activity).

All other employees are returned to their original roster pattern, then given a random starting week [7.1 - 15] by the metaheuristic. While the allocation of starting weeks is random, the allocation of skills per hour follow a series of preference based considerations. This will be described here textually, but the

concepts are perhaps best understood graphically. As such, this process can be viewed in Figure 7.2.

This rebuild component of the metaheuristic only considers employees which are not already allocated by the IP solver, CPLEX, as the solver will also assign optimal skills to shifts for the particular employees being considered.

First, all employees which have just been allocated fresh RPs are input to this skill allocation function to provide ideal skills per hour. The choice was between random allocation or something more intelligent. Due to the fairly simple set-covering nature of the problem, a basic heuristic like this can provide higher quality results than random allocations.

In order to not favour one employee over another, each repair involves a single random employee, on a random hour needing allocated, on a random day, and this selection is repeated until all employee hours have skill allocations.

Once a random hour, day and employee are selected, a random skill required in demand for that period is selected. If the employee has this skill, they are assigned, and the process starts anew. Otherwise, another random skill required on that hour and day is selected. This repeats until either the employee is allocated a skill which there is demand for, or all demand has been tried.

If the employee has no suitable skills for current demand, the algorithm begins considering all demand already fulfilled, as it is preferable to have an overhead of employees available for specific skills, for considerations regarding annual leave, sickness, etc. However, if the employee has no skills that have been fulfilled as well as no skills currently needed, the employee is set to work any skill they have at random. This fulfils the requirement for HC6, ensuring they are assigned to all contractually required shifts, even if the fitness is not improved by this action.

If instead the employee does have skills which were originally required on this day and hour, the employee is allocated one. If plural, the specific skill chosen is dependent on a distributed random probability. This way, higher demand has higher overhead than lower demand. Again, this is not a SC and does not help fitness, but is an acknowledgement of the real-world nature of this problem.

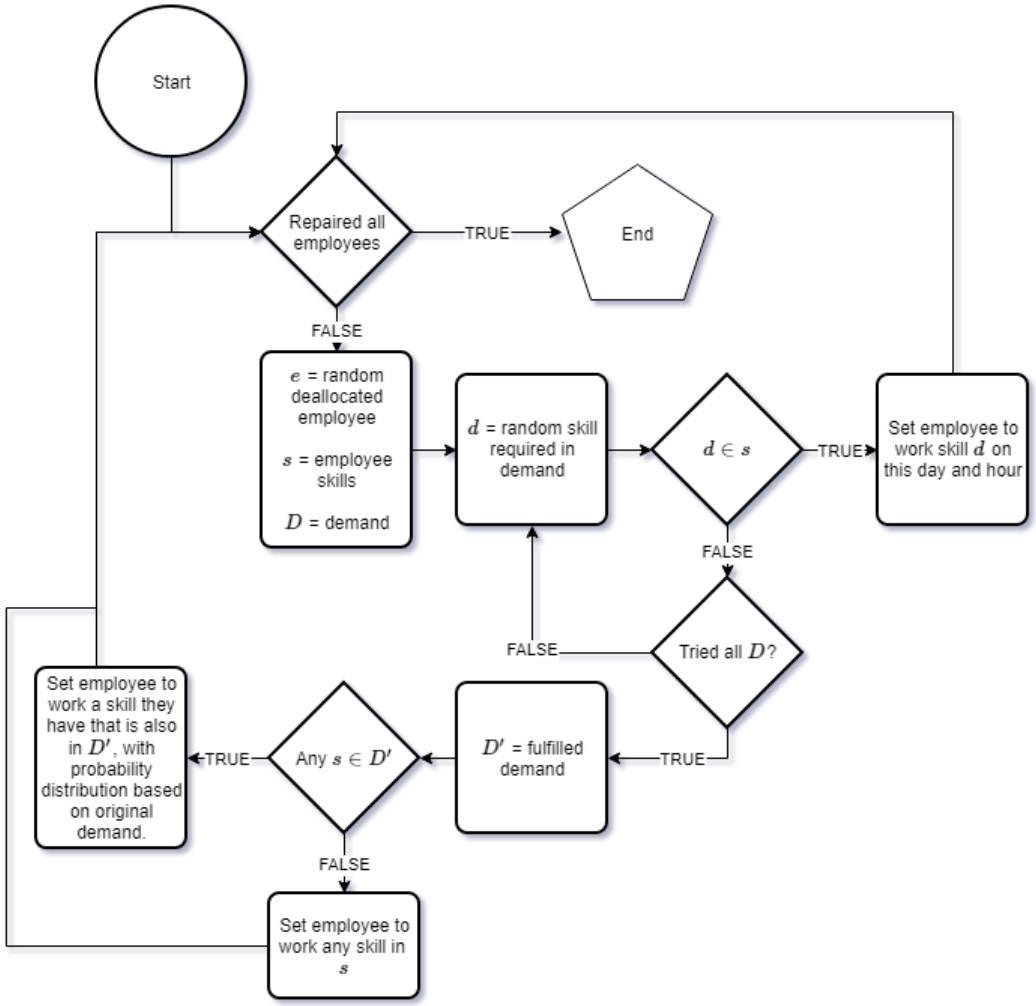


Figure 7.2: Metaheuristic Based Skill Allocations

Fitness of the newly ruined and rebuilt solution takes place [7.1 - 17], again considering the SC weightings. The algorithm will then probabilistically accept the changes [7.1 - 21-22]. This uses EMCAC, and is calculated as follows:

$$P = \exp((o - n)/t) > r \tag{7.3}$$

P is the probability of accepting a worsened state, o is the original solution energy cost (modelled as inverse fitness), n is the energy cost of the new solution, t is current temperature and r is a uniformly generated random number between 0 and 1.

If p is found to be less than a randomly generated number between 0 and 1, the schedule is reverted to the original state in the previous iteration (or the original hand-written schedule, if the first iteration). Due to the nature of EMCAC, the chance of a worsened solution being accepted at the start of run-time is high, declining to nil chance towards the end of run-time. This encourages exploration earlier in run-time, and exploitation later - depending on parameter selection and data structure.

7.3 RESULTS

Presented in this section are the results from tests run on an Intel i7 2.20GHz Central Processing Unit (CPU), using Java SE 1.8 to build the metaheuristic and the CPLEX 12.8.0 exact solver in hybridity.
The Configurations tested were:

(A) ER&SR with IP

(B) ER&SR only

(C) IP only

(D) SA only

(E) Baseline

Initial testing included IP only and was conducted before the bulk of implementation, as there is little need for a metaheuristic if an exact solver can provide an optimal solution in an acceptable time. However, testing revealed

that unless the constraints are relaxed to the point of no longer having any tangible effect, CPLEX cannot provide optimal solutions to the problem described in 2.

The baseline is also considered in the results for comparison; the baseline is the fitness computed from the manually generated solution in place at the FSO. The baseline solution is always feasible.

Metaheuristic effectiveness is largely dependent on well-tuned parameters [4]. It is important to carefully select parameters to find solutions in a reasonable time-frame, as required by the FSO. In order to achieve a good quality solution in short time frames the `irace` parameter configuration tool [62] was utilised. Automated parameter tuning is preferable to onerous manual configuration [83], especially when the parameters require specialist knowledge to understand. During `irace` testing, the testing of elite configurations was enabled, meaning that only the best configurations found during the run of `irace`, the final best, will be used in the testing phase. The first elimination test value was set to the default of five. The default statistical test configuration for `irace` was used (F-test), and `irace` was allowed a 5000 experiment cap. [1]

A time limit was tested, but found to be inadequate at controlling CPLEX from the `irace` target runner. Instead, a `cplex` parameter was added, which dictates the maximum percentage of employees to be scheduled by CPLEX in a single iteration, and allows the base metaheuristic to schedule the remaining employees. This solved the issue of excessive run-time, which can be seen in Fig 7.3 where there is an exponential increase in run-time correlated with the percentage of employees being scheduled by CPLEX. The reason for this is likely due to percentage of employees being dependent on the number of employees selected or ruined during the evolutionary ruin phase, and this variance becomes more apparent with higher percentages.

1 A higher limit was tested, but the combination of `R`, Java and `irace` used excessive quantities of RAM and crashed

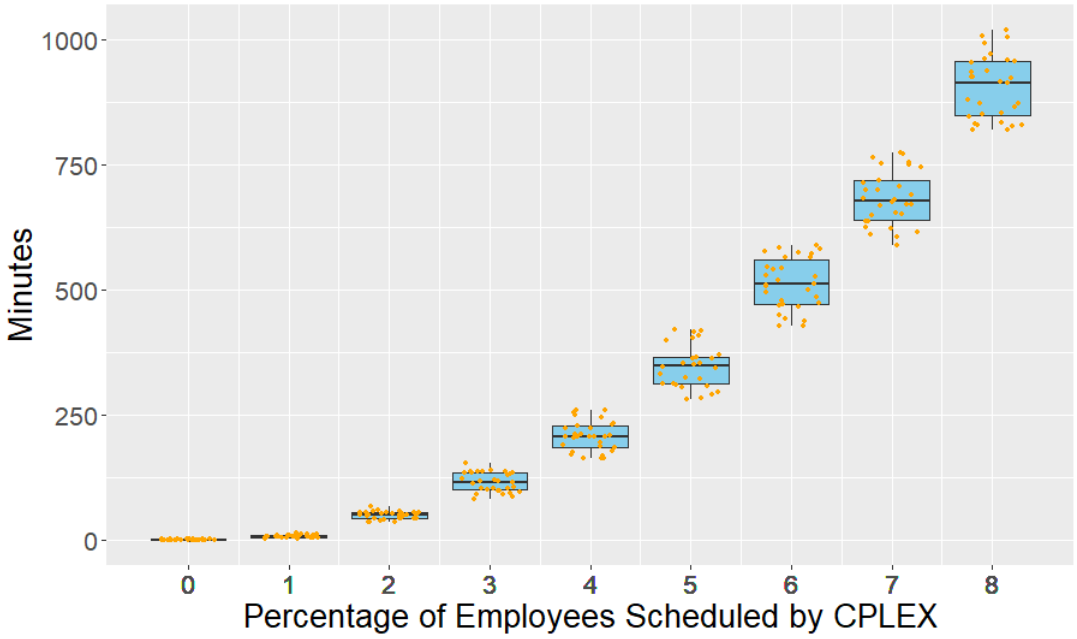


Figure 7.3: Run-time compared with Percentage of Employees CPLEX Tasked With Scheduling

The specific parameters which were provided to `irace` for tuning are described in Table 7.1. No optional parameters were necessary and are therefore not displayed.

An interesting query raised by the FSO during testing — what has the greatest effect on fitness, RP changes or starting week changes? [2] To test this, every possible whole number percentage of changes allowed from each parameter (excluding 0% RP changes and 0% staring week changes) were compared. The initial results can be viewed in Figure 7.4. This showed that for the most part the changes to fitness were not massively different — which is an interesting find, and useful to the FSO, as it means fewer changes are needed to attain a mostly good result. A small number of starting week changes are more beneficial than a small number of RP changes.

However, when the outliers are removed (see Fig 7.5), there is a clear correlation between fitness and percentage of changes allowed, for both RPs

2 We assume the FSO meant "In an ideal situation with tuned parameters over 30 tests, which of the two variable changes performed the best?"

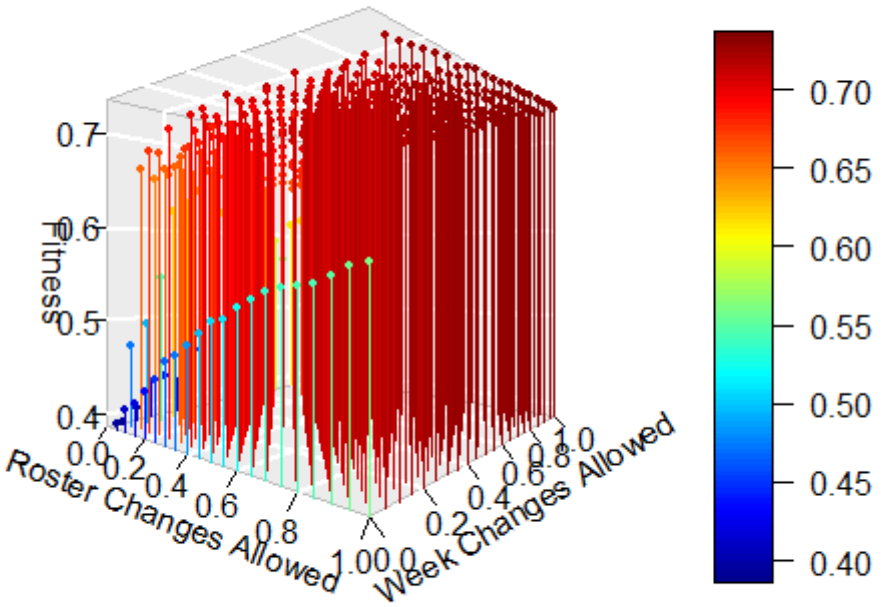


Figure 7.4: RP Changes Parameter Vs Starting Week Parameter

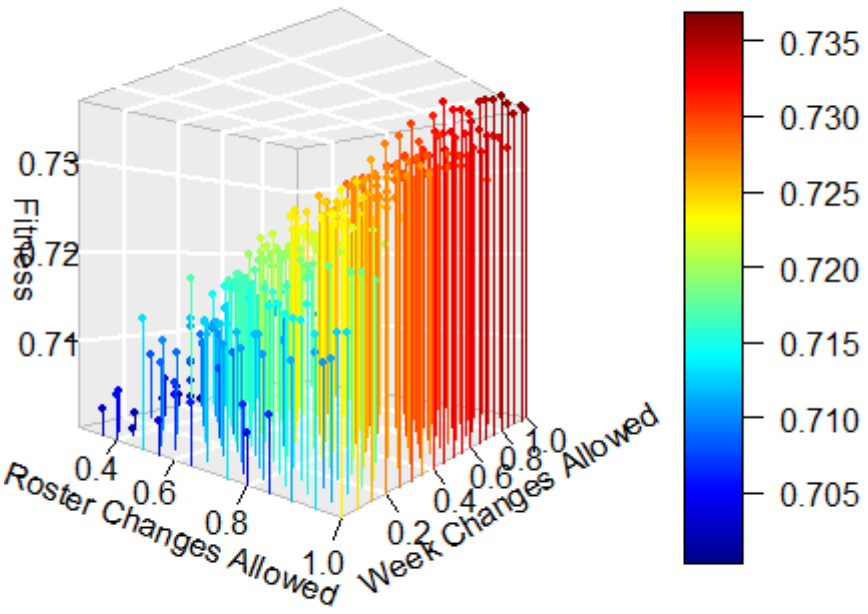


Figure 7.5: RP Changes Parameter Vs Starting Week Parameter (Outliers Removed)

Parameter	Reference	Type	Range
temp	–a	r	(0.0001,1000000)
cr	–b	r	(0.0001,0.99999)
sr	–c	r	(0.0001,1.0)
mr	–d	r	(0.0001,1.0)
delta	–e	r	(0.000001, 10)
cplex	–g	r	(1.0,815.0)

Table 7.1: Parameters To Be Tuned by irace

Temp	**CR**	**SR**	**MR**	δ	**CPLEX**	**Fitness**
388527.2672	0.2156	0.8845	0.3373	0.0234	7.6743	0.7498
679610.9251	0.3851	0.9654	0.5419	0.0101	7.4710	0.7421
289172.3473	0.2492	0.8991	0.3400	0.0129	7.5691	0.7402
301493.5945	0.2234	0.9235	0.4531	0.1617	7.4905	0.7389
421419.0231	0.4169	0.8846	0.2098	0.0335	7.2732	0.7330

Table 7.2: Top 5 Parameter Configurations

and starting weeks. Modifying employee starting weeks provides more fitness improvement than RPs, but if both more RP changes and starting week changes are allowed then fitness can reach a higher level.

After 5000 experiments, `irace` produced tuned parameters of the best configurations found. The top 5 of these — in terms of fitness — for Configuration A (the proposed hybrid algorithm) can be viewed in Table 7.2.

Similar testing was conducted for the other configurations (B & D) and the results of the tests can be viewed alongside the best parameters found by `irace` in Table 7.3.

Config	Temperature	CR	SR	MR	δ	cplex
A	**388527.2672**	**0.2156**	**0.8845**	**0.3373**	**0.0234**	**7.9197**
B	136782.5277	0.7646	0.1065	0.9867	8.6166	N/A
C	N/A	N/A	N/A	N/A	N/A	100
D	331916.4221	0.8934	N/A	N/A	4.8860	N/A
E	N/A	N/A	N/A	N/A	N/A	N/A

Table 7.3: Tuned Parameters

Testing between the various configurations (except for the removed configuration C, as it is infeasible due to time limits) and the baseline can be viewed in Fig 7.6. Unsurprisingly, the baseline ranked worst when compared to any of the optimisation techniques. SA provided a significant improvement in fitness values to the baseline, and ER&SR is a nearly equivalent improvement from SA. Most noteworthy is the new hybrid ER&SR with IP, which has provided a clear improvement from the base ER&SR, and the base IP (though any feasible solution is better than no solution).

A non-parametric ANOVA test (Kruskal-Wallis) was applied to the results and returned a p-value of 0.4651, therefore the null hypothesis (that the results for each configuration are drawn from the same distribution) cannot be rejected. In further investigation of normality, the data-sets were confirmed visually as normally distributed using histogram plots as well as using Q-Q plots, which is a plot of the quantiles of the two distributions. The Q-Q plot of each configuration can be viewed in Figures 7.7 (configuration A), 7.8 (configuration B) and 7.9 (configuration C).

Unlike the results in Chapter 6, there is little to be gained showing the demand met per configuration, as demand is met by every configuration (except the baseline) fully due to the number of employees in the region tested,

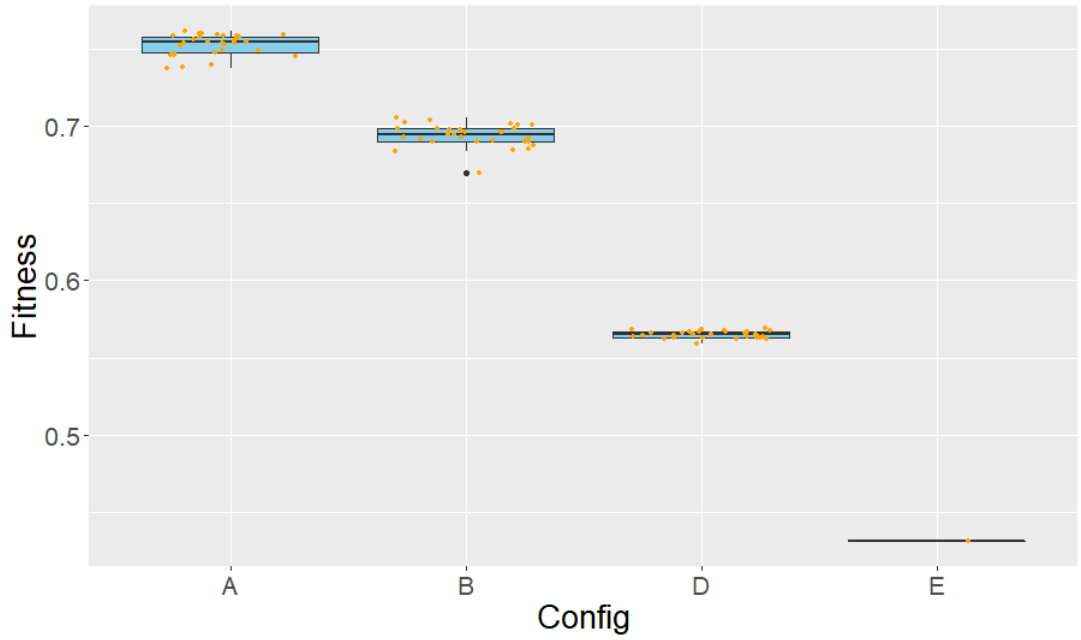


Figure 7.6: Boxplot of Each Configuration Tested 30x With Tuned Parameters

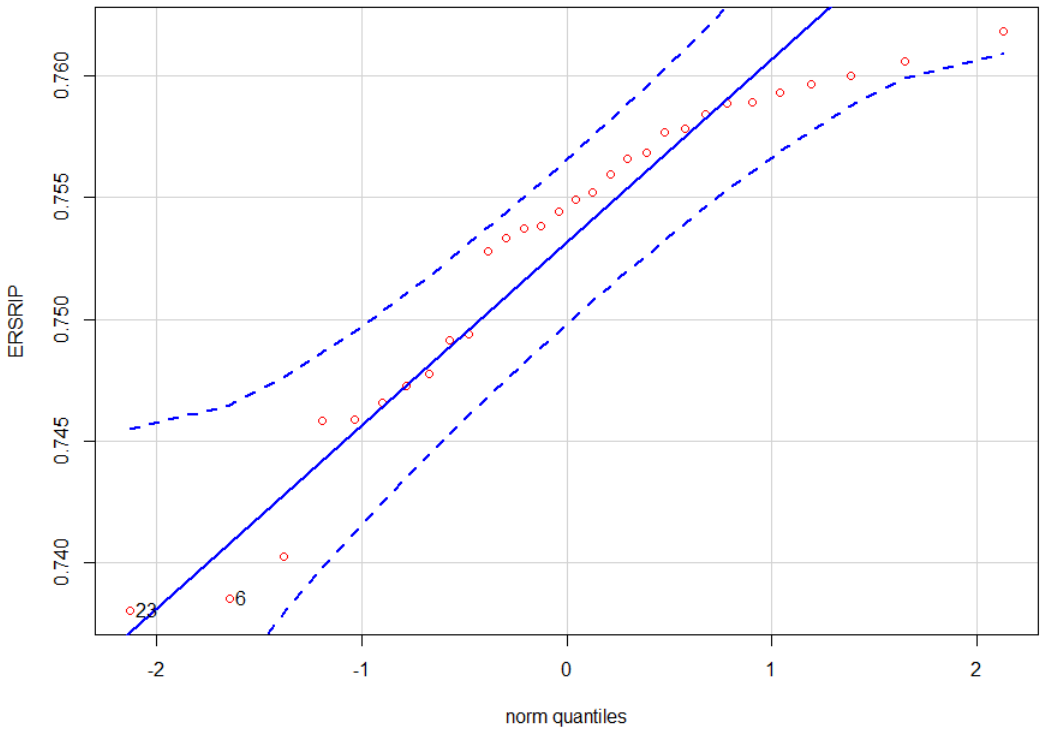


Figure 7.7: Q-Q Plot of Configuration A

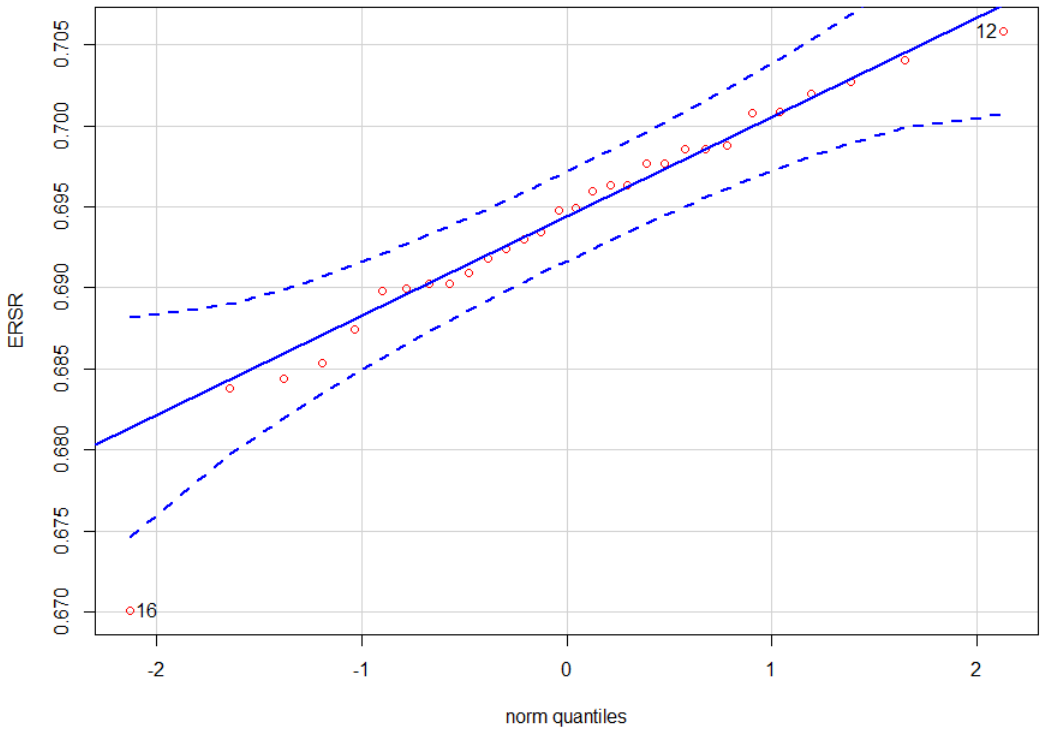


Figure 7.8: Q-Q Plot of Configuration B

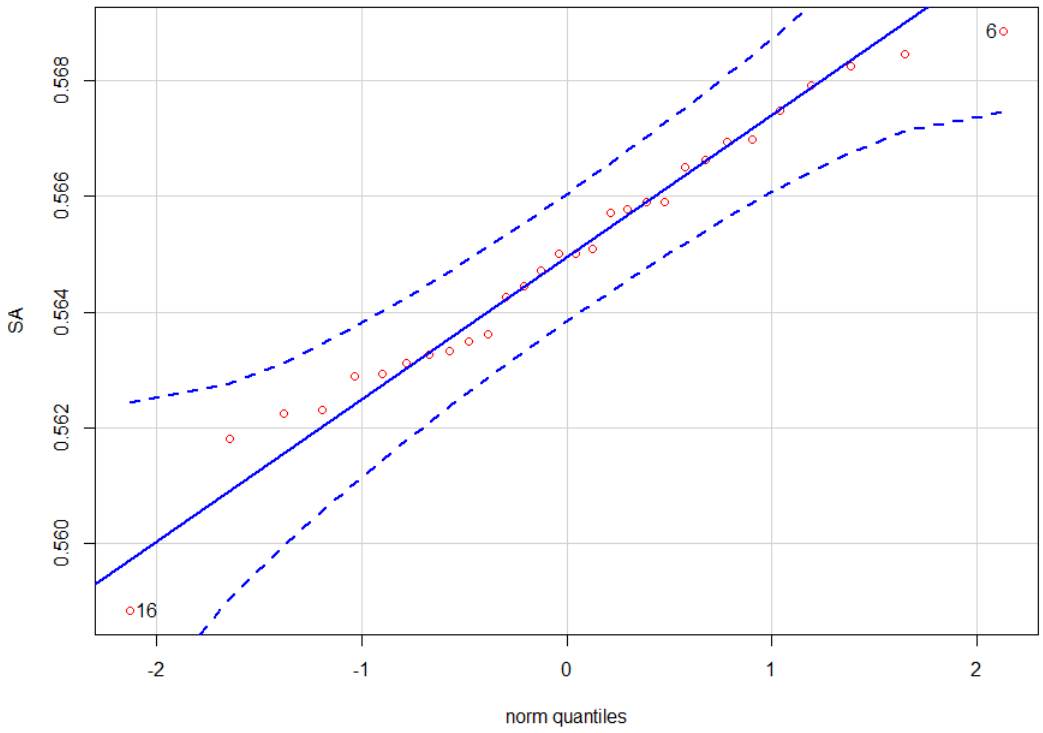


Figure 7.9: Q-Q Plot of Configuration D

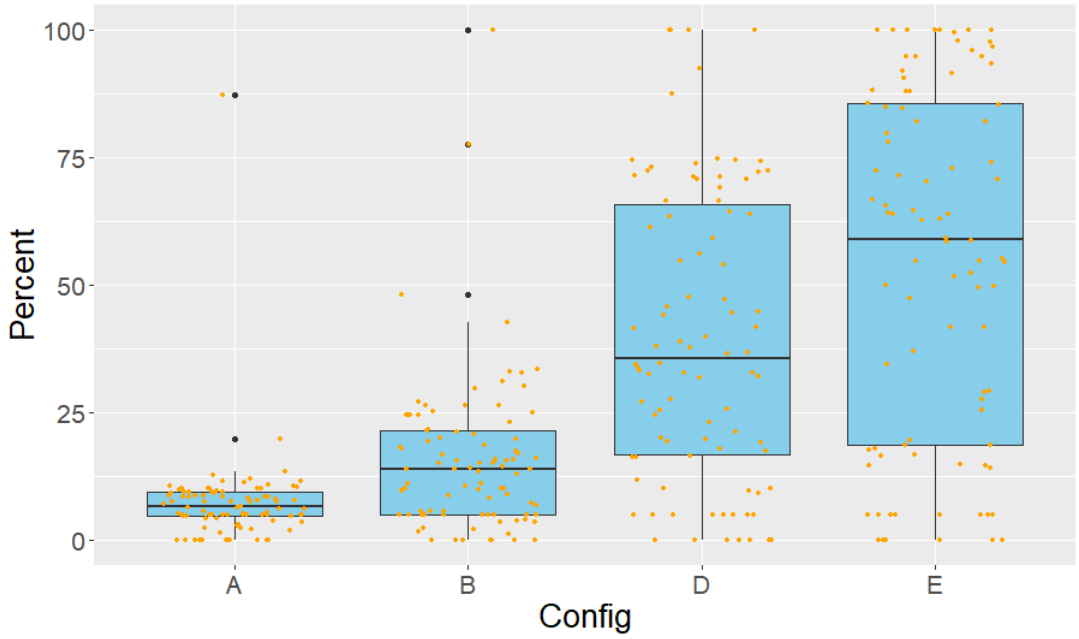


Figure 7.10: Average Absolute Percentage Difference Per Config

and the forecast demand being particularly low in this region. However, SC4 calculates the absolute percentage difference each day is from the average percentage of demand coverage (absolute so that whether lacking employee allocations, or having too many, these are equally undesirable). This is an interesting perspective for quality of a schedule and can be viewed in Figure 7.10. The baseline has the most variability (see box-plot E in Figure 7.10) and performs the worst on average across average absolute percentage demand coverage. The proposed algorithm (Configuration A) performs the best, but still has one outlier. This, and the outliers in Configuration B were identified as being Saturdays, which are particularly difficult days to meet demand for. This is due to most employee contracts specifying a maximum number of Saturdays in a period, making the potential pool of preferred RPs for employees smaller than would be necessary to reduce the number of outliers.

The same data is shown as averages per day of week in figure 7.11, figure 7.12, figure 7.13 and figure 7.14. Sundays (#7 in the plots) have some demand which is only met by a small number of RPs. Generally, employees who are

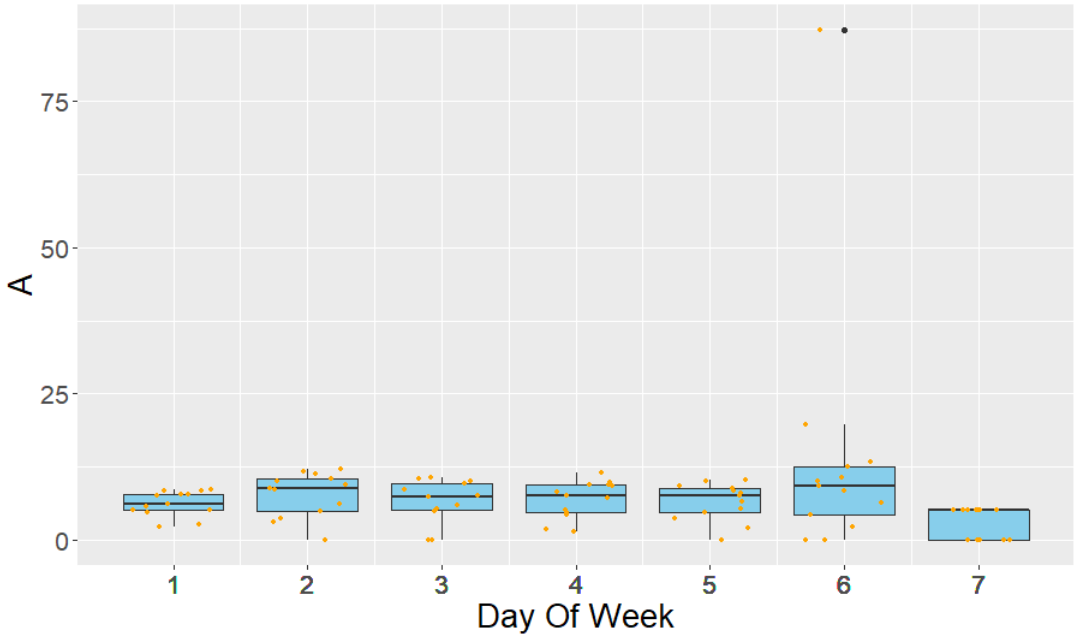


Figure 7.11: Day Of Week Absolute Percentage Difference - Config A

originally allocated these RPs do not have any other preferences, and therefore the absolute percentage demand coverage is static on Sundays (these RPs also have the same recurring one-week pattern, so changing starting week would have no effect).

The testing completed in this chapter indicates that the proposed ER&SR with IP algorithm can provide higher quality solutions than the standalone base components, and better than SA-produced schedules, for the region of employees and the forecast demand provided by the FSO. The higher fitness values from the proposed algorithm correlate with a higher percentage of employee RP allocations being scheduled by CPLEX, but this also correlates with a much higher computational time. For real-world usage, it is recommended that a Pareto front of fitness compared to computation time is calculated in order to select a preferred trade-off.

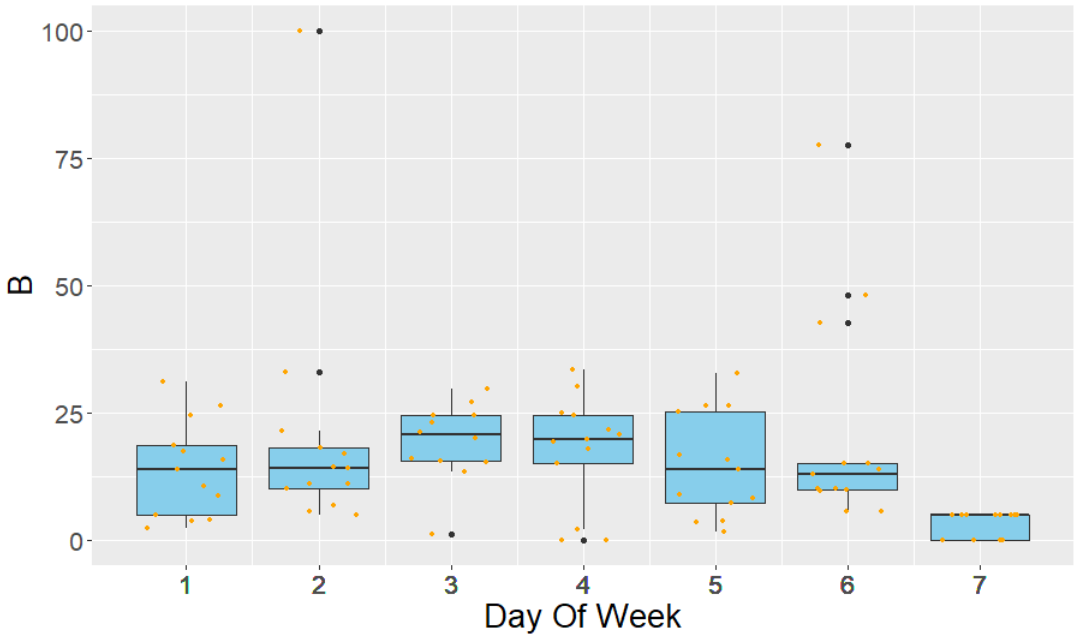


Figure 7.12: Day Of Week Absolute Percentage Difference - Config B

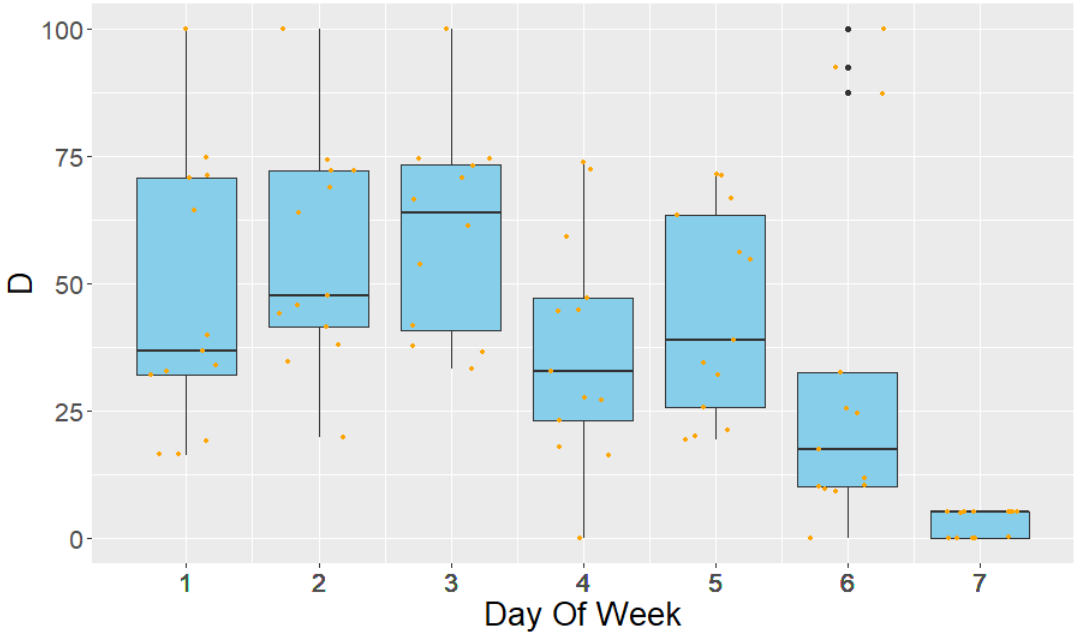


Figure 7.13: Day Of Week Absolute Percentage Difference - Config D

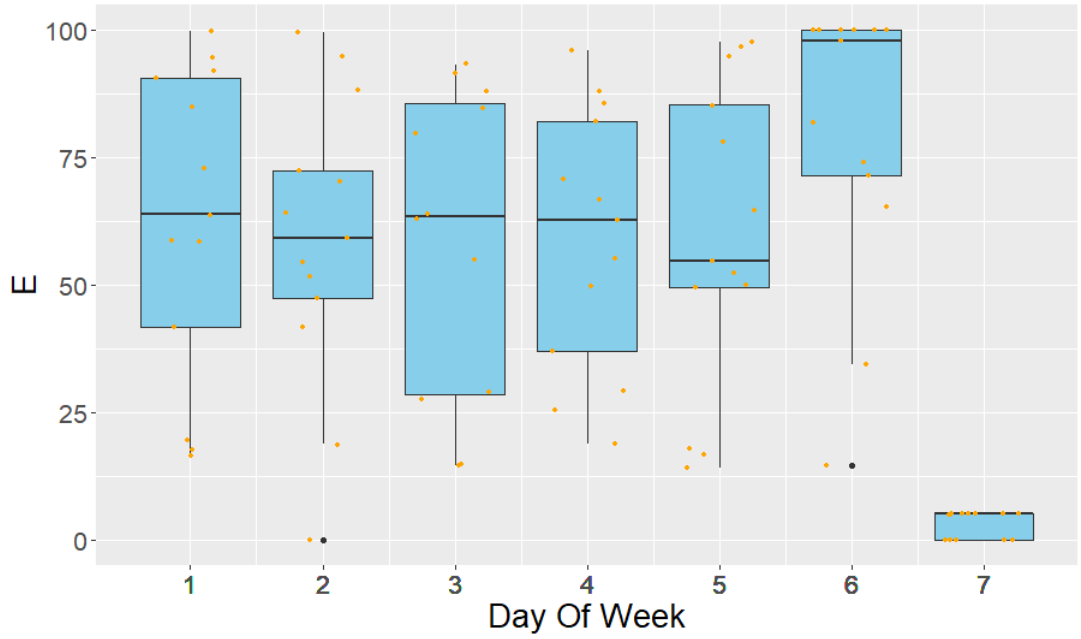


Figure 7.14: Day Of Week Absolute Percentage Difference - Config E

7.4 CONCLUSIONS

In this chapter a hybrid algorithm combining ER&SR with IP is presented and compared to its standalone components as well as to SA. The novelty of this chapter lies in the improvements to the recreate phase of the ER&SR algorithm, and providing improved schedule quality due to this. This matheuristic was applied to a real-world employee scheduling problem as provided by an FSO.

There is a variety of future work which could be seeded from this chapter. Hybridisation of metaheuristics in general is a field of work with much yet to be explored (as discussed in Chapter 3), particularly for use in real-world problems. In particular, this work has shown that a matheuristic approach can provide higher quality results than a base metaheuristic, even with a capped percentage of computation tasked to the exact solver. Additionally, there are a number of potential acceptance criteria which have not been utilised within the ER&SR algorithm, which may provide better results.

8

CHAPTER 8 - REAL WORLD USAGE

The work described in this thesis was sponsored by a field service operations company. Additionally, the problems described in Chapter 2 are formed from various real-world scenarios faced by the field service operations company. As such, the problem formulations — and the algorithms created to tackle said problems — have the benefit adding to the literature of personnel scheduling as well as providing solutions for real-world problems. In this chapter the various real-world implementations and integration of the algorithms described in this thesis are described. As with all previous chapters, the information in this chapter has some facts and figures redacted in order to respect data privacy laws and GDPR.

In this chapter two real-world uses of the research described in this thesis are discussed and shown visually, where possible.

8.1 ROSTER GENERATION

Personnel schedules are generated by the various algorithms described in chapters 4 through 7. These algorithms have been integrated into the FSOs infrastructure, allowing input from databases of forecasted demand data, employee skill; contract; preference data, and allow output of schedules for engineer and managerial usage.

The User Interface (UI) in the following screenshots was developed by other developers within the FSOs research team. In the examples within this section,

I developed the underlying algorithms (which in the integrated context I will refer to as the 'engine') and integrated the engine into the FSOs systems.

See Fig 8.1 for a view of the UI which is used mostly by managers of personnel in a region, and by researchers.

On the left of Fig 8.1 under "Inputs" the interface allows the user to choose from test data (synthetic data generated for the purpose of testing) database data (from various FSO servers and personnel databases) or from spreadsheet files. A pre-processing step of 'shrinkage' can be applied, which for the purpose of research in this thesis has been deemed out-of-scope as it can be applied to the input data before actually running the engine. Shrinkage in this case changes the demand requirements, and does not affect the number of employees available.

The "Tuning" section of Fig 8.1 allows the user to select the type of engine variant, such as VNS or ER&SR. The number of attempts can be modified so that the best schedule (according to weighted fitness) over multiple attempts can be presented. There is also an additional external parameter which can force the various algorithms to provide the best schedule found within a time-limit, as opposed to the specified internal acceptance criteria. Other parameters are available, based on the specific version of each engine selected (some are simplified for easier usage, reading this thesis is not a pre-requisite of running the software).

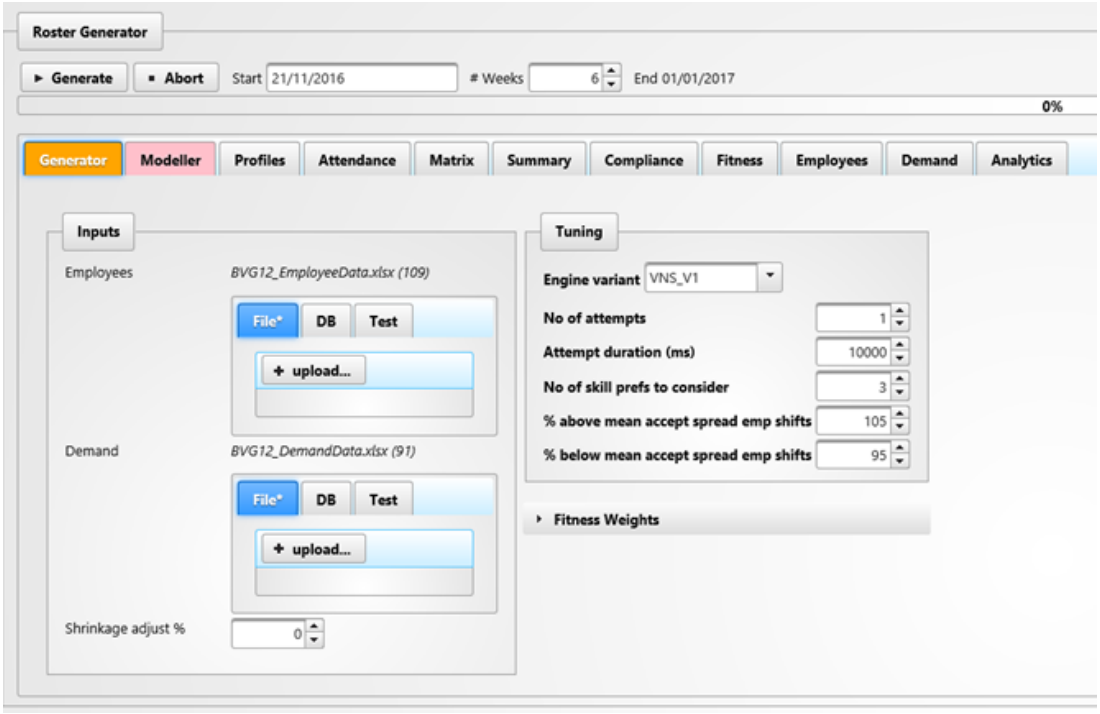


Figure 8.1: Managerial Initialisation Screen

After generating a schedule, a manager can look at a number of analytics and graphical overviews of the schedule. At this point the schedule is not necessarily accepted, and this tool may be used for guiding the generation of another schedule, or in testing various engine parameters before accepting a schedule. Such analytics can be viewed in Fig 8.2. In this example, there is a 3-month schedule which has been generated by the engine. The line / bar combo chart by default shows demand in comparison to coverage by shifts, for all skills. This view can be redrawn with a specific skill, or specific times or shift types. This can be useful if a manager is considering how much overtime would be required on specific days, for example.

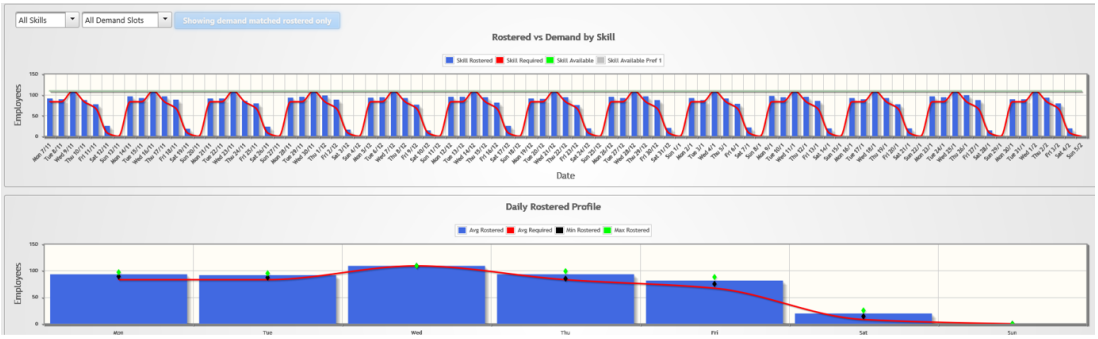
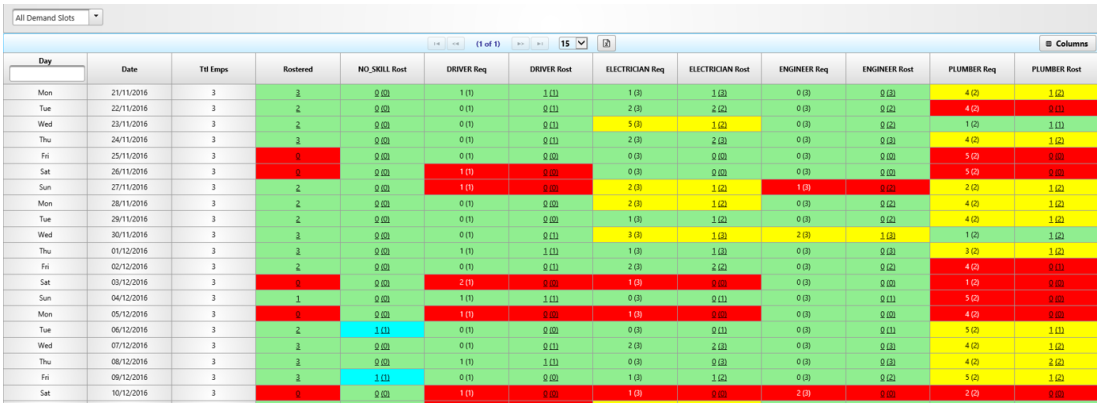


Figure 8.2: Managerial Schedule Overview Screen

If a manager wishes for a more detailed breakdown, the UI in Fig 8.3 would be appropriate. The colour coding scheme is as follows: green means demand has been satisfied on that time and day for a specific skill. Yellow means that there are technically employees available that could work this shift and skill, but would need shifts swapped with someone else (generally this is worse for fitness, but may be useful since the algorithms are for all regions, not specific unique circumstances). Red is used to tell the manager that it is impossible to meet demand on the highlighted time and day for that skill, regardless of rescheduling. The red shifts are where overtime is required, which is out-of-scope for the algorithms in this thesis. This software merely highlights it after the schedule generation.



Day	Date	Ttl Emps	Rostered	NO_SKILL Rost	DRIVER Req	DRIVER Rost	ELECTRICIAN Req	ELECTRICIAN Rost	ENGINEER Req	ENGINEER Rost	PLUMBER Req	PLUMBER Rost
Mon	21/11/2016	3	3	0 (0)	1 (1)	1 (1)	1 (3)	1 (3)	0 (3)	0 (3)	4 (2)	1 (2)
Tue	22/11/2016	3	2	0 (0)	0 (1)	0 (1)	2 (3)	2 (2)	0 (3)	0 (2)	4 (2)	0 (1)
Wed	23/11/2016	3	2	0 (0)	0 (1)	0 (1)	5 (3)	1 (2)	0 (3)	0 (2)	1 (2)	1 (1)
Thu	24/11/2016	3	3	0 (0)	0 (1)	0 (1)	2 (3)	2 (3)	0 (3)	0 (3)	4 (2)	1 (2)
Fri	25/11/2016	3	0	0 (0)	0 (1)	0 (0)	0 (3)	0 (0)	0 (3)	0 (0)	5 (2)	0 (0)
Sat	26/11/2016	3	0	0 (0)	1 (1)	0 (0)	0 (3)	0 (0)	0 (3)	0 (0)	5 (2)	0 (0)
Sun	27/11/2016	3	2	0 (0)	1 (1)	0 (0)	2 (3)	1 (2)	1 (3)	0 (2)	2 (2)	1 (2)
Mon	28/11/2016	3	2	0 (0)	0 (1)	0 (0)	2 (3)	1 (2)	0 (3)	0 (2)	4 (2)	1 (2)
Tue	29/11/2016	3	2	0 (0)	0 (1)	0 (0)	1 (3)	1 (2)	0 (3)	0 (2)	4 (2)	1 (2)
Wed	30/11/2016	3	3	0 (0)	0 (1)	0 (1)	3 (3)	1 (3)	2 (3)	1 (3)	1 (2)	1 (2)
Thu	01/12/2016	3	3	0 (0)	1 (1)	1 (1)	1 (3)	1 (3)	0 (3)	0 (3)	3 (2)	1 (2)
Fri	02/12/2016	3	2	0 (0)	0 (1)	0 (1)	2 (3)	2 (2)	0 (3)	0 (2)	4 (2)	0 (1)
Sat	03/12/2016	3	0	0 (0)	2 (1)	0 (0)	1 (3)	0 (0)	0 (3)	0 (0)	1 (2)	0 (0)
Sun	04/12/2016	3	1	0 (0)	1 (1)	1 (1)	0 (3)	0 (1)	0 (3)	0 (1)	5 (2)	0 (0)
Mon	05/12/2016	3	0	0 (0)	1 (1)	0 (0)	1 (3)	0 (0)	0 (3)	0 (0)	4 (2)	0 (0)
Tue	06/12/2016	3	2	1 (1)	0 (1)	0 (0)	0 (3)	0 (1)	0 (3)	0 (1)	5 (2)	1 (1)
Wed	07/12/2016	3	3	0 (0)	0 (1)	0 (1)	2 (3)	2 (3)	0 (3)	0 (3)	4 (2)	1 (2)
Thu	08/12/2016	3	3	0 (0)	1 (1)	1 (1)	0 (3)	0 (3)	0 (3)	0 (3)	4 (2)	2 (2)
Fri	09/12/2016	3	3	1 (1)	0 (1)	0 (0)	1 (3)	1 (2)	0 (3)	0 (2)	5 (2)	1 (2)
Sat	10/12/2016	3	0	0 (0)	1 (1)	0 (0)	1 (3)	0 (0)	2 (3)	0 (0)	2 (2)	0 (0)

Figure 8.3: Managerial Schedule Breakdown Screen

Team leaders (who may oversee small teams of employees) or individual engineers have use of a UI which is more relevant to their needs, as can be seen in Fig 8.4. This includes an overview of the employees own schedule, team schedule and shift types (such as late, or day shifts).

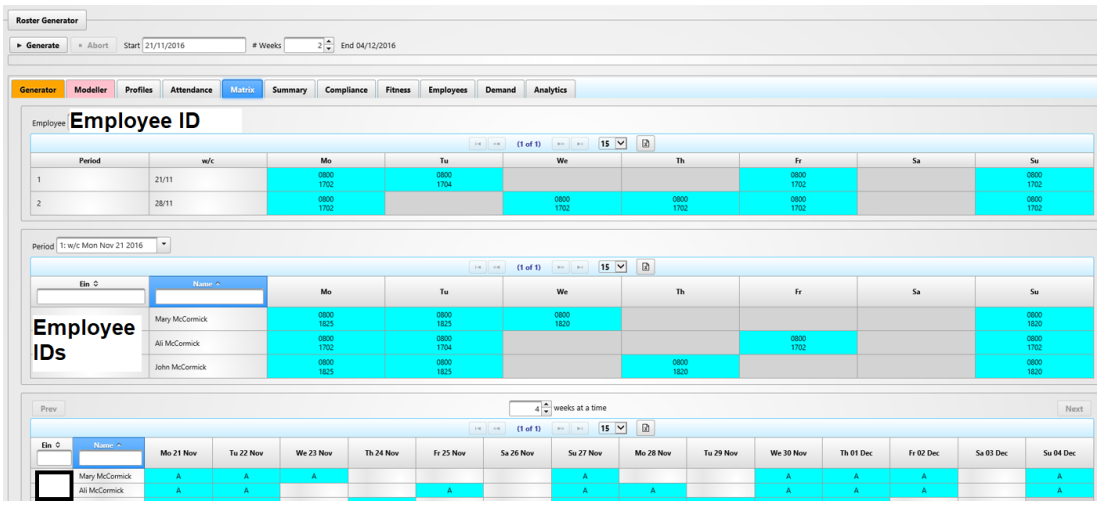


Figure 8.4: Employee View

As described in Chapter 2, there are two main problems tackled in this thesis, the personalised scheduling problem and the roster pattern scheduling problem. The personalised scheduling algorithms are not in active use for creating schedules at the FSO, as they are a proof-of-concept for future business opportunities. However, the solutions to the latter problem are applicable to real-world use. As of writing, the schedule generation tool has been utilised in test runs for creating rosters for thousands of employees, and discussions between employee unions and the FSO are taking place to allow further usage of this research in the real-world.

8.2 SIMULATOR

The research described in this thesis, which was implemented and integrated into the FSOs systems, has been used as an input for another research project.

This flow of data is shown visually in Fig 8.5 (iRoster contains the algorithms and engines described in this thesis).

To be clear — the simulator described in this section was created by researchers at the FSO, and is not an outcome of the work described in this thesis in any manner. The simulator is described here as it uses results from the algorithms arising from research described in this thesis as an input, and is being described with explicit permission from the FSO.

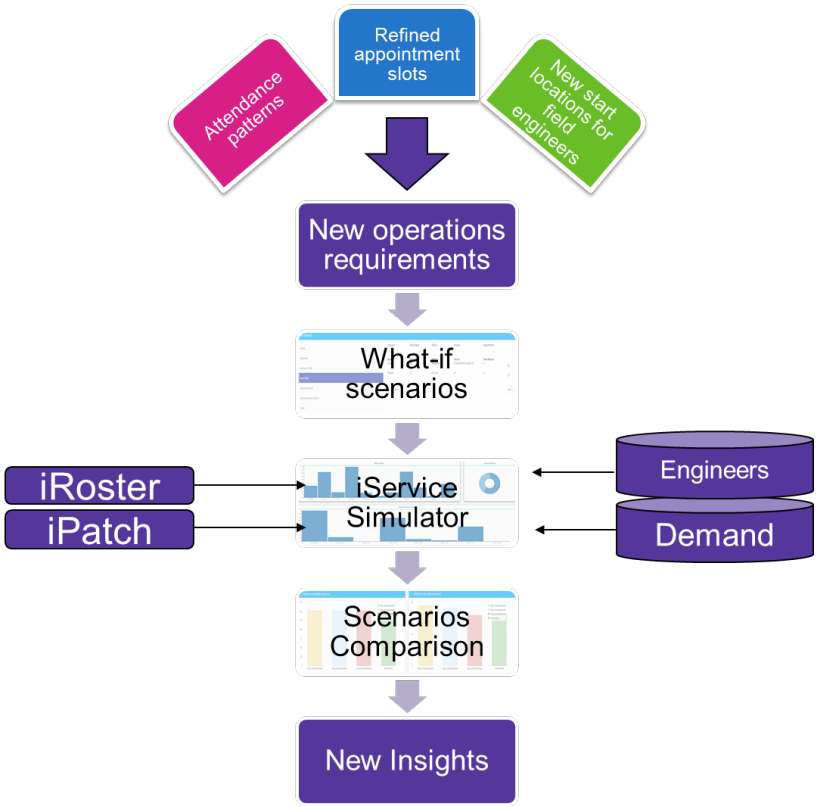


Figure 8.5: Cross-Project Data Flow

The premise of the simulator is to receive a one or more what-if scenarios. Such scenarios can include a variety of variables, such as new training budgets, new hiring budgets, redeployment of personnel to different geographical regions, modifications of team structure, new roster patterns, etc. In order to judge the potential benefits or disadvantages of these scenarios, the algorithms described in this thesis are provided with the tweaked data and run. This may

include a range of changes, for example a question such as "Would training 3 employees in this region, or hiring 1 new employee be more beneficial in regards to meeting demand?" could be asked, and would require multiple testing instances from the algorithms.

The potential benefits of testing scenarios may be improved demand satisfaction (which is beneficial for clients as well as the business), reduced carbon emissions by reducing travel time (or reducing travel requirement in general), reduced overtime requirement, improved employee satisfaction (by providing higher personnel coverage in certain regions, some employees may be awarded with less unfavourable shifts).

The scope of the work completed in this thesis schedules employees into shifts where they work a single skill throughout a shift, or per hour. Tasks are a non-abstracted view of this — if employees are working the shifts designated by the algorithms in this thesis, which specific tasks, in different locations, should be worked? iPatch, the tool utilised in the flow diagram in Fig 8.5, calculates a *task* focused solution.

The simulator can use the output from both the algorithms in this thesis in conjunction with output from iPatch to answer the what-if scenarios. An example of the analytics available after running the simulator can be viewed in Fig 8.6.

Figure 8.6: Simulation Analytics

Additional information regarding travel statistics and task satisfaction can be seen in Fig 8.7.

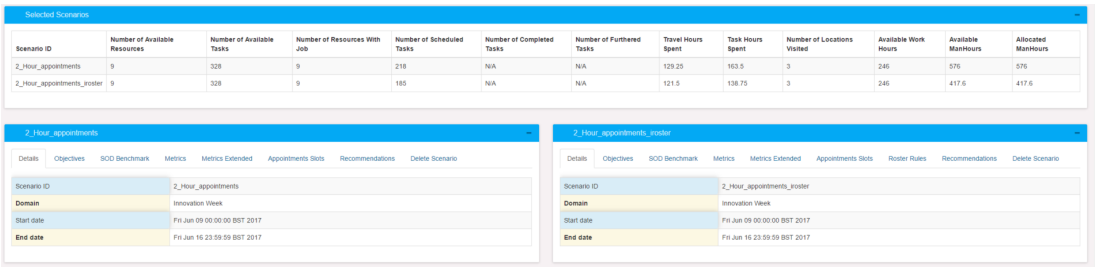


Figure 8.7: Additional Analytics for Simulator

Finally, a geographic visual can be viewed in Fig 8.8. This allows the user to see travel routes from task to task for individual engineers, which can be useful for visually verifying task allocations.

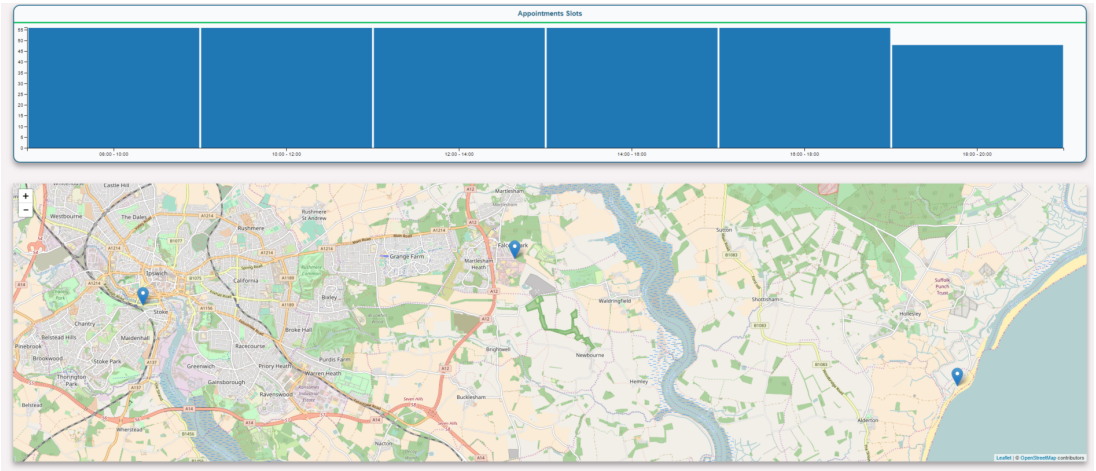


Figure 8.8: Geographic Visual Screenshot

9

CHAPTER 9 - CONCLUSIONS

This thesis has provided solutions to the employee scheduling problem, specifically in two sub-problems: the personalised rostering problem, and the roster pattern problem. An outline of the contributions made by this work follows, before evaluating the overall hypothesis, making suggestions for future work, and finally drawing the thesis to a close with general conclusions.

9.1 THESIS CONTRIBUTIONS

9.1.1 *Contribution 1: Problem Formulation & Improved Schedules over Baseline*

The problem provided by the FSO has been mathematically defined and formulated into clearly defined structures. The algorithms described in this thesis provide solutions of improved quality upon the baseline provided by the FSO, as well as providing better quality solutions than other tested metaheuristics. The algorithms were tested for two scenarios, one being a test of capability with an extensive period of time, and in a shorter time frame (e.g. 5 seconds, 1 minute, 2 minutes, 5 minutes), and were found to produce improved results in both scenarios.

9.1.2 *Contribution 2: Re-purposing of Evolutionary Ruin & Stochastic Recreate*

ER&SR has been successfully re-purposed for the employee scheduling problem, which has a much larger and more convoluted search space than the research undertaken by Li et al [15][16], where the exam scheduling problem is tackled with a much smaller number of variables and constraints. Thanks to the work produced by Li et al in producing a theoretic framework and general baseline to work from, this algorithm has been found to be novel in the field of shift scheduling and employee rostering.

9.1.3 *Contribution 3: Hybrid Algorithms*

Hybrid algorithms have been tested in comparison to their base components in this thesis, and found to provide higher quality solutions than their component algorithmic parts. This is true for both the ER&SR & VNS hybrid algorithm, and the ER&SR & IP hybrid. This aligns with other findings in the literature that show improved results for hybrid methods for real-world problems.

9.1.4 *Contribution 4: Constraints and Problem Formulation to Minimise Employee Dissatisfaction*

In discussions with the FSO and employee unions representing the engineers whose schedules are to be generated, in part, by work in this thesis, a programmatic representation of 'fairness' has been explored. This required much consideration in Chapters 5 & 6, where personalised schedules are generated. For example, SC4 which gives higher fitness to schedules where employees have consecutive rest days — this consecutive rest days attribute is 'baked in' to RPs in RP employee rostering problems in Chapters 7 & 8. Thus, in both the

personalised scheduling problem and the RP scheduling problem, minimising employee dissatisfaction has been successfully formulated and implemented.

9.1.5 *Contribution 5: Literature Review*

The literature review in Chapter 3 is in itself a contribution to the field. The trend of hybrid metaheuristics & matheuristics increasing in popularity in real-world problems was highlighted. The analysis on these optimisation tools may aid in focusing efforts by researchers in future works. Additionally, there has been no previous literature review on this intersection of topics.

9.1.6 *Contribution 6: 'Keystone' Allocations*

Observations of the data throughout this thesis highlighted an interesting feature of the search space which may be exploitable. In both the personalised scheduling problem and the RP problem, it was found that select allocations of employees to RPs, and of employees to shifts, have a ripple effect on one or more constraint satisfiers. This author has dubbed this characteristic a 'keystone' allocation, in that these allocations may in themselves have an deceivingly average fitness level, but allow several other improved allocations to be selected. An example of this may be a relatively unskilled worker being assigned a Saturday shift; freeing up high skilled employees to work the high demand Monday - Friday.

9.2 EVALUATION OF HYPOTHESIS

Thesis Hypothesis *By using a variety of known metaheuristics and integer programming techniques, a real-world shift scheduling & employee rostering problem can*

be solved to feasibility and a higher level of quality than the baseline, as provided by the FSO company.

This thesis utilised VNS, ER&SR (including one hybrid version and one math-euristic version of ER&SR) to produce employee rosters. All the produced employee rosters are feasible, so long as the parameters and data input represent physically possible outcomes. Additionally, the baseline is improved upon in quality of schedule, as represented by fitness values, in all strategies produced. I conclude that this thesis has provided sufficient evidence which supports the aforementioned hypothesis.

9.3 FUTURE WORK

There is a variety of potential future work that could be derived from research delivered in this thesis.

1. ER&SR could be applied to benchmark problems in other application areas to evaluate against the state of the art in other fields. Additionally it would be interesting to see the particular strengths of ER&SR, for example if the ruinous nature of the algorithm makes it better suited for search spaces with less uniformly distributed local optima.

2. ER&SR hybridisation is tested twice in this thesis, once with ER&SR combined with VNS, and once with ER&SR combined with IP. More hybridisation attempts could be explored in both the rebuild phase, ruin phase, as well as running after the final iteration.

3. ER&SR has never been tested with a hyper-heuristic, which would provide insight into the suitability of ER&SR to solve other problems, by mining which scenarios that a hyper-heuristic picks ER&SR to solve problems.

4. ER&SR has never been considered outside the domain of scheduling (exam timetabling in [16], and employee scheduling in this thesis). Using ER&SR in other problem domains could provide insights into the effectiveness of this metaheuristic in other problems and search spaces.

5. ER&SR has never been evaluated against an exhaustively searched fitness landscape. Visualising the search ER&SR makes could provide insight into the best problem domains it is applicable for, as well as improved parameterisation and possibly even improvements to the algorithm directly.

6. There has been little investigation into how ER&SR performs when given an initial solution compared to a random solution, or a blank solution. While this is likely problem space dependent, it may be worthwhile investigating, as it may improve run-time (and again, problem dependent, may worsen run-time, or improve the quality of fitness).

7. Alternative methods for acceptance criterion could be considered. For example Great Deluge (see Fig 9.1).

8. Parameter tuning is considered in this thesis, both with `irace` in Chapters 6 & 7, and with the Taguchi method in Chapter 5. As is described in the literature review, parameter tuning is somewhat in its infancy, with many researchers estimating good parameter values for their algorithms through trial and error, and others using tools such as `irace`. In reality, parameter tuning is useful for improving the quality of solutions, but that minor improvement is often not worth the time taken to test. For example, consider a real-world routing algorithm, which repeatedly redirects a driver to specific tasks needing completed. Optimising the route to maximise tasks completed would improve profit, if that was the objective. However, if the driver had to wait several hours for an

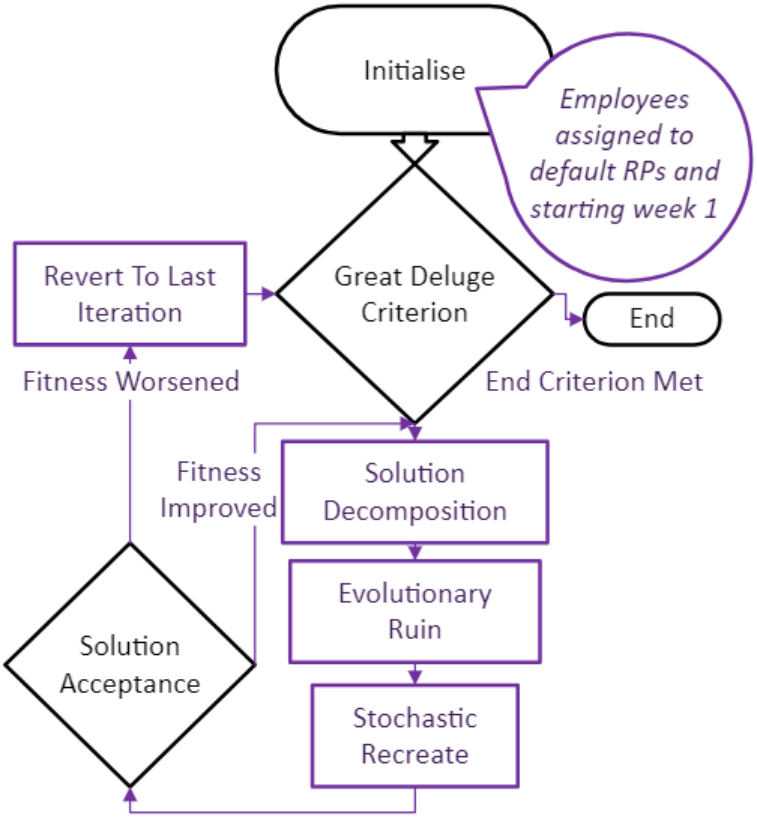


Figure 9.1: Alternate ER&SR Acceptance Criterion

ideal route to be calculated with ideal parameterisation, the solution would become pointless. In short, there are diminishing returns for parameterisation, depending on time constraints and problem domain. Studies into whether parameterisation should take place, and to what extent, in such constrained circumstances are warranted.

9. During the literature review (Chapter 3) writing for this thesis, it was noted that there is a lack of literature reviews in the past 4 years for real-world employee scheduling problems. A potential future work could be an extensive literature review in the field for these recent years, and discussing whether predictions made in previous reviews have come to fruition or were predicted incorrectly.

10. During testing of various algorithms throughout this thesis, it was noted that certain allocations of employees to RPs, or to specific shifts or hours, had a major effect on fitness - potentially by freeing other employees

to work more suited to their skill sets, or working RPs that are preferred rather than driven by demand. This 'keystone' characteristic of certain allocations is an interesting point of research, and understanding these specific turning points in search spaces could allow for further exploitation of the structure of the search space.

11. Software which uses the fitness functions described in this thesis to judge manual changes and assist in explaining why certain changes are less preferable (for example — 'changing employee A to RP 5 will cease allocations to Saturday shifts, causing less demand to be met throughout the schedule').

    - Furthering this — explainability is an interesting avenue for future research in employee scheduling. The technology exists to automatically schedule employees across most industries, yet there seems to be a hesitation in uptake of the AI-driven scheduling. This author hypothesises that this is largely due to a lack of explainability - fitness values are useful for operational researchers but not for non-AI specialist personnel managers. Explainability, in the form of highlighting keystone [1] allocations, visualising demand coverage, or describing cause and effect of specific decisions, might encourage more trust in automatic systems.

12. While there are many benchmarking instances of employee scheduling, none were found which have a similar structure to this problem. A future work could be in creating benchmark employee scheduling problems with RPs, preferred RP constraints and starting week rotations.

1 This phrase is described in suggestion #10

9.4 CONCLUDING REMARKS

In this thesis I have investigated the applicability of standalone metaheuristics and hybrid metaheuristics for a real-world industrial optimisation problem. In particular, the ER&SR algorithm has a firmer foundation in the literature than it did previously, now spanning two real-world problems, and now implemented in two hybrid forms. A real-world company now has multiple algorithms and software products to use in scheduling employees, and related problems (such as simulation, potential decision tree foundations, tailor made hyper-heuristic components, etc.) Throughout all content chapters of this thesis, baseline fitness has been improved upon, and several contributions to the literature now exist in the form of scientific papers. An overview of the literature in both recent years and specifically regarding real-world employee scheduling. Finally, a variety of potential future works have been described, providing a basis for further work by my colleagues in the field.

BIBLIOGRAPHY

[1] C. Heimerl and R. Kolisch, "Scheduling and staffing multiple projects with a multi-skilled workforce," *OR spectrum*, vol. 32, no. 2, pp. 343–368, 2010.

[2] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[3] M. Gendreau and J.-Y. Potvin, *Handbook of metaheuristics*. Springer, 2010, vol. 2.

[4] E. K. Burke, G. Kendall *et al.*, *Search methodologies*. Springer, 2005.

[5] E. K. Burke, J. Li, and R. Qu, "A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems," *European Journal of Operational Research*, vol. 203, no. 2, pp. 484–493, 2010.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://science.sciencemag.org/content/220/4598/671

[7] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[8] A. El-Yaakoubi, A. El-Fallahi, M. Cherkaoui, and M. R. Hamzaoui, "Tabu search and memetic algorithms for a real scheduling and routing problem." *Logistics Research*, vol. 10, no. 1, p. 7, 2017.

[9] P. Hansen, N. Mladenović, and J. A. M. Pérez, "Variable neighbourhood search: methods and applications," *4OR*, vol. 6, no. 4, pp. 319–360, 2008.

[10] P. Hansen, N. Mladenović, and J. A. M. Pérez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.

[11] C. Binhui, Q. Rong, B. Ruibin, and I. Hisao, "A variable neighbourhood search algorithm with compound neighbourhoods for vrptw," in *The 2016 International Conference on Operations Research and Enterprise Systems (ICORES'16)*, 2016, pp. 23–25.

[12] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[13] J. H. Drake, N. Kililis, and E. Özcan, "Generation of vns components with grammatical evolution for vehicle routing," in *European Conference on Genetic Programming*. Springer, 2013, pp. 25–36.

[14] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & operations research*, vol. 34, no. 8, pp. 2403–2435, 2007.

[15] J. Li, R. Qu, and Y. Shen, "Evolutionary ruin and stochastic recreate: A case study on the exam timetabling problem." in *ECMS*, 2012, pp. 347–353.

[16] J. Li, R. Bai, Y. Shen, and R. Qu, "Search with evolutionary ruin and stochastic rebuild: A theoretic framework and a case study on exam timetabling," *European Journal of Operational Research*, vol. 242, no. 3, pp. 798–806, 2015.

[17] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[18] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[19] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, 1970.

[20] D. Teodorović and G. Pavković, "A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand," *Transportation Planning and Technology*, vol. 16, no. 4, pp. 261–273, 1992.

[21] A. H. Chown, C. J. Cook, and N. B. Wilding, "A simulated annealing approach to the student-project allocation problem," *American Journal of Physics*, vol. 86, no. 9, pp. 701–708, 2018.

[22] B. K. Saha, S. Misra, and S. Pal, "Seer: Simulated annealing-based routing in opportunistic mobile networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, pp. 2876–2888, 2017.

[23] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of operations research*, vol. 41, no. 4, pp. 421–451, 1993.

[24] A. R. Yıldız, "Hybrid taguchi-harmony search algorithm for solving engineering optimization problems," *International Journal of Industrial Engineering*, vol. 15, no. 3, pp. 286–293, 2008.

[25] Z.-J. Lee and W.-L. Lee, "A hybrid search algorithm of ant colony optimization and genetic algorithm applied to weapon-target assignment

problems," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2003, pp. 278–285.

[26] K. F. Doerner and V. Schmid, "Survey: matheuristics for rich vehicle routing problems," in *International Workshop on Hybrid Metaheuristics*. Springer, 2010, pp. 206–221.

[27] C. Archetti and M. G. Speranza, "A survey on matheuristics for routing problems," *EURO Journal on Computational Optimization*, vol. 2, no. 4, pp. 223–246, 2014.

[28] L. Fanjul-Peyro, F. Perea, and R. Ruiz, "Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources," *European Journal of Operational Research*, vol. 260, no. 2, pp. 482–493, 2017.

[29] J.-F. Cordeau, G. Laporte, F. Pasin, and S. Ropke, "Scheduling technicians and tasks in a telecommunications company," *Journal of Scheduling*, vol. 13, no. 4, pp. 393–409, 2010.

[30] M. Stojković, F. Soumis, and J. Desrosiers, "The operational airline crew scheduling problem," *Transportation Science*, vol. 32, no. 3, pp. 232–245, 1998.

[31] B. Feiring, "A model generation approach to the personnel assignment problem," *Journal of the Operational Research Society*, vol. 44, no. 5, pp. 503–512, 1993.

[32] G. B. Crockett and P. H. Leamon, "Skills-based scheduling for telephone call centers," Mar. 28 2000, uS Patent 6,044,355.

[33] A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *European journal of operational research*, vol. 153, no. 1, pp. 3–27, 2004.

[34] K. R. Baker, "Workforce allocation in cyclical scheduling problems: A survey," *Journal of the Operational Research Society*, vol. 27, no. 1, pp. 155–167, 1976.

[35] U. Benlic, E. K. Burke, and J. R. Woodward, "Breakout local search for the multi-objective gate allocation problem," *Computers & Operations Research*, vol. 78, pp. 80–93, 2017.

[36] E. K. Burke, P. De Causmaecker, G. De Maere, J. Mulder, M. Paelinck, and G. V. Berghe, "A multi-objective approach for robust airline scheduling," *Computers & Operations Research*, vol. 37, no. 5, pp. 822–832, 2010.

[37] E. Hart, P. Ross, and D. Corne, "Evolutionary scheduling: A review," *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 191–220, 2005.

[38] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck, "Personnel scheduling: A literature review," *European Journal of Operational Research*, vol. 226, no. 3, pp. 367–385, 2013.

[39] J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu, "Workforce scheduling and routing problems: literature survey and computational study," *Annals of Operations Research*, vol. 239, no. 1, pp. 39–67, 2016.

[40] C. Fikar and P. Hirsch, "Home health care routing and scheduling: A review," *Computers & Operations Research*, vol. 77, pp. 86–95, 2017.

[41] M. Erhard, J. Schoenfelder, A. Fügener, and J. O. Brunner, "State of the art in physician scheduling," *European Journal of Operational Research*, vol. 265, no. 1, pp. 1–18, 2018.

[42] M. Pinedo, C. Zacharias, and N. Zhu, "Scheduling in the service industries: An overview," *Journal of Systems Science and Systems Engineering*, vol. 24, no. 1, pp. 1–48, 2015.

[43] R. L. Pinheiro, D. Landa-Silva, and J. Atkin, "A variable neighbourhood search for the workforce scheduling and routing problem," in *Advances in Nature and Biologically Inspired Computing*. Springer, 2016, pp. 247–259.

[44] I. P. Solos, I. X. Tassopoulos, and G. N. Beligiannis, "Optimizing shift scheduling for tank trucks using an effective stochastic variable neighbourhood approach," *Int. J. Artif. Intell*, vol. 14, no. 1, pp. 1–26, 2016.

[45] B. Maenhout and M. Vanhoucke, "An evolutionary approach for the nurse rerostering problem," *Computers & Operations Research*, vol. 38, no. 10, pp. 1400–1411, 2011.

[46] V. Pillac, C. Gueret, and A. L. Medaglia, "A parallel matheuristic for the technician routing and scheduling problem," *Optimization Letters*, vol. 7, no. 7, pp. 1525–1535, 2013.

[47] V. Pillac, C. Guéret, and A. Medaglia, "A fast reoptimization approach for the dynamic technician routing and scheduling problem," in *Recent Developments in Metaheuristics*. Springer, 2018, pp. 347–367.

[48] M. E. Bucci, J. A. Velazquez, and P. J. Piccolomini, "Dynamic workforce scheduler," Nov. 23 2004, uS Patent 6,823,315.

[49] M. J. Brusco and L. W. Jacobs, "A simulated annealing approach to the solution of flexible labour scheduling problems," *Journal of the Operational Research Society*, vol. 44, no. 12, pp. 1191–1200, 1993.

[50] P. Brucker, R. Qu, and E. Burke, "Personnel scheduling: Models and complexity," *European Journal of Operational Research*, vol. 210, no. 3, pp. 467–473, 2011.

[51] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Applied Soft Computing*, vol. 11, no. 6, pp. 4135–4151, 2011.

[52] S. M. Pour, J. H. Drake, L. S. Ejlertsen, K. M. Rasmussen, and E. K. Burke, "A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem," *European Journal of Operational Research*, vol. 269, no. 1, pp. 341–352, 2018.

[53] E. Rahimian, K. Akartunalı, and J. Levine, "A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems," *European Journal of Operational Research*, vol. 258, no. 2, pp. 411–423, 2017.

[54] G. R. Raidl, J. Puchinger, and C. Blum, "Metaheuristic hybrids," in *Handbook of Metaheuristics*. Springer, 2019, pp. 385–417.

[55] E. K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman, "A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem," *European Journal of Operational Research*, vol. 188, no. 2, pp. 330–341, 2008.

[56] F. Della Croce and F. Salassa, "A variable neighborhood search based matheuristic for nurse rostering problems," *Annals of Operations Research*, vol. 218, no. 1, pp. 185–199, 2014.

[57] X. Chen, B. W. Thomas, and M. Hewitt, "Multi-period technician scheduling with experience-based service times and stochastic customers," *Computers & Operations Research*, vol. 82, pp. 1–14, 2017.

[58] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[59] P. J. Ross and P. J. Ross, *Taguchi techniques for quality engineering: loss function, orthogonal experiments, parameter and tolerance design*. McGraw-Hill New York, 1988, no. TS156 R12.

[60] H. Wang, Q. Geng, and Z. Qiao, "Parameter tuning of particle swarm optimization by using taguchi method and its application to motor design," in *2014 4th IEEE International Conference on Information Science and Technology*. IEEE, 2014, pp. 722–726.

[61] A. Maleki-Daronkolaei and I. Seyedi, "Taguchi method for three-stage assembly flow shop scheduling problem with blocking and sequence-dependent set up times," *Journal of Engineering Science and Technology*, vol. 8, no. 5, pp. 603–622, 2013.

[62] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[63] P. Balaprakash, M. Birattari, and T. Stützle, "Improvement strategies for the f-race algorithm: Sampling design and iterative refinement," in *International workshop on hybrid metaheuristics*. Springer, 2007, pp. 108–122.

[64] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated f-race: An overview," in *Experimental methods for the analysis of optimization algorithms*. Springer, 2010, pp. 311–336.

[65] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools," *Computers & operations research*, vol. 51, pp. 190–199, 2014.

[66] J. R. Woodward and J. Swan, "The automatic generation of mutation operators for genetic algorithms," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. ACM, 2012, pp. 67–74.

[67] S. O. Haraldsson and J. R. Woodward, "Automated design of algorithms and genetic improvement: contrast and commonalities," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2014, pp. 1373–1380.

[68] A. Yarimcam, S. Asta, E. Özcan, and A. J. Parkes, "Heuristic generation via parameter tuning for online bin packing," in *2014 IEEE symposium on evolving and autonomous learning systems (EALS)*. IEEE, 2014, pp. 102–108.

[69] N. Rangel-Valdez and J. Torres-Jimenez, "Solving employee timetabling in a call center of a telecommunications company in mexico with simulated annealing," in *2009 Eighth Mexican International Conference on Artificial Intelligence*. IEEE, 2009, pp. 170–175.

[70] A. Petrovski, A. Brownlee, and J. McCall, "Statistical optimisation and tuning of ga factors," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1. IEEE, 2005, pp. 758–764.

[71] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[72] L. P. Cáceres, B. Bischl, and T. Stützle, "Evaluating random forest models for irace," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 1146–1153.

[73] M. Buddenhagen and Y. Lierler, "Performance tuning in answer set programming," in *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 2015, pp. 186–198.

[74] R. Burger, M. Bharatheesha, M. van Eert, and R. Babuška, "Automated tuning and configuration of path planning algorithms," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4371–4376.

[75] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[76] D. H. Wolpert, W. G. Macready *et al.*, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[77] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," *IEEE Transactions on Evolutionary Computation*, 2019.

[78] G. Taguchi and G. Taguchi, "System of experimental design; engineering methods to optimize quality and minimize costs," UNIPUB/Kraus Internation, Tech. Rep., 1987.

[79] S. Mahapatra and A. Patnaik, "Optimization of wire electrical discharge machining (wedm) process parameters using taguchi method," *The International Journal of Advanced Manufacturing Technology*, vol. 34, no. 9, pp. 911–925, 2007.

[80] R. Tyasnurita, E. Özcan, and R. John, "Learning heuristic selection using a time delay neural network for open vehicle routing," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 1474–1481.

[81] R. N. Kacker, E. S. Lagergren, and J. J. Filliben, "Taguchi's orthogonal arrays are classical designs of experiments," *Journal of research of the National Institute of Standards and Technology*, vol. 96, no. 5, p. 577, 1991.

[82] L. Simeonova, N. Wassan, S. Salhi, and G. Nagy, "The heterogeneous fleet vehicle routing problem with light loads and overtime: Formulation and population variable neighbourhood search with adaptive memory," *Expert Systems with Applications*, vol. 114, pp. 183–195, 2018.

[83] H. H. Hoos, "Programming by optimization." *Commun. ACM*, vol. 55, no. 2, pp. 70–80, 2012.

[84] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

[85] P. De Bruecker, J. Beliën, J. Van den Bergh, and E. Demeulemeester, "A three-stage mixed integer programming approach for optimizing the skill mix and training schedules for aircraft maintenance," *European Journal of Operational Research*, vol. 267, no. 2, pp. 439–452, 2018.

[86] K. Kim and S. Mehrotra, "A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management," *Operations Research*, vol. 63, no. 6, pp. 1431–1451, 2015.

[87] K. N. Reid, J. Li, J. Swan, A. McCormick, and G. Owusu, "Variable neighbourhood search: A case study for a highly-constrained workforce scheduling problem," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–6.

[88] J. Ingels and B. Maenhout, "A memetic algorithm to maximise the employee substitutability in personnel shift scheduling," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2017, pp. 44–59.

[89] M. Comer, "Sickness absence in the uk labour market - office for national statistics," 2017. [Online]. Available: https:

//www.ons.gov.uk/employmentandlabourmarket/peopleinwork/labourproductivity/articles/sicknessabsenceinthelabourmarket/2016

[90] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The irace package, iterated race for automatic algorithm configuration," IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004, 2011. [Online]. Available: http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf

APPENDIX A

A.1 OTHER WORKS

A.1.1 *Introduction*

While developing initial solutions for the FSO, several interim solutions were used as proof of concepts. Of these, two are worth mentioning: the "insider knowledge engine" and the "iterative pattern approach". These mechanisms are heuristics, in that they do not guarantee optimal solutions. They are useful mechanisms for testing parameterisation of algorithms before completion of implementation; for comparing the algorithms to quick solutions which do little for fitness improvement; and useful for testing expected outputs of algorithms before they are completed. In short, they are useful for integration and prototyping. These two solutions are described in this appendix.

A.1.2 *Insider Knowledge Engine*

Shifts do not necessarily have to meet any demand in order to exist. Employees can work shifts where there is no demand at all, which is important in order to meet HCs. In this solution, it is fairly likely to have a number of allocations which rank poorly when considering SCs. This is acceptable, so long as the solution is feasible, and the Insider Knowledge Engine (IKE) ensures that every solution is feasible by design.

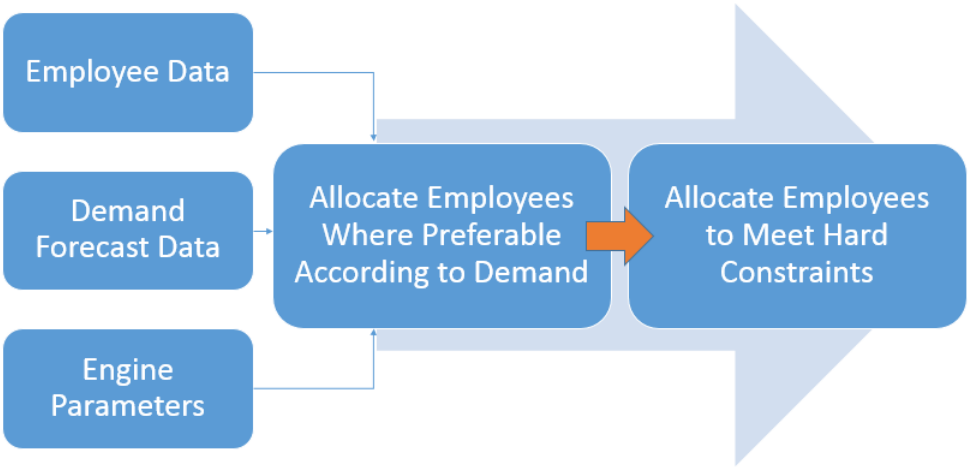


Figure A.1: IKA Diagram

This solution works by iterating through the list of all employees, assigning them shifts which meet all contractual and legal requirements, with a preference for shifts with higher demand. If no such shift exists, then any feasible shift is chosen. No other SCs are considered (in this instance, see Chapter 2 for the relevant constraints). This means there is a great likelihood that employees will not have consecutive rest days, or work preferred skills. For a visual representation of IKE see Fig A.1.

A.1.3 *Iterative Pattern Approach*

Similar to subsection A.1.2, this approach does not consider SCs, and always produces feasible solutions (addressing all HCs). The constraints for this engine are the same as in Chapter 2. This engine is an improvement upon the IKE.

The Iterative Pattern Approach (IPA) approach works by creating a set of personalised rosters for employees, then using HC compliant patterns iteratively for multiple employees. The hypothesis was that by reusing already functional patterns, run-time could be reduced. Fitness was not considered, however more constraints were hard-coded, for example patterns which had consecutive rest days were preferred over those without. This means the solu-

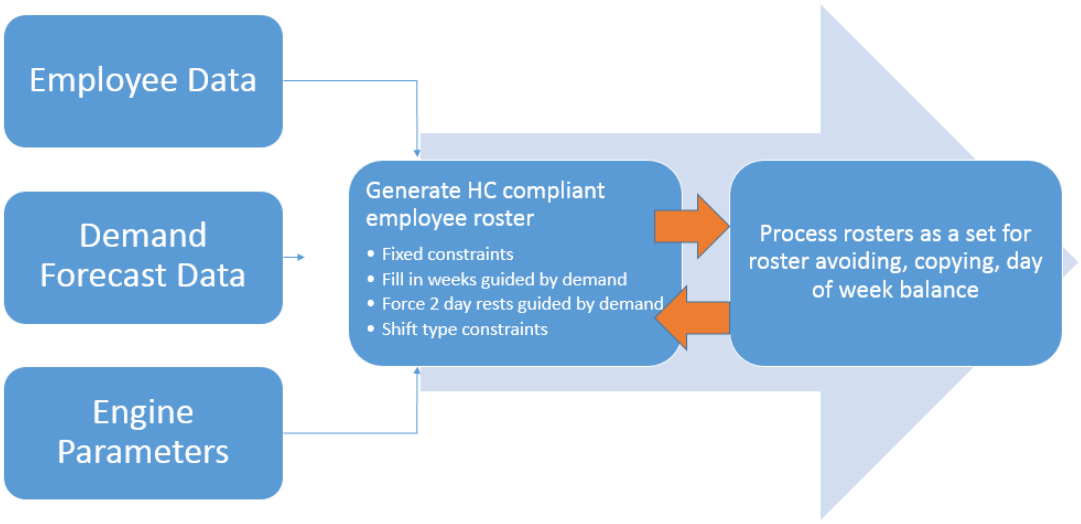


Figure A.2: IPA Diagram

tions produced were more pleasing to the specifications, while no numerical values were presented to prove performance (fitness, cost, etc.). This approach has been visually represented in Fig A.2.

A.1.4 *Conclusion*

These approaches were useful for testing various approaches which were not evolutionarily driven, and useful for testing integration with FSO systems. Further, as these engines were first created early in my PhD, they proved useful in developing my skillset in software development and becoming familiar with the FSO's specific problems.

B

APPENDIX B

List of Figures

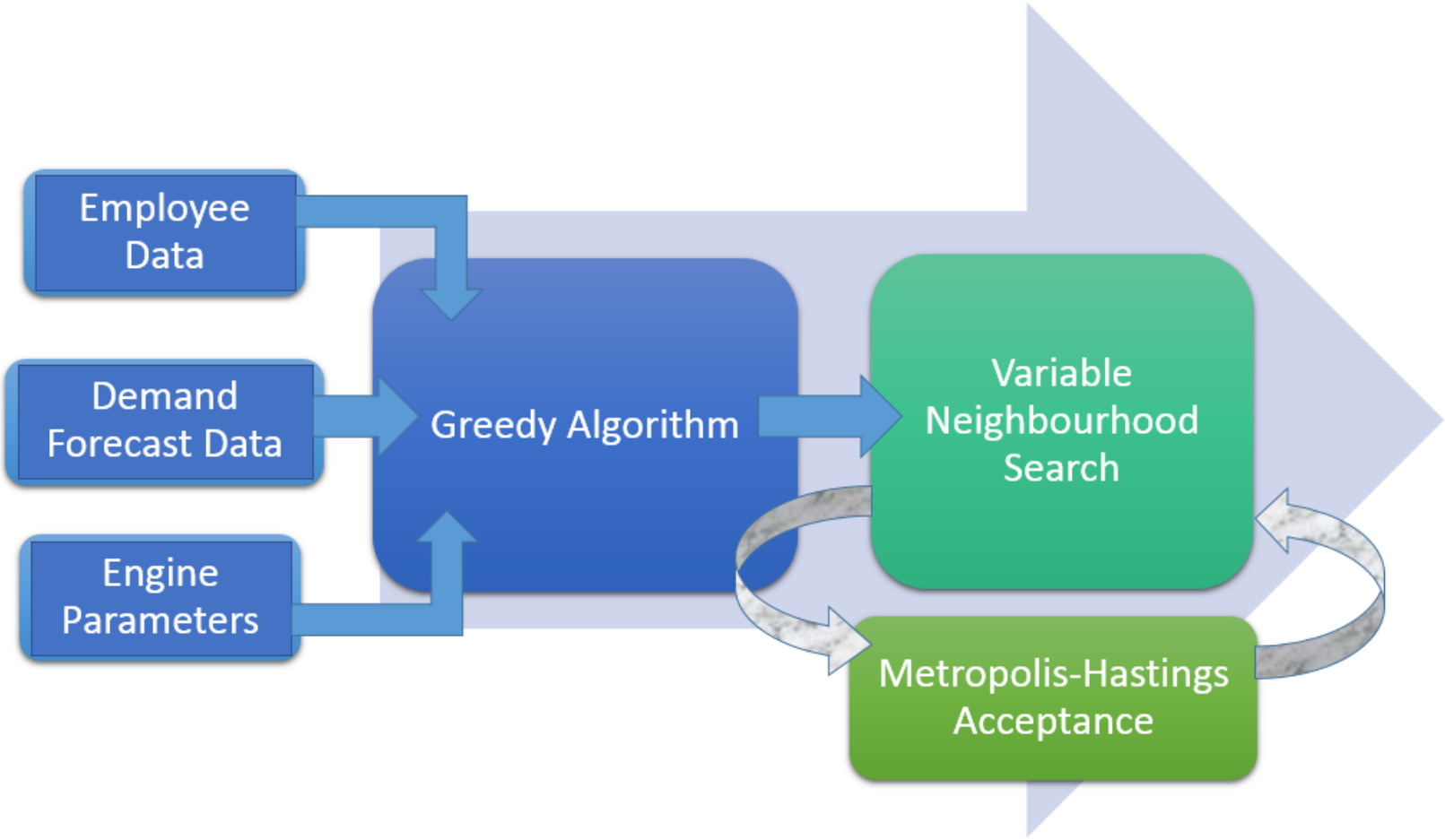


Figure 1: Chapter 4 - Diagrammatic Overview of Algorithm Flow

E	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
e1	D	D	D	L	D		
e2	L	D		D	D	D	

0.789

E	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
e1	D	D	D	L	D		
e2	L	D		D	D	D	

E	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
e1	L	D	D	L	D		
e2	D	D		D	D	D	

0.79

Figure 2: Chapter 4 - Intra-day Swap Mechanism

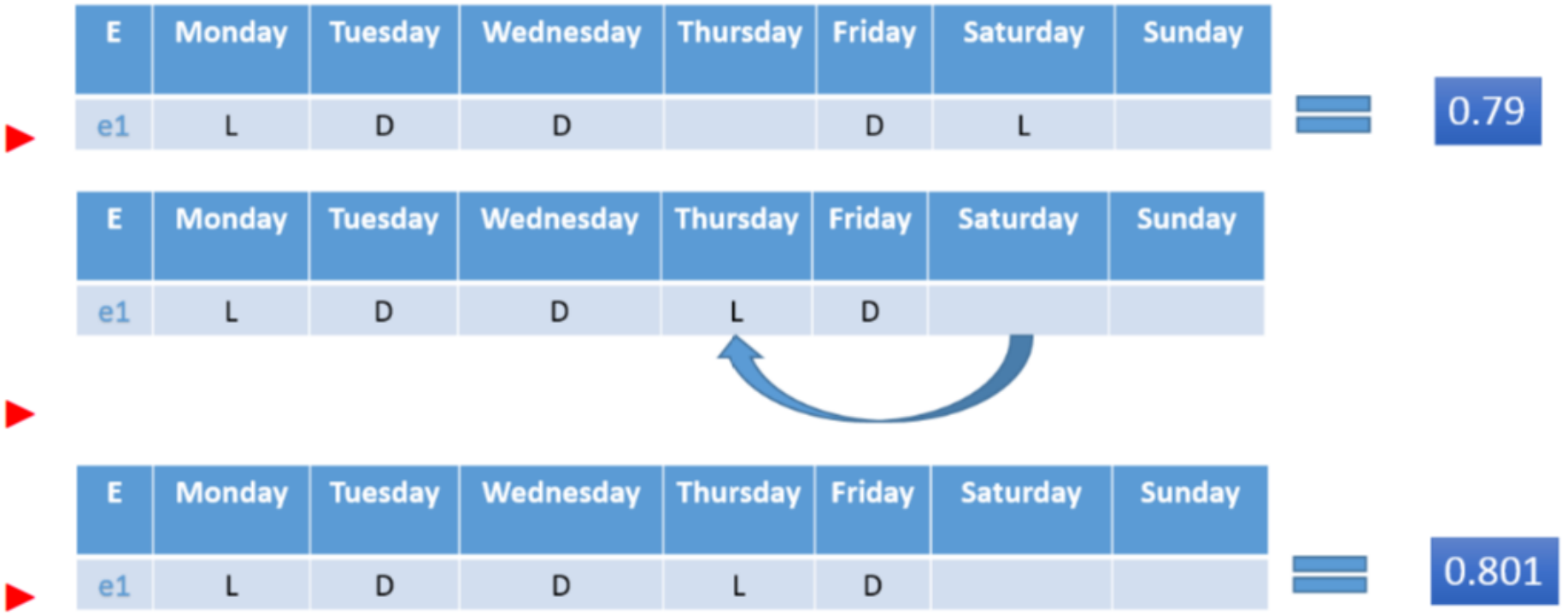


Figure 3: Chapter 4 - Inter-day Swap Mechanism

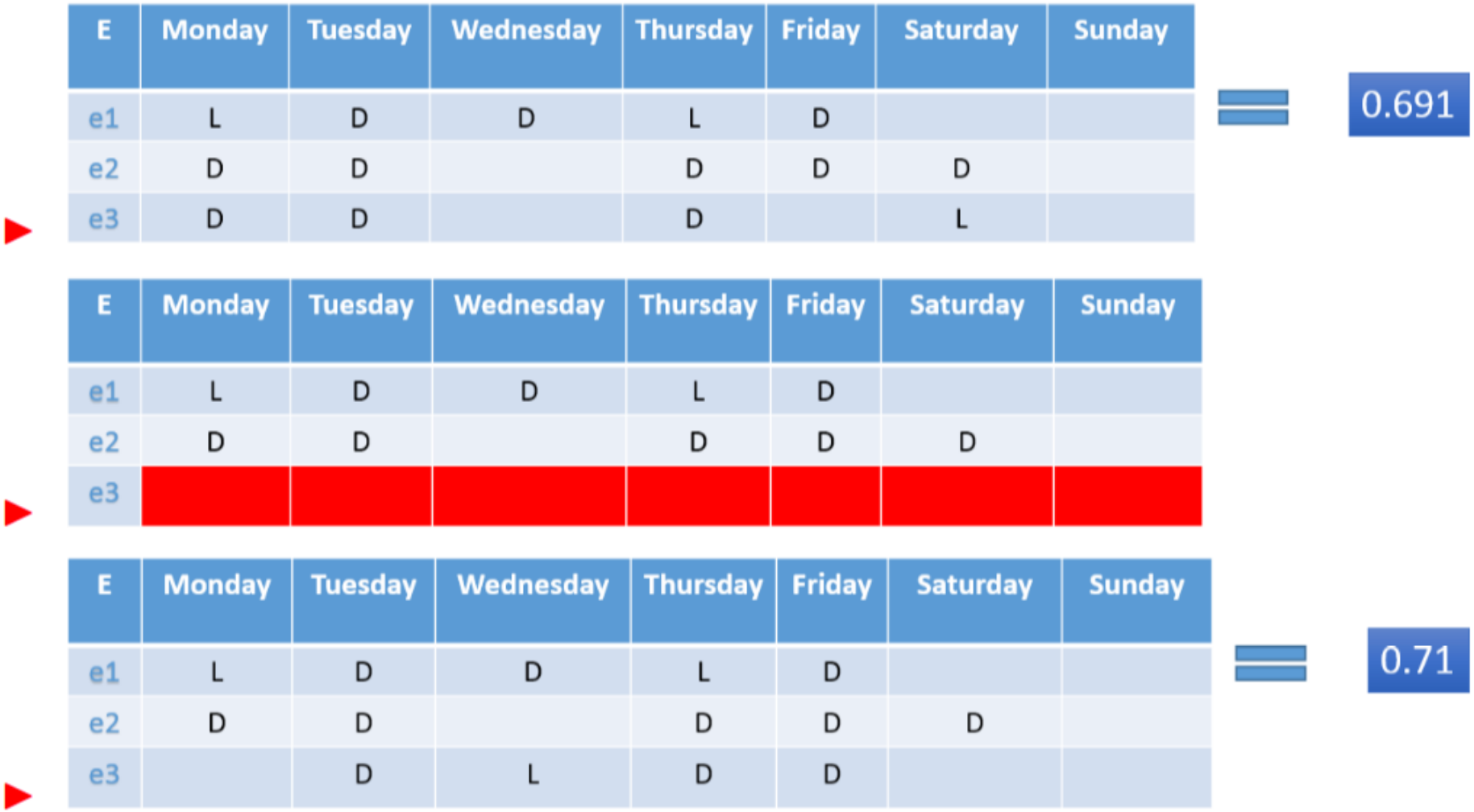


Figure 4: Chapter 4 - Delete & Regenerate Mechanism

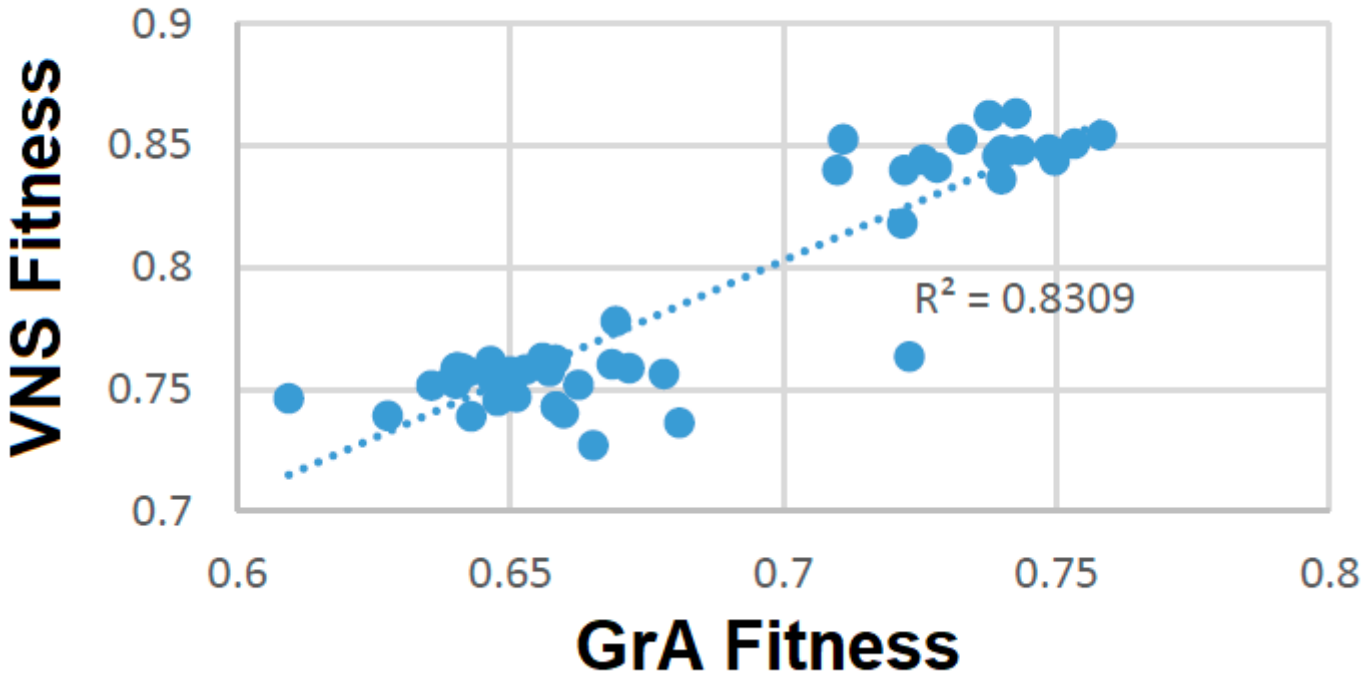


Figure 5: Chapter 4 - Mean VNS Scatter Plot

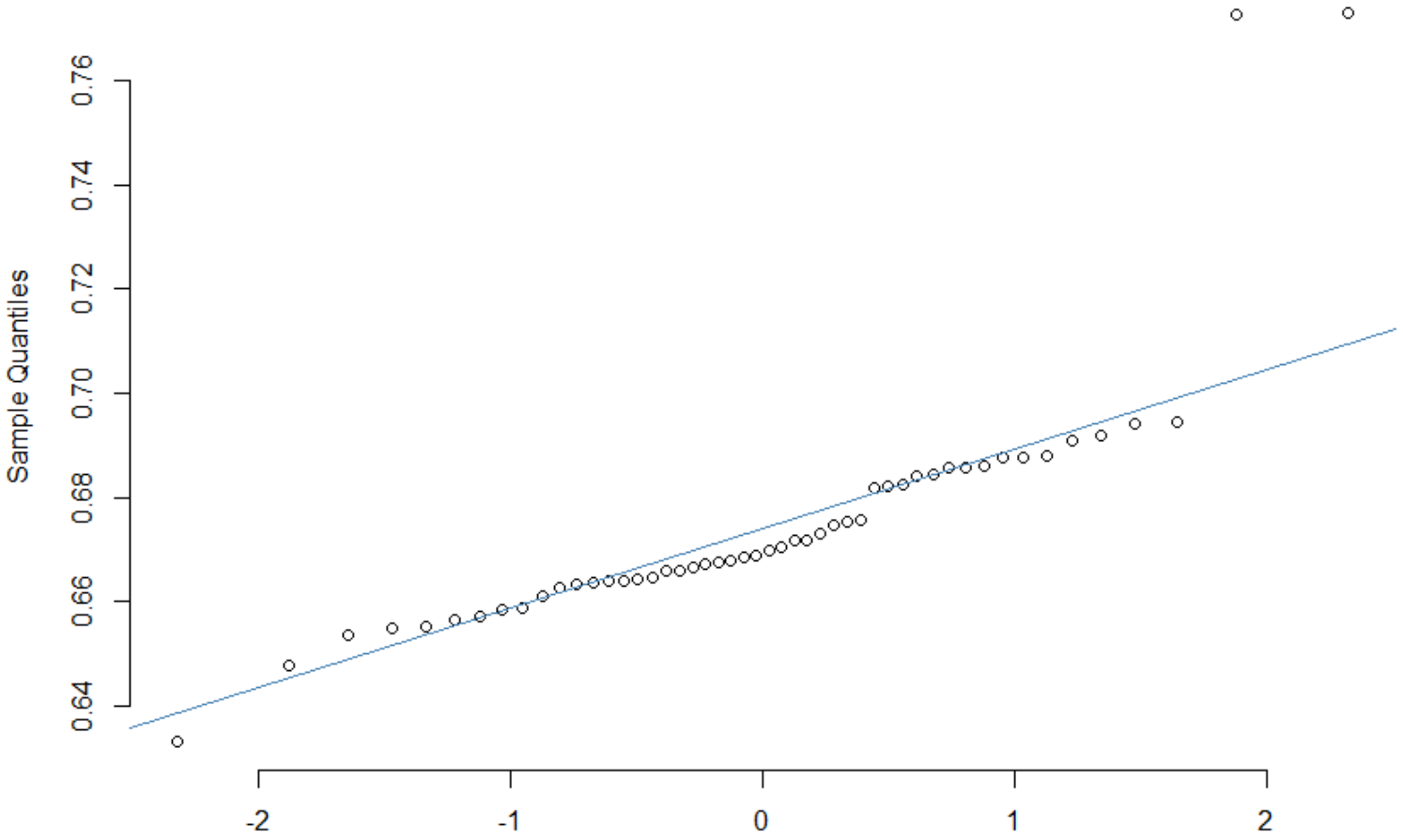


Figure 6: Chapter 4 - Greedy Algorithm Q-Q Plot

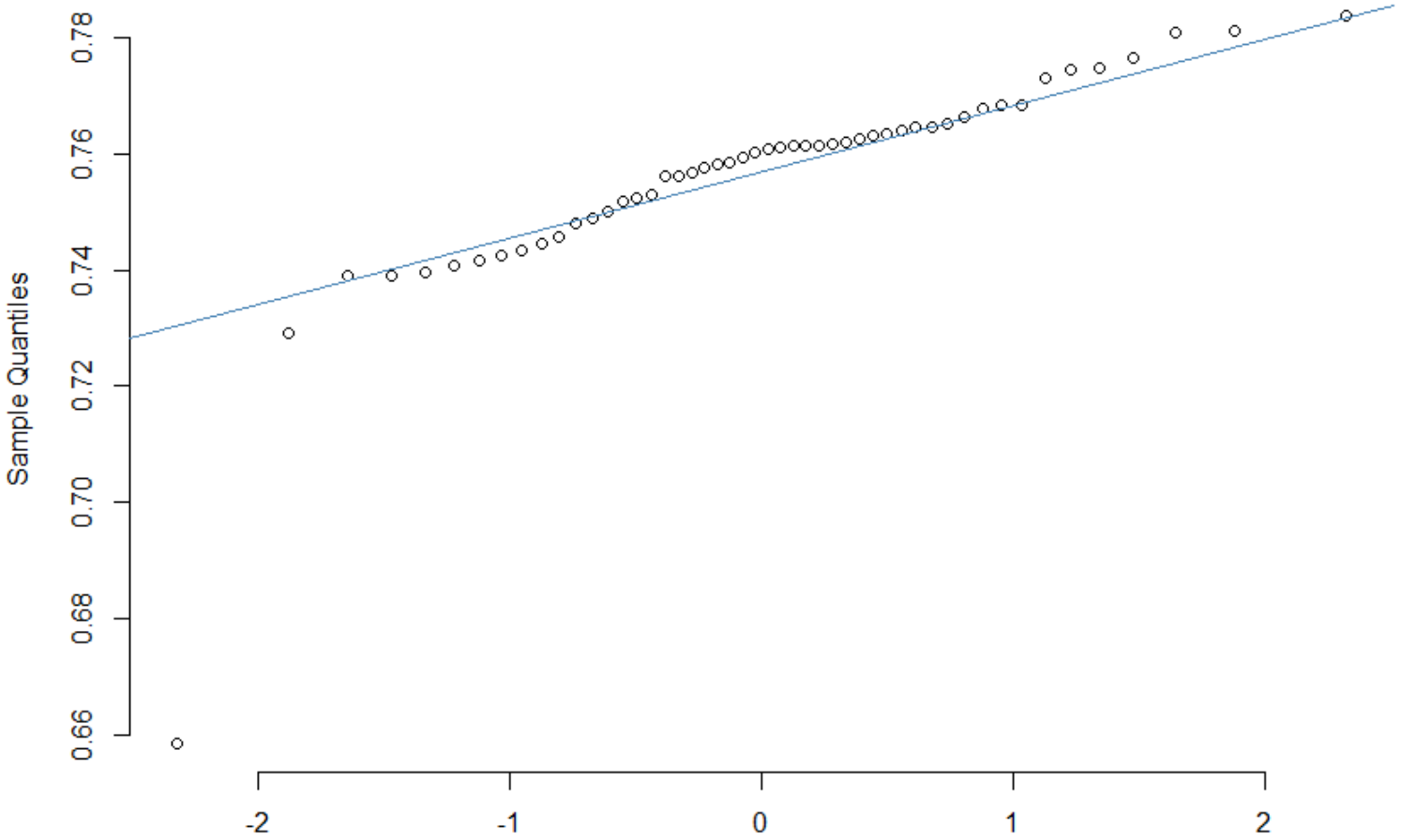


Figure 7: Chapter 4 - VNS Q-Q Plot

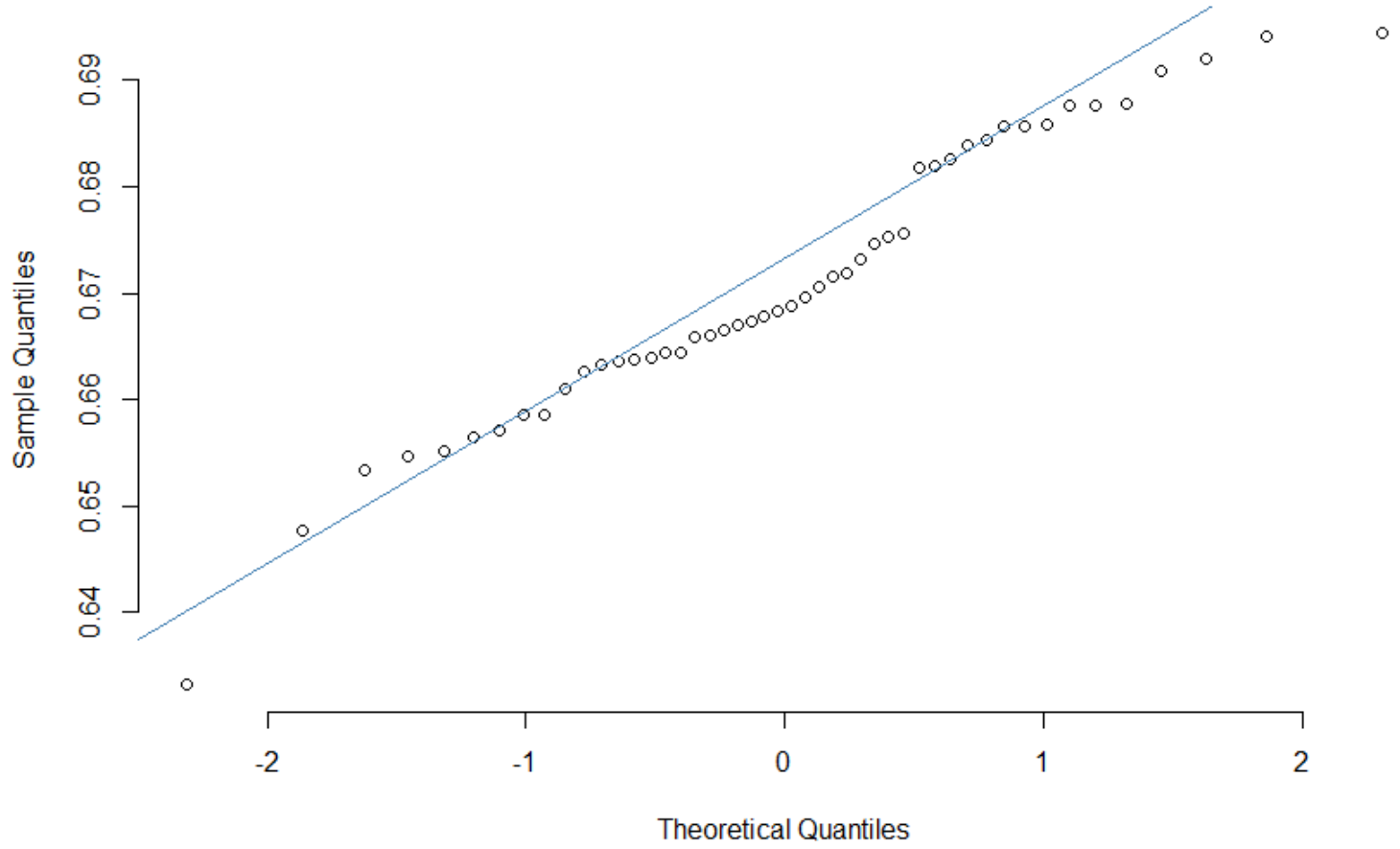


Figure 8: Chapter 4 - Greedy Algorithm Q-Q Plot with Outliers Removed

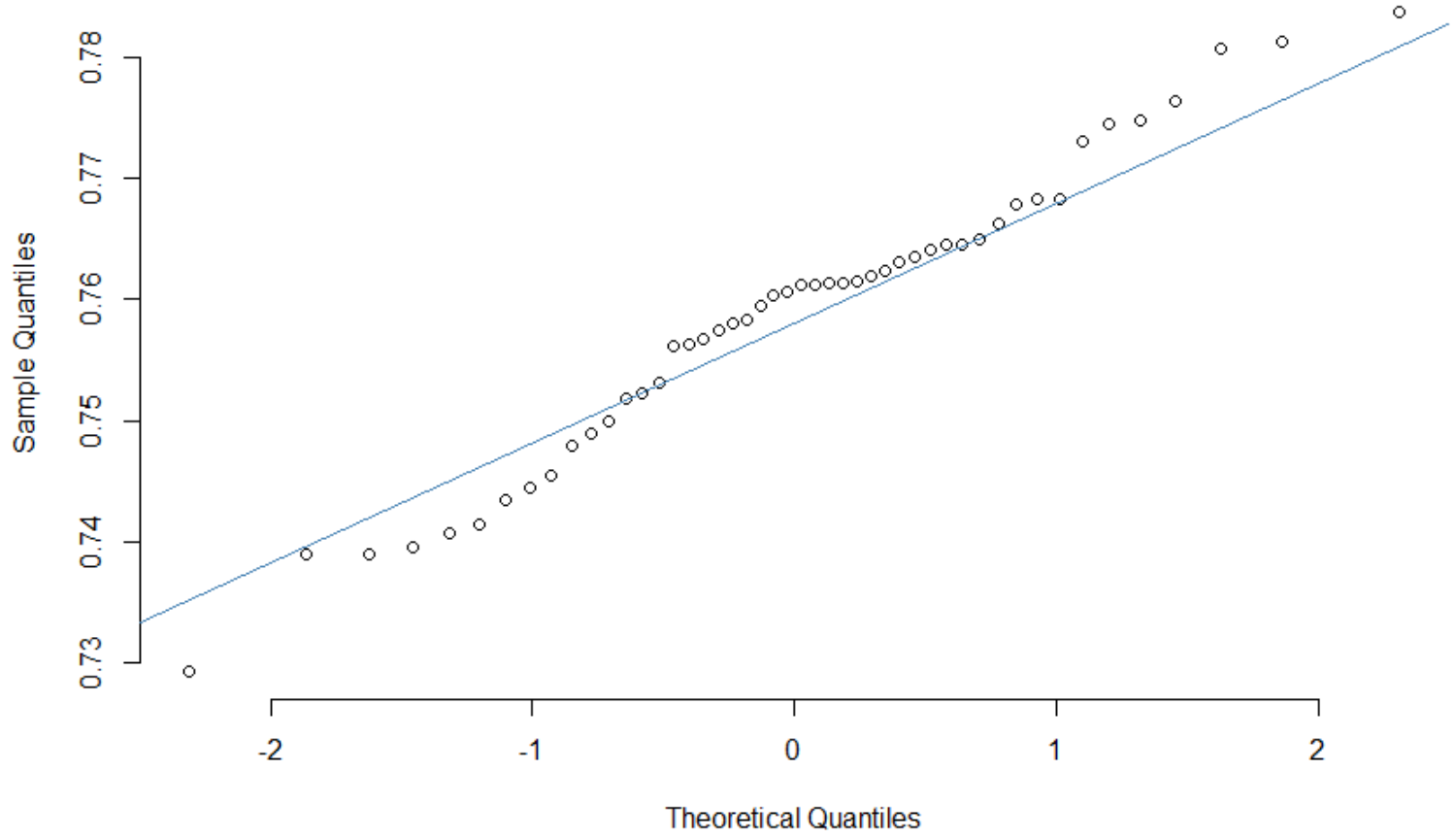


Figure 9: Chapter 4 - VNS Q-Q Plot with Outliers Removed

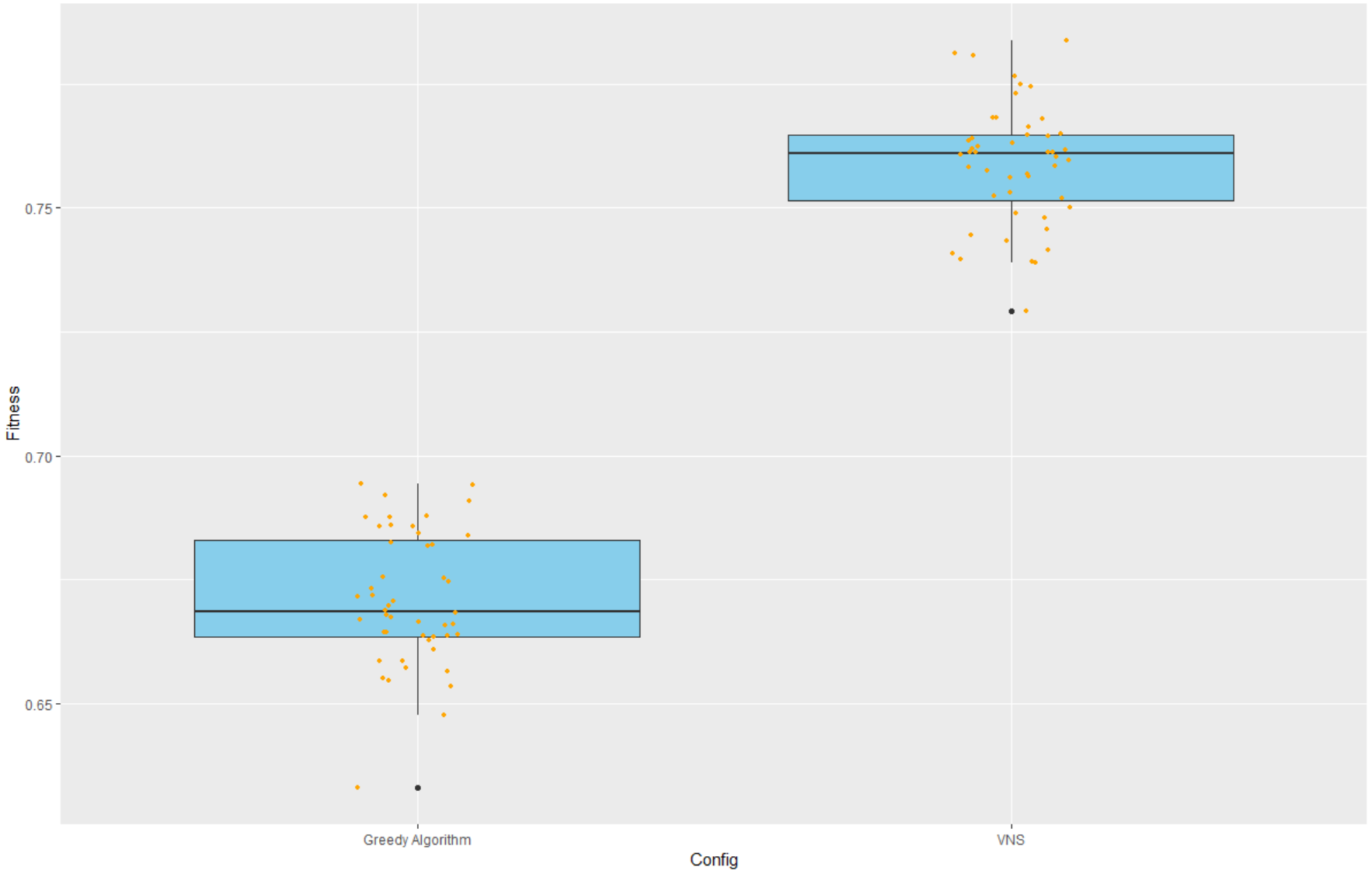


Figure 10: Chapter 4 - Greedy Algorithm vs VNS Boxplot

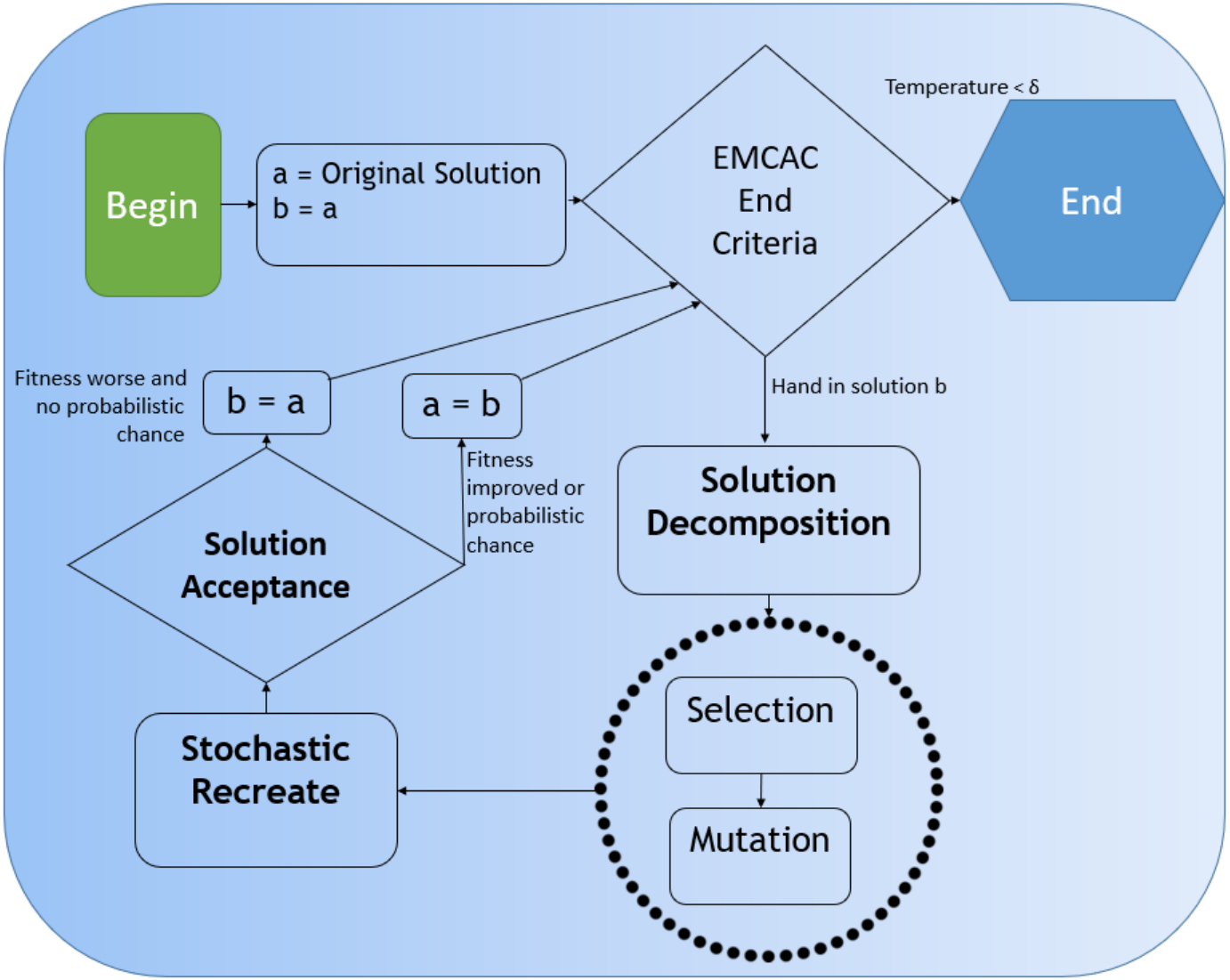


Figure 11: Chapter 5 - Flow Diagram

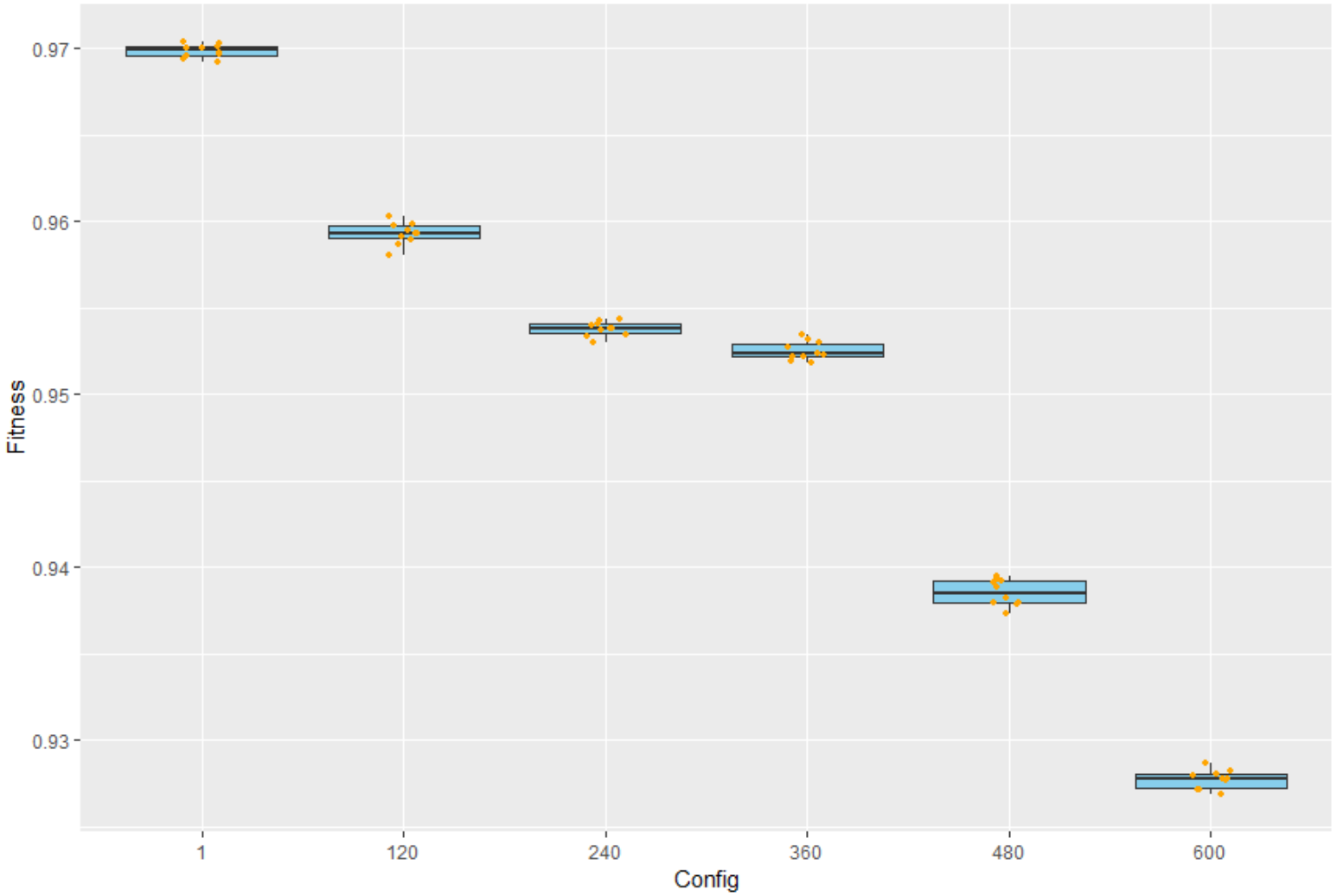


Figure 12: Chapter 5 - Box-Plot of Fitness for 6 sample parameter configurations

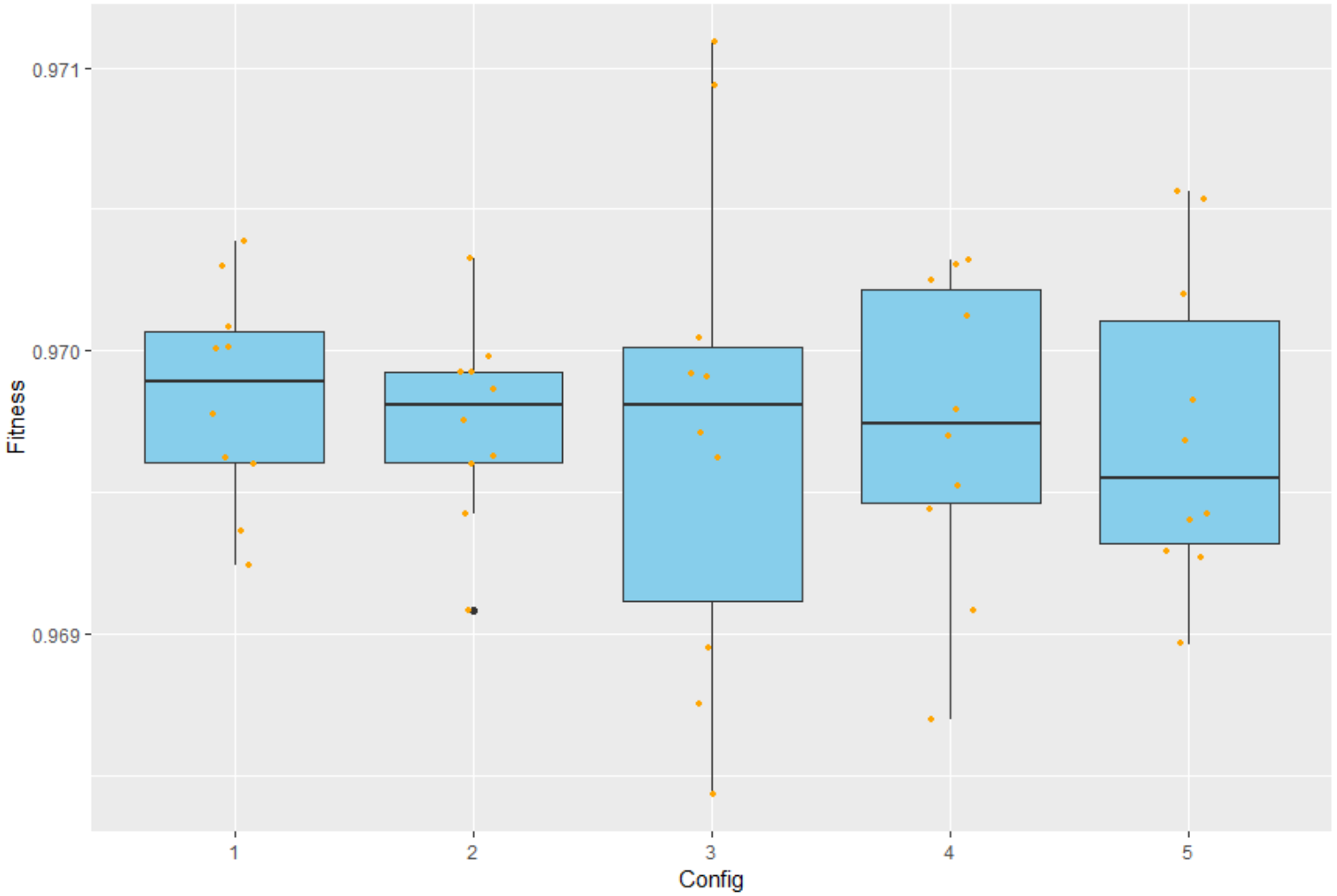


Figure 13: Chapter 5 - Box-Plot of Fitness for the best 5 parameter configurations

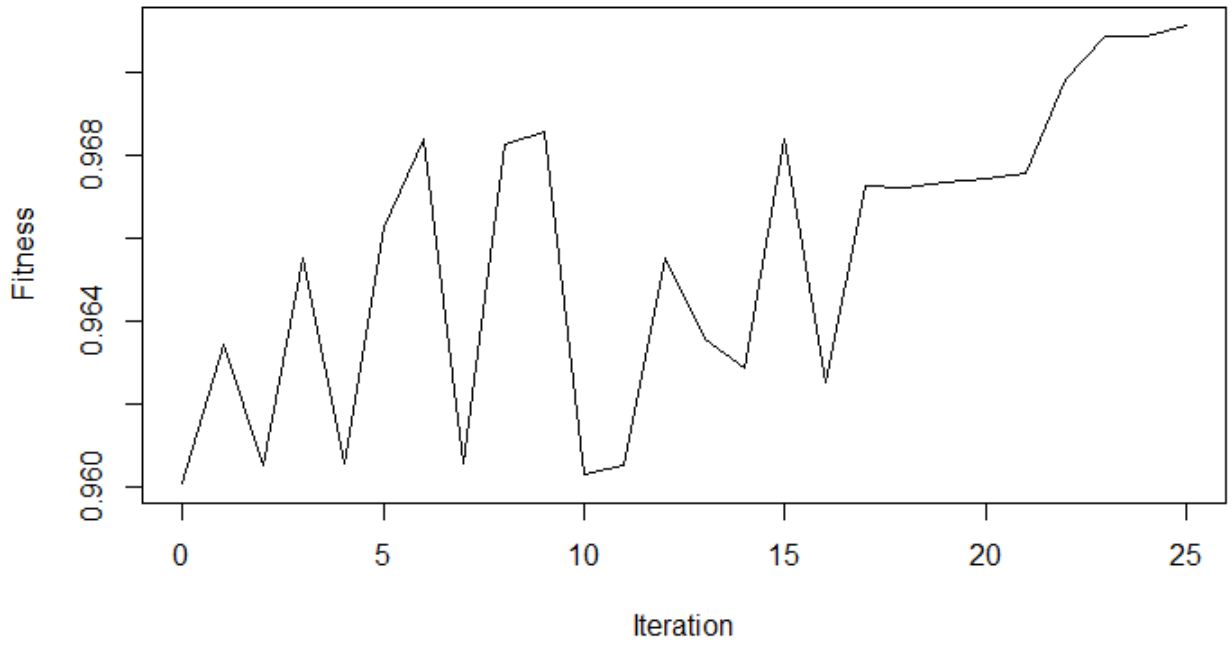


Figure 14: Chapter 5 - Fitness per Iteration of a Single Test.

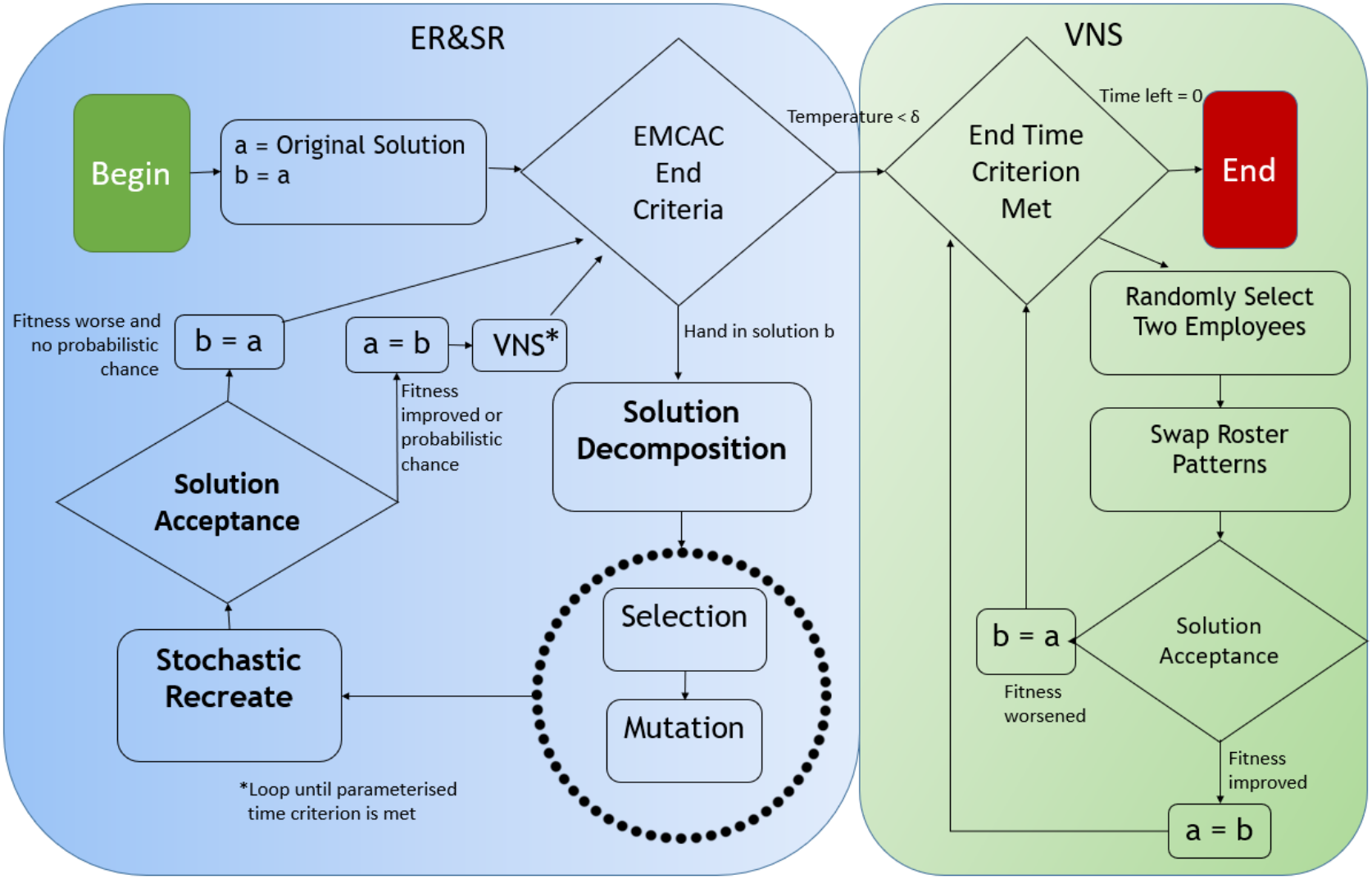


Figure 15: Chapter 6 - Diagrammatic Overview of ER&SR & VNS Hybrid

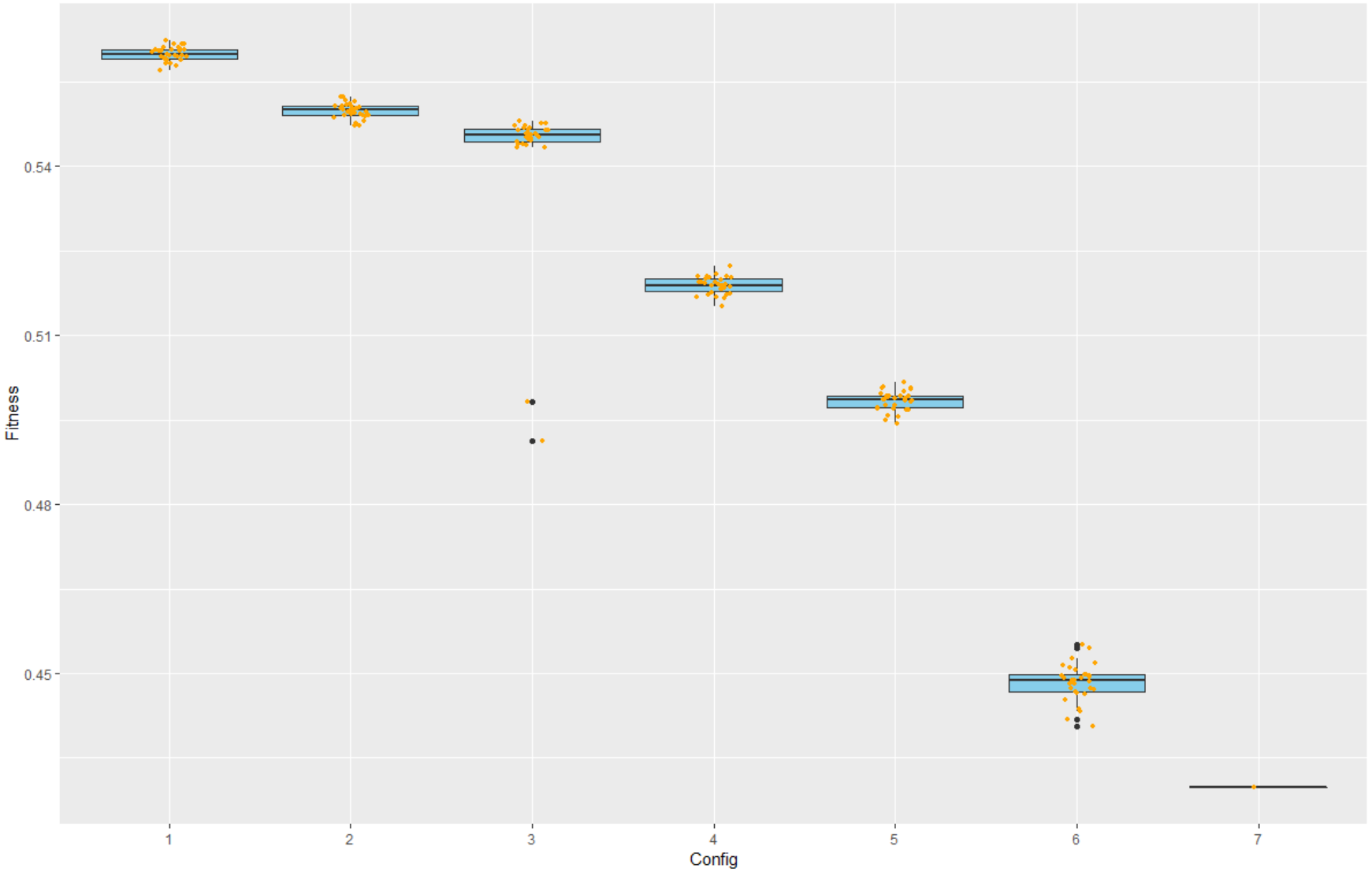


Figure 16: Chapter 6 - Box-Plot of Average Fitness (A-F) and Baseline Fitness (G)

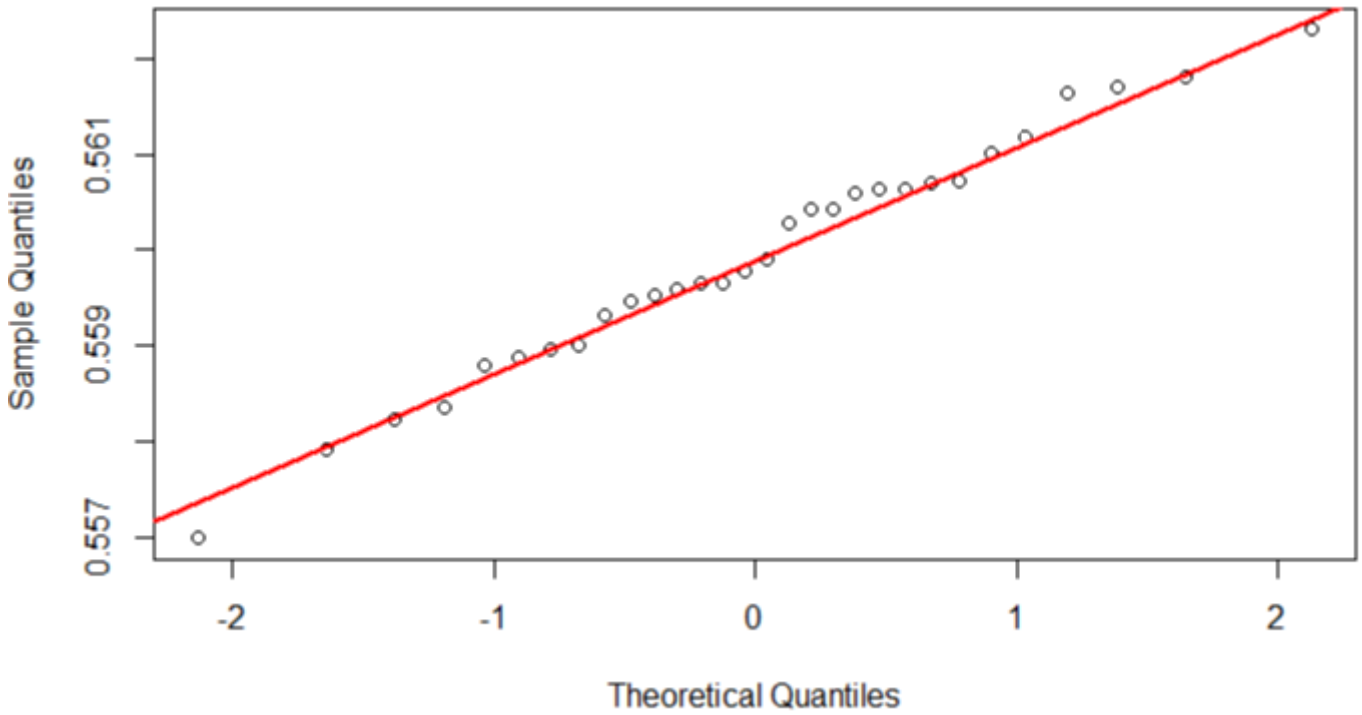


Figure 17: Chapter 6 - Q-Q Plot of Config. A

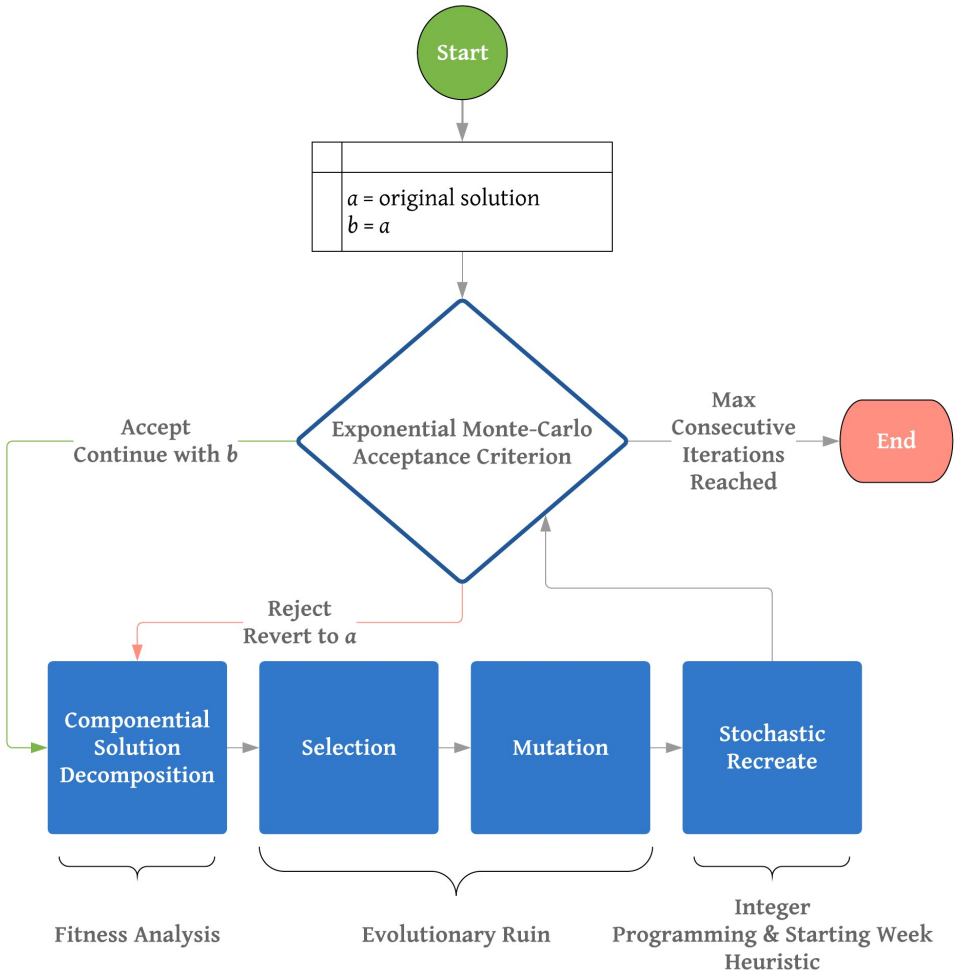


Figure 18: Chapter 7 - ERSRIP Flow Diagram

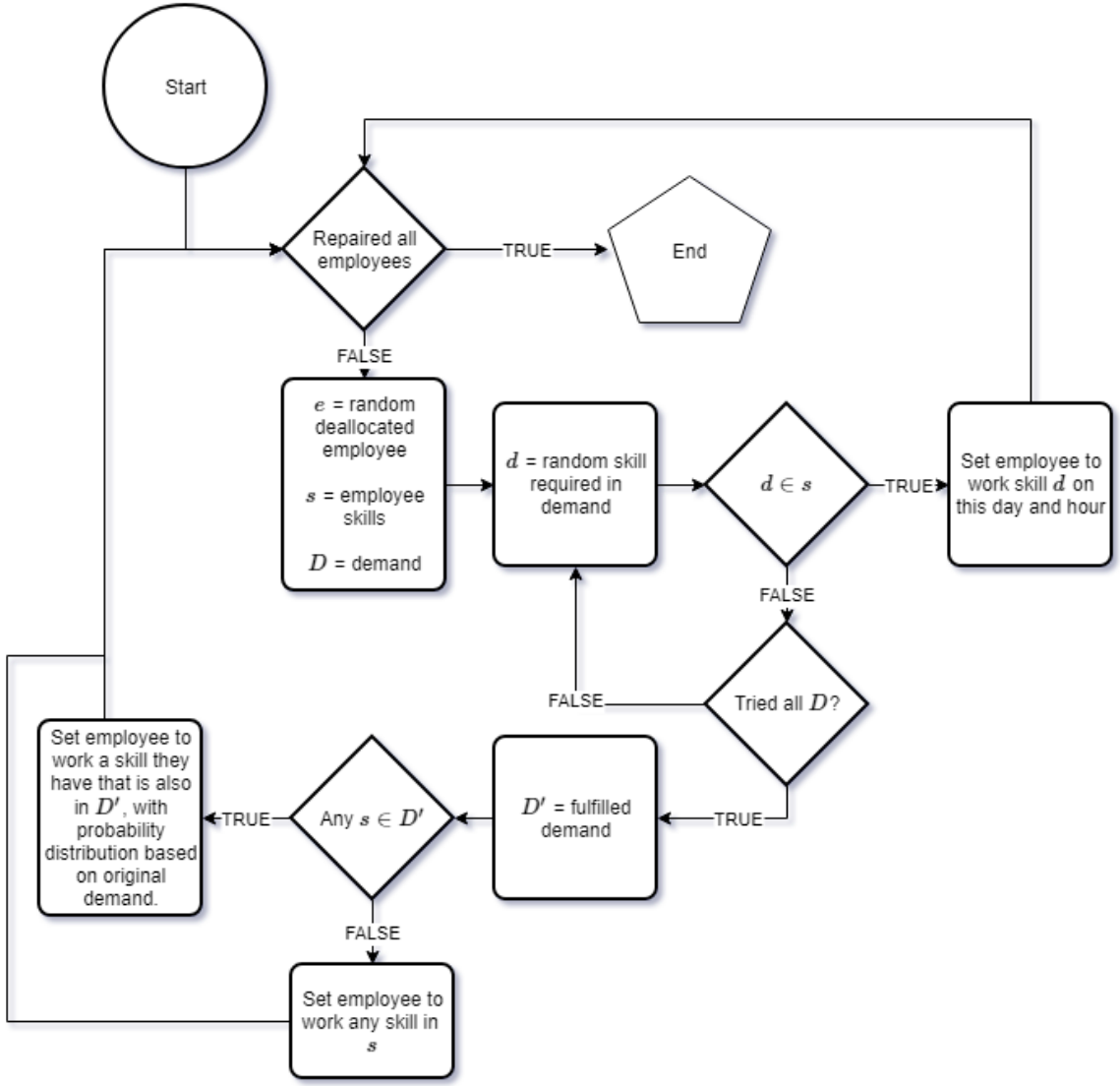


Figure 19: Chapter 7 - Metaheuristic Based Skill Allocations

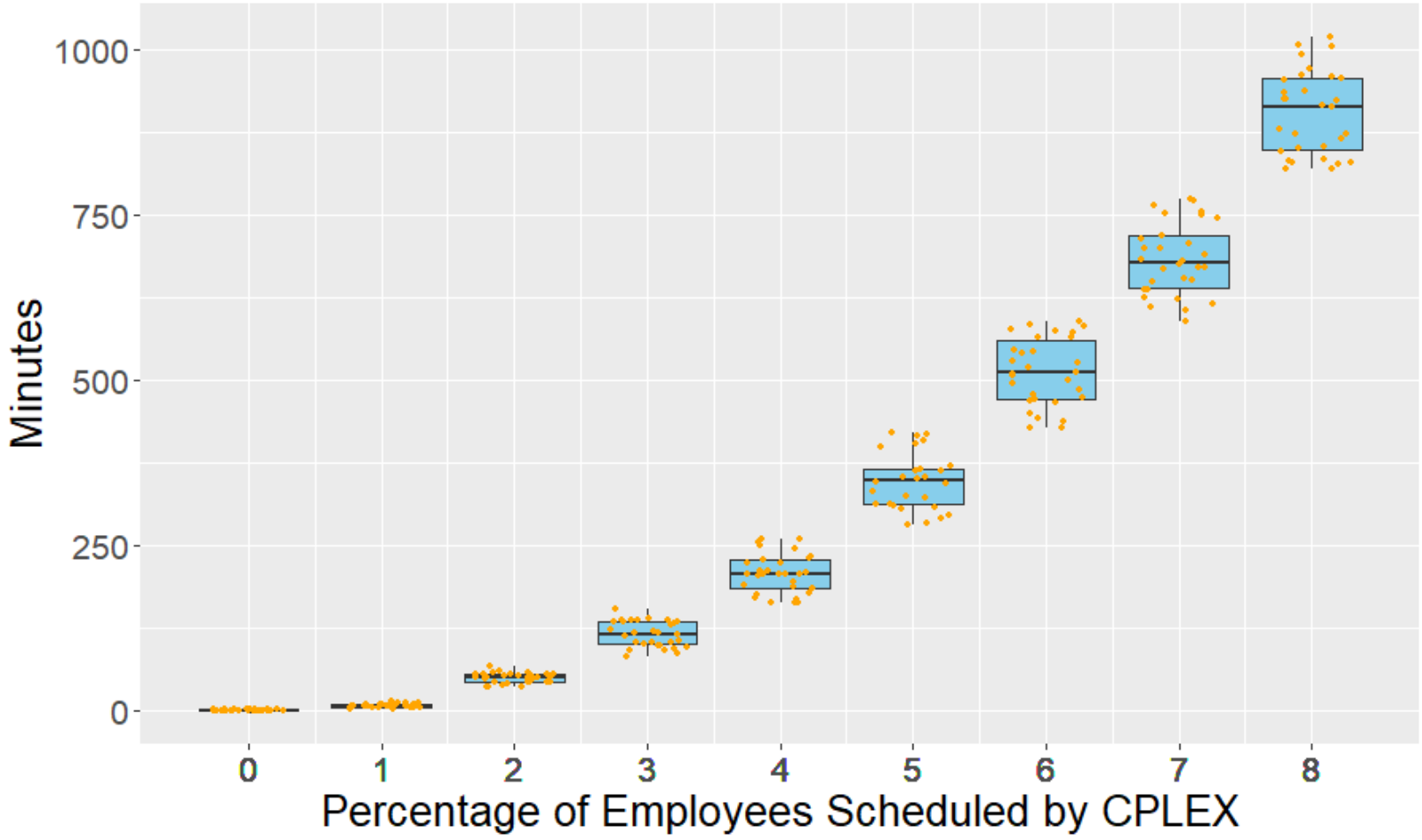


Figure 20: Chapter 7 - Run-time compared with Percentage of Employees CPLEX Tasked With Scheduling

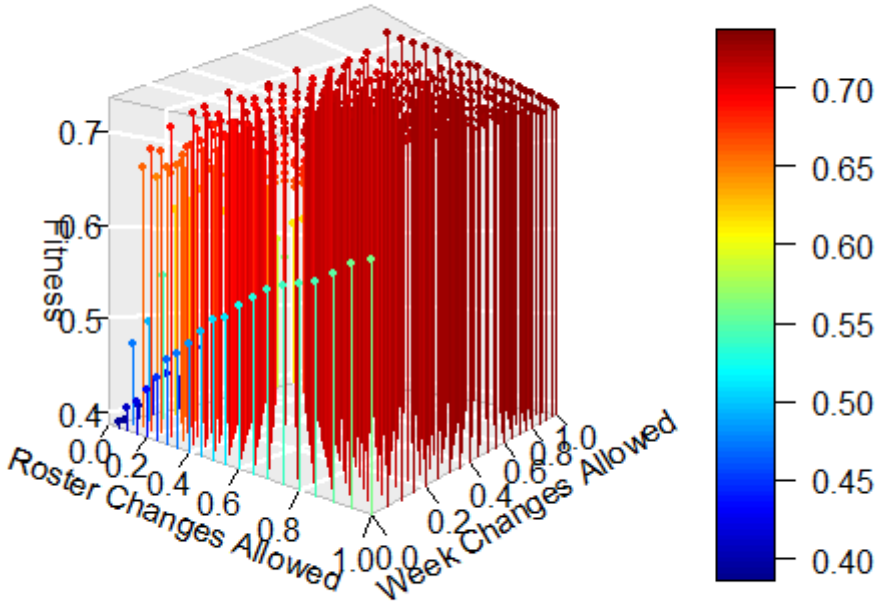


Figure 21: Chapter 7 - Run-time compared with Percentage of Employees CPLEX Tasked With Scheduling

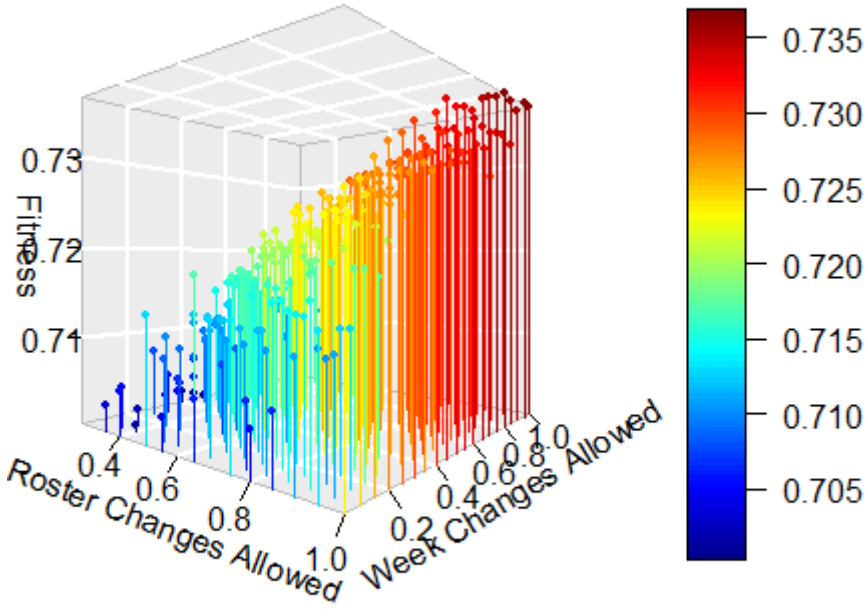


Figure 22: Chapter 7 - RP Changes Parameter Vs Starting Week Parameter (Outliers Removed)

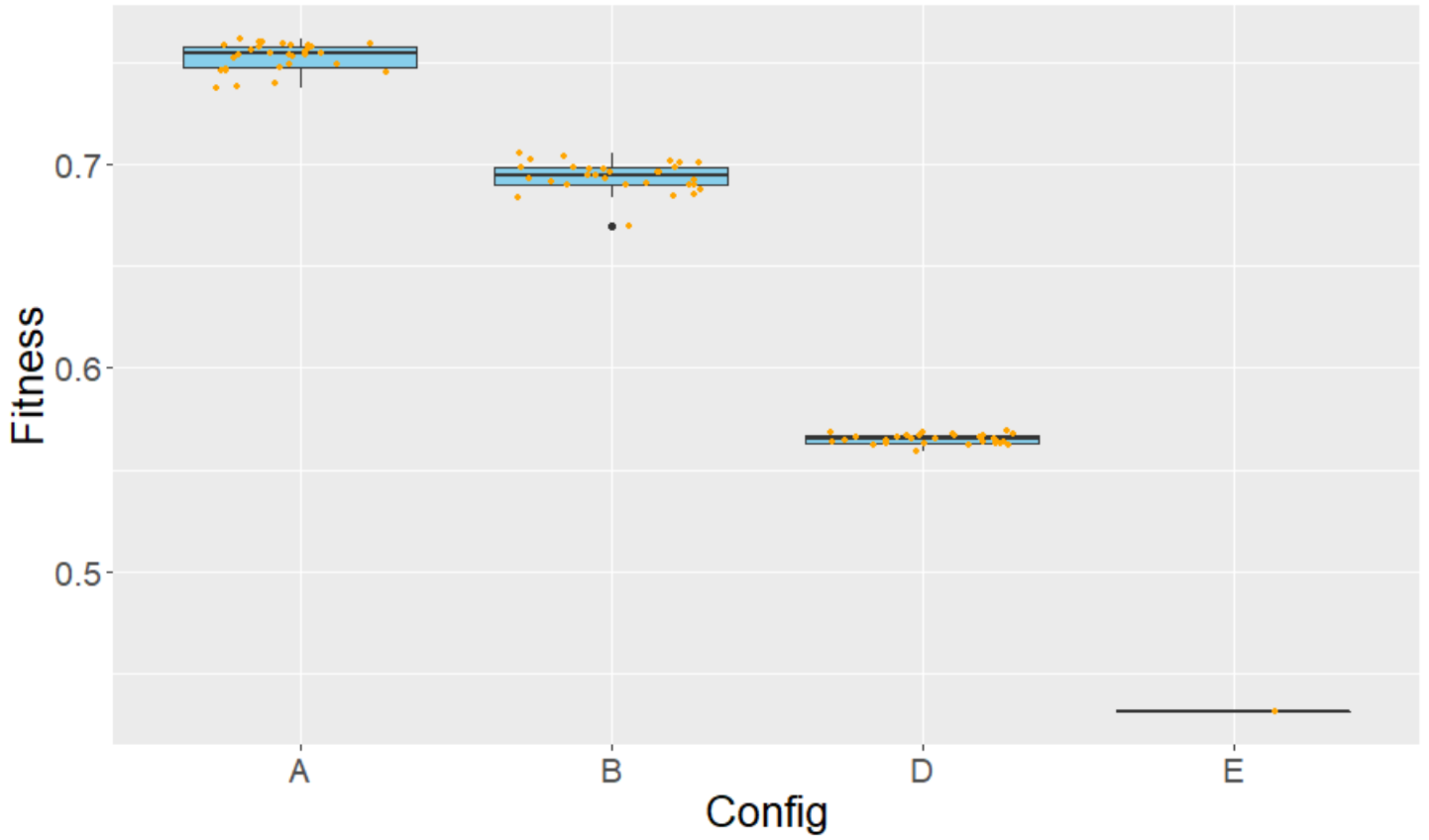


Figure 23: Chapter 7 - Boxplot of Each Config Tested 30x With Tuned Parameters

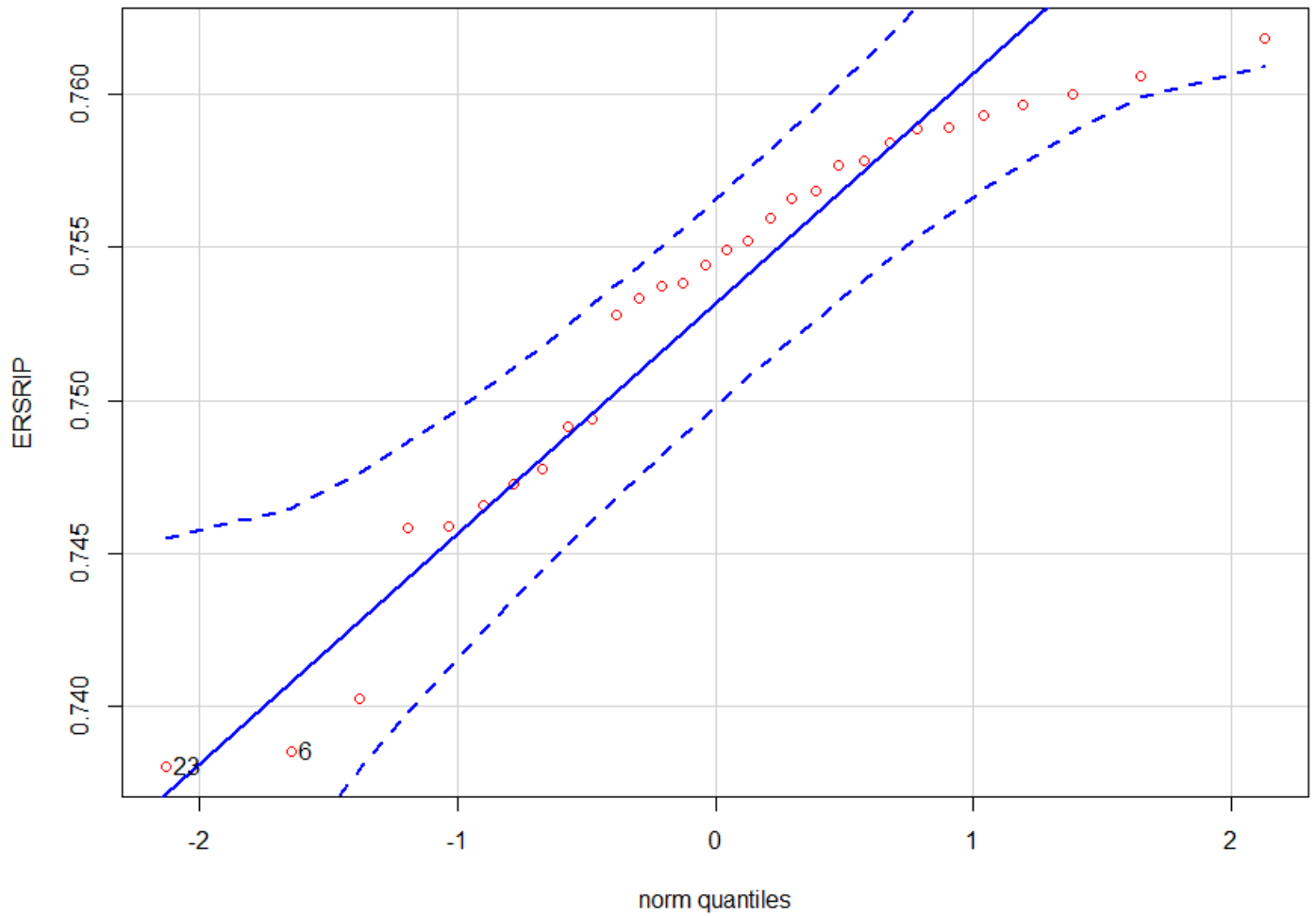


Figure 24: Chapter 7 - Q-Q Plot of Config A

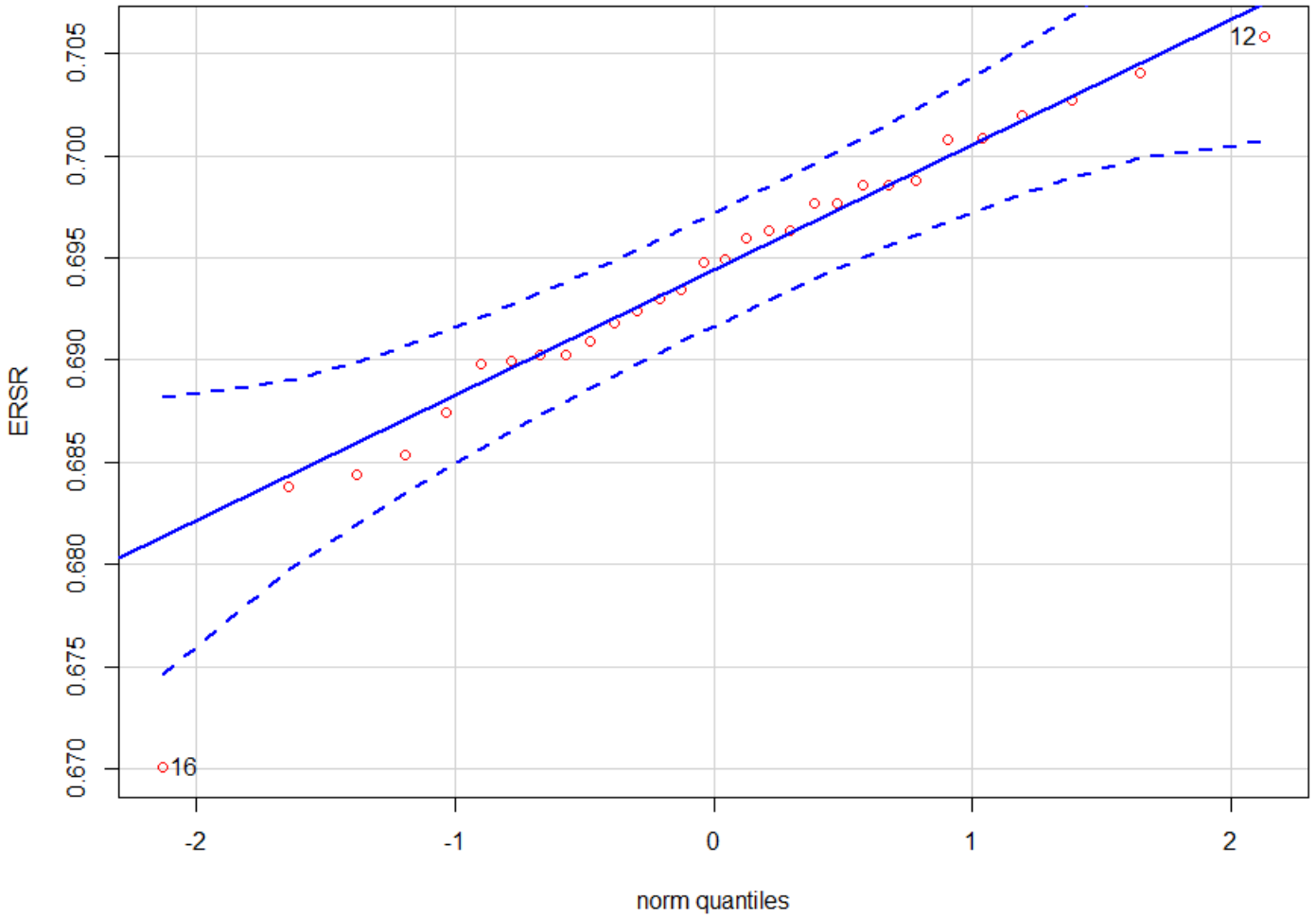


Figure 25: Chapter 7 - Q-Q Plot of Config B

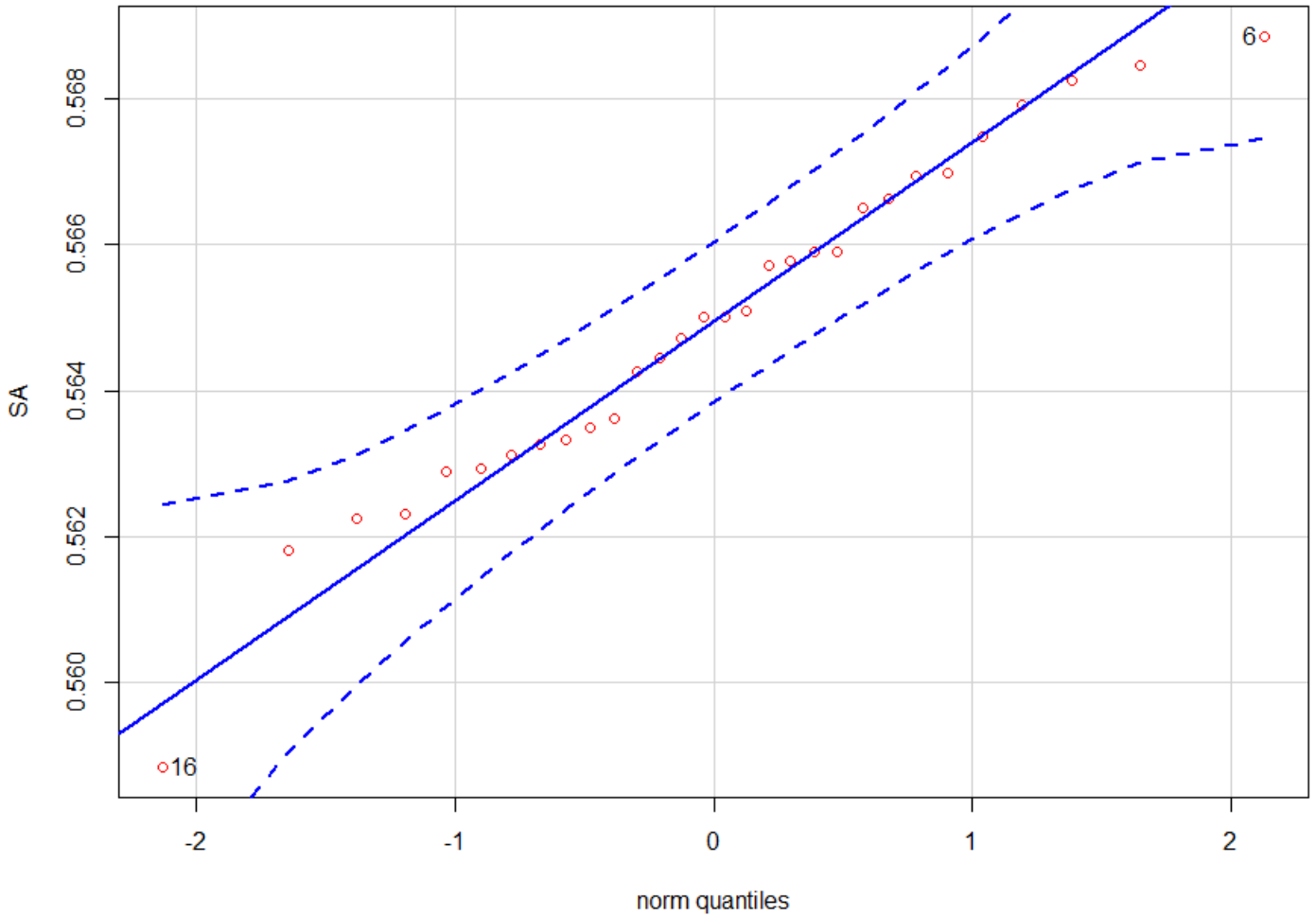


Figure 26: Chapter 7 - Q-Q Plot of Config. D

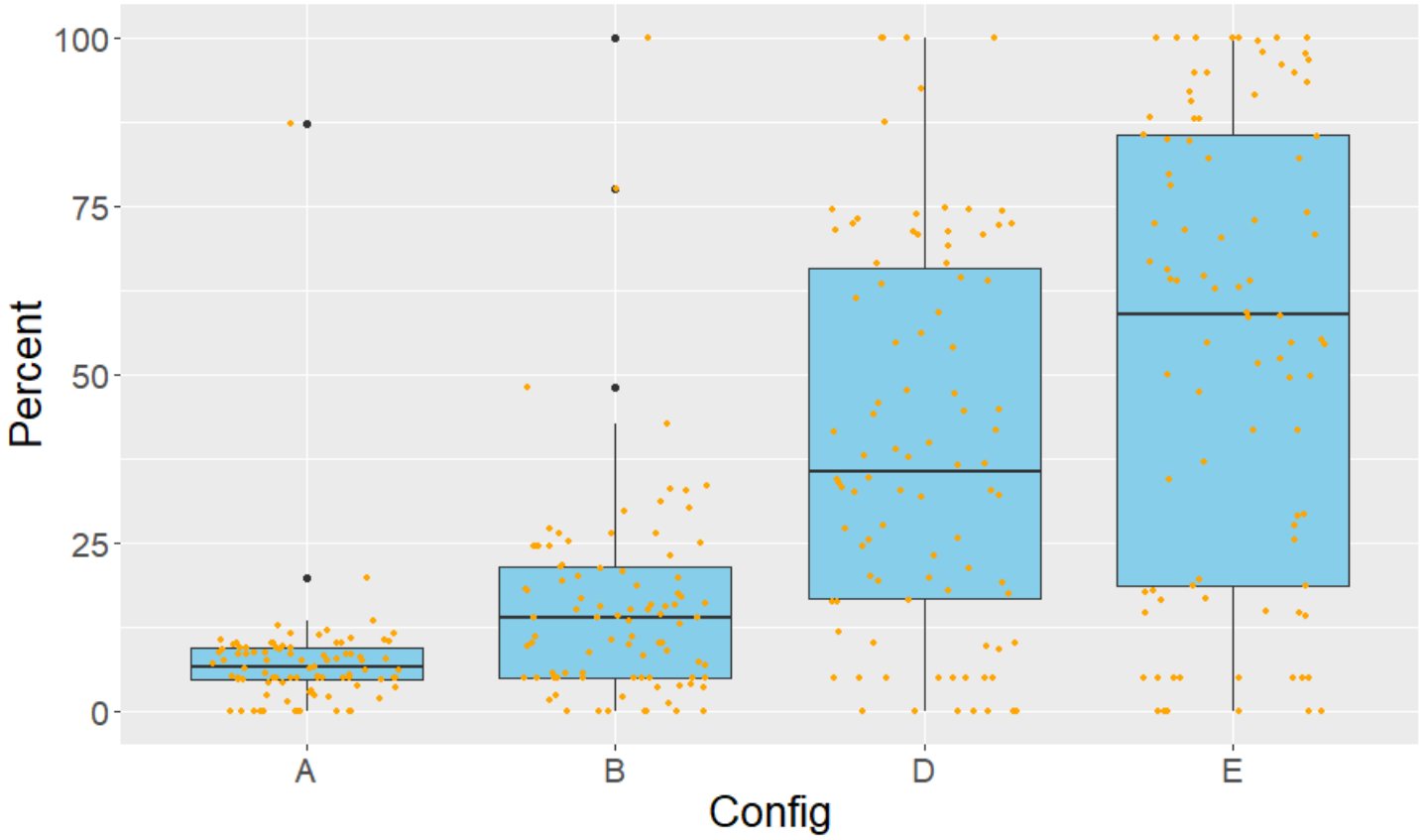


Figure 27: Chapter 7 - Average Absolute Percentage Difference Per Config

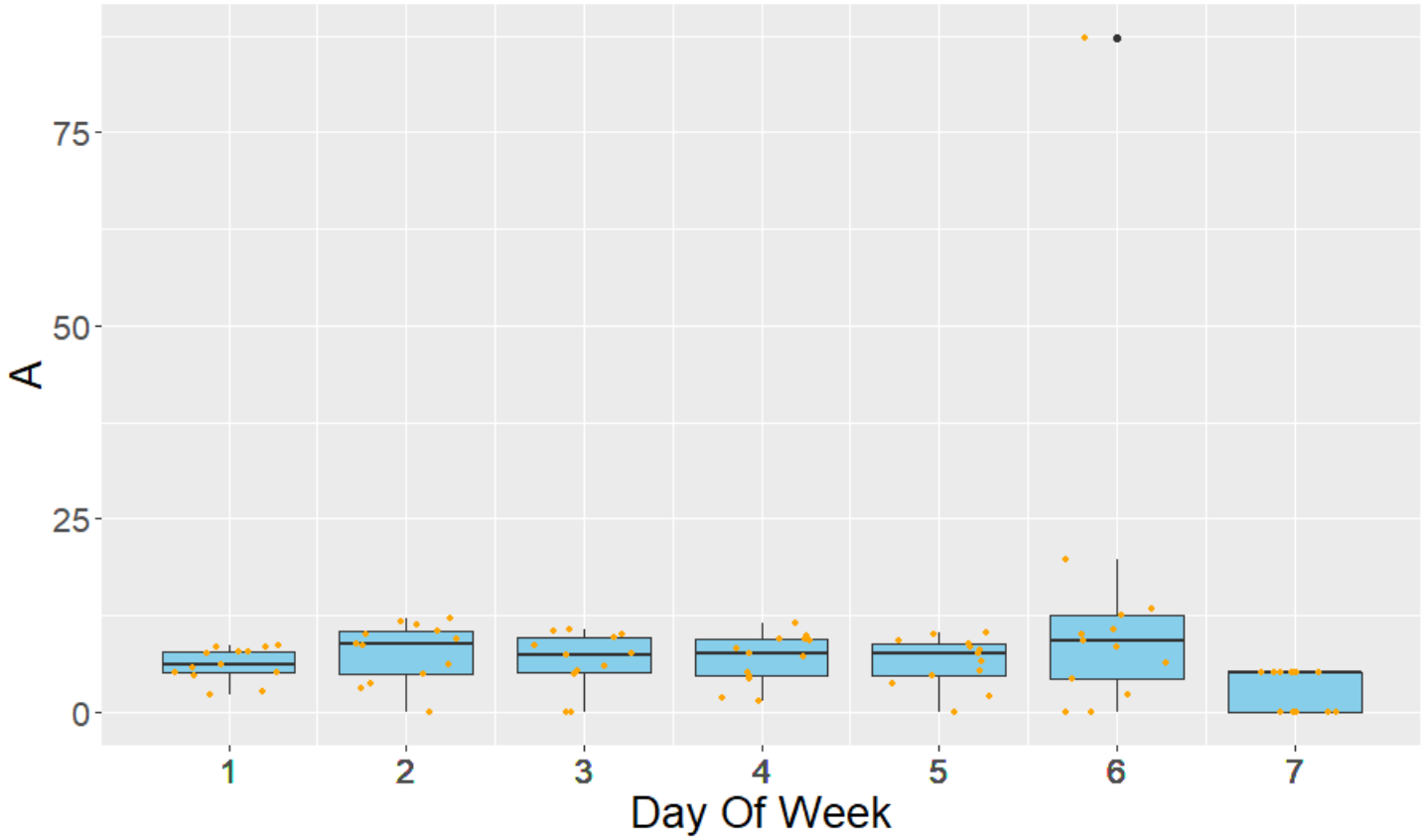


Figure 28: Chapter 7 - Day Of Week Absolute Percentage Difference - Config A

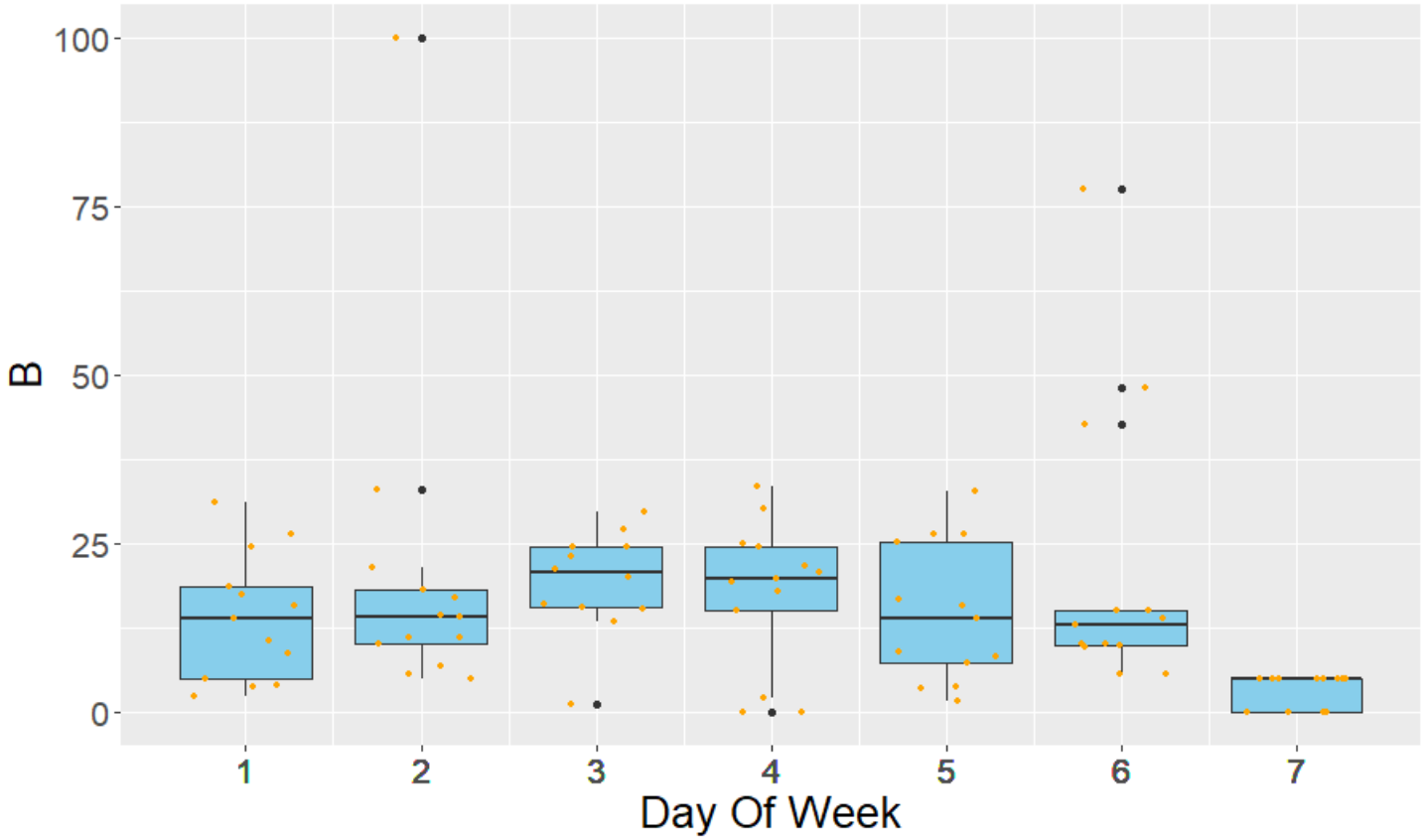


Figure 29: Chapter 7 - Day Of Week Absolute Percentage Difference - Config B

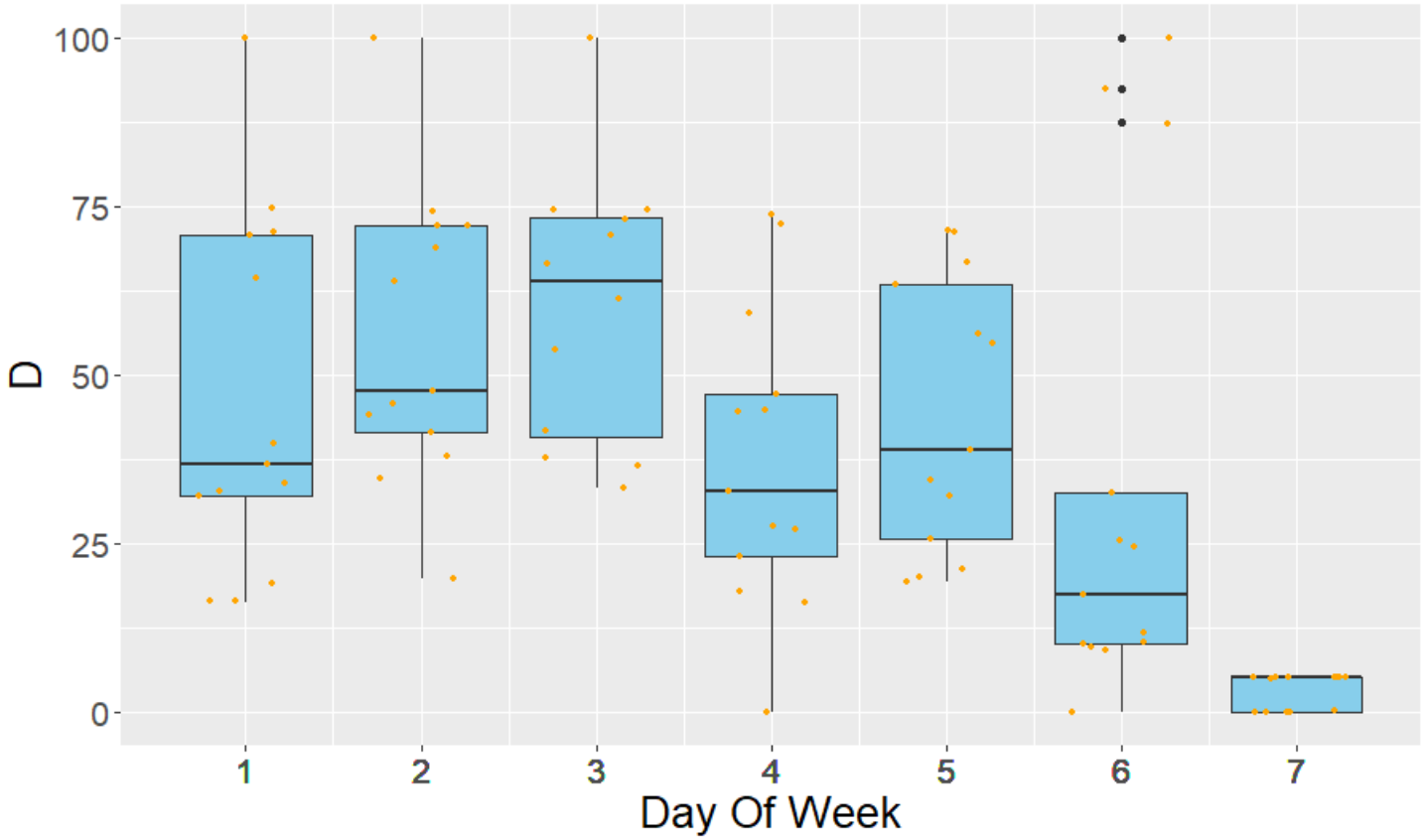


Figure 30: Chapter 7 - Day Of Week Absolute Percentage Difference - Config D

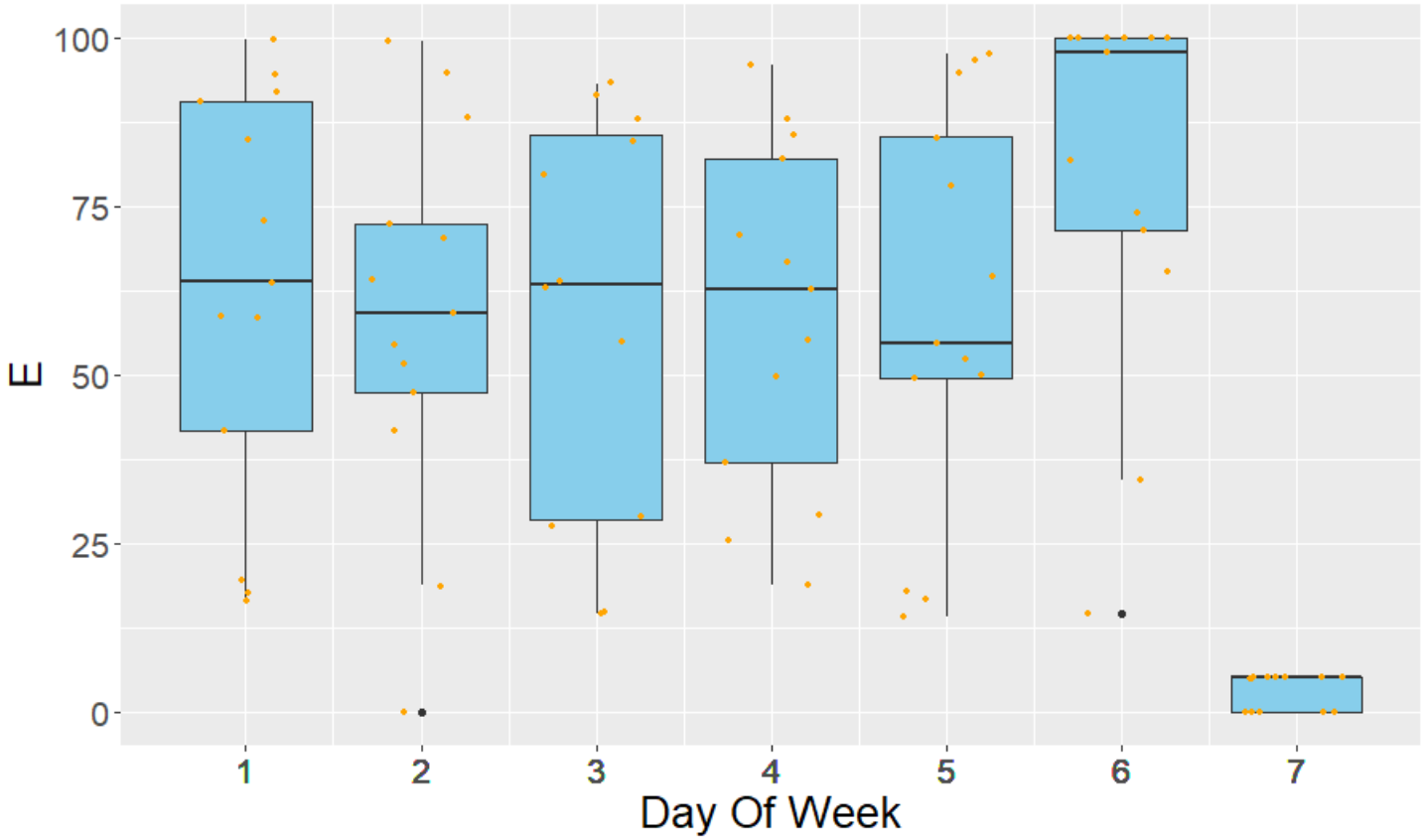


Figure 31: Chapter 7 - Day Of Week Absolute Percentage Difference - Config E

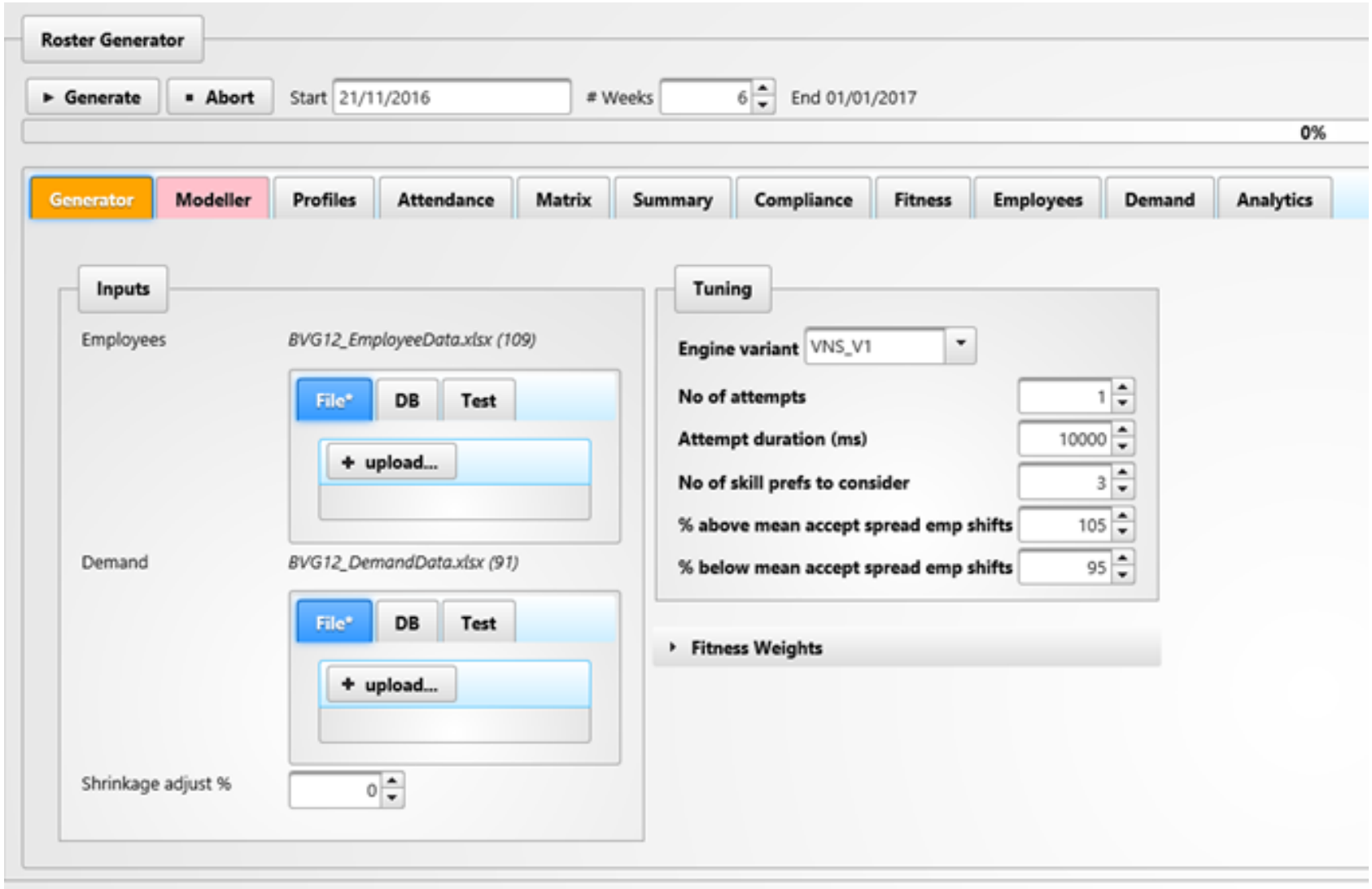


Figure 32: Chapter 8 - Managerial Initialisation Screen

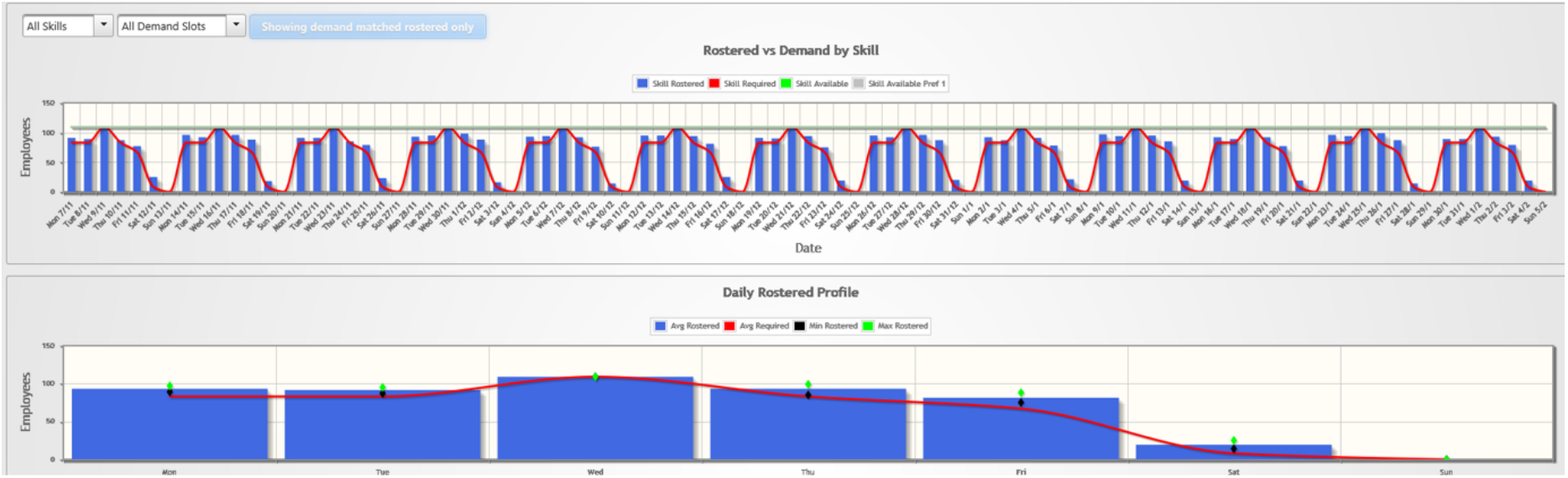
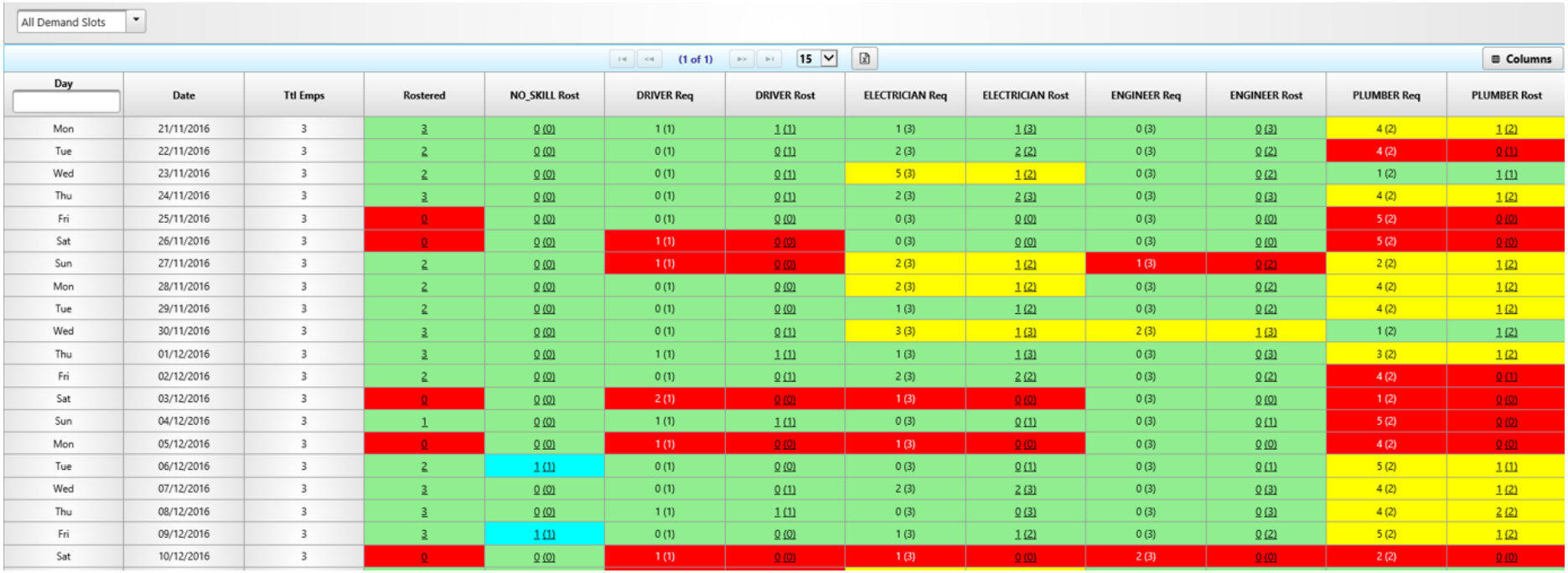


Figure 33: Chapter 8 - Managerial Schedule Overview Screen



Day	Date	Ttl Emps	Rostered	NO_SKILL Rost	DRIVER Req	DRIVER Rost	ELECTRICIAN Req	ELECTRICIAN Rost	ENGINEER Req	ENGINEER Rost	PLUMBER Req	PLUMBER Rost
Mon	21/11/2016	3	3	0 (0)	1 (1)	1 (1)	1 (3)	1 (3)	0 (3)	0 (3)	4 (2)	1 (2)
Tue	22/11/2016	3	2	0 (0)	0 (1)	0 (1)	2 (3)	2 (2)	0 (3)	0 (2)	4 (2)	0 (1)
Wed	23/11/2016	3	2	0 (0)	0 (1)	0 (1)	5 (3)	1 (2)	0 (3)	0 (2)	1 (2)	1 (1)
Thu	24/11/2016	3	3	0 (0)	0 (1)	0 (1)	2 (3)	2 (3)	0 (3)	0 (3)	4 (2)	1 (2)
Fri	25/11/2016	3	0	0 (0)	0 (1)	0 (0)	0 (3)	0 (0)	0 (3)	0 (0)	5 (2)	0 (0)
Sat	26/11/2016	3	0	0 (0)	1 (1)	0 (0)	0 (3)	0 (0)	0 (3)	0 (0)	5 (2)	0 (0)
Sun	27/11/2016	3	2	0 (0)	1 (1)	0 (0)	2 (3)	1 (2)	1 (3)	0 (2)	2 (2)	1 (2)
Mon	28/11/2016	3	2	0 (0)	0 (1)	0 (0)	2 (3)	1 (2)	0 (3)	0 (2)	4 (2)	1 (2)
Tue	29/11/2016	3	2	0 (0)	0 (1)	0 (0)	1 (3)	1 (2)	0 (3)	0 (2)	4 (2)	1 (2)
Wed	30/11/2016	3	3	0 (0)	0 (1)	0 (1)	3 (3)	1 (3)	2 (3)	1 (3)	1 (2)	1 (2)
Thu	01/12/2016	3	3	0 (0)	1 (1)	1 (1)	1 (3)	1 (3)	0 (3)	0 (3)	3 (2)	1 (2)
Fri	02/12/2016	3	2	0 (0)	0 (1)	0 (1)	2 (3)	2 (2)	0 (3)	0 (2)	4 (2)	0 (1)
Sat	03/12/2016	3	0	0 (0)	2 (1)	0 (0)	1 (3)	0 (0)	0 (3)	0 (0)	1 (2)	0 (0)
Sun	04/12/2016	3	1	0 (0)	1 (1)	1 (1)	0 (3)	0 (1)	0 (3)	0 (1)	5 (2)	0 (0)
Mon	05/12/2016	3	0	0 (0)	1 (1)	0 (0)	1 (3)	0 (0)	0 (3)	0 (0)	4 (2)	0 (0)
Tue	06/12/2016	3	2	1 (1)	0 (1)	0 (0)	0 (3)	0 (1)	0 (3)	0 (1)	5 (2)	1 (1)
Wed	07/12/2016	3	3	0 (0)	0 (1)	0 (1)	2 (3)	2 (3)	0 (3)	0 (3)	4 (2)	1 (2)
Thu	08/12/2016	3	3	0 (0)	1 (1)	1 (1)	0 (3)	0 (3)	0 (3)	0 (3)	4 (2)	2 (2)
Fri	09/12/2016	3	3	1 (1)	0 (1)	0 (0)	1 (3)	1 (2)	0 (3)	0 (2)	5 (2)	1 (2)
Sat	10/12/2016	3	0	0 (0)	1 (1)	0 (0)	1 (3)	0 (0)	2 (3)	0 (0)	2 (2)	0 (0)

Figure 34: Chapter 8 - Managerial Schedule Breakdown Screen

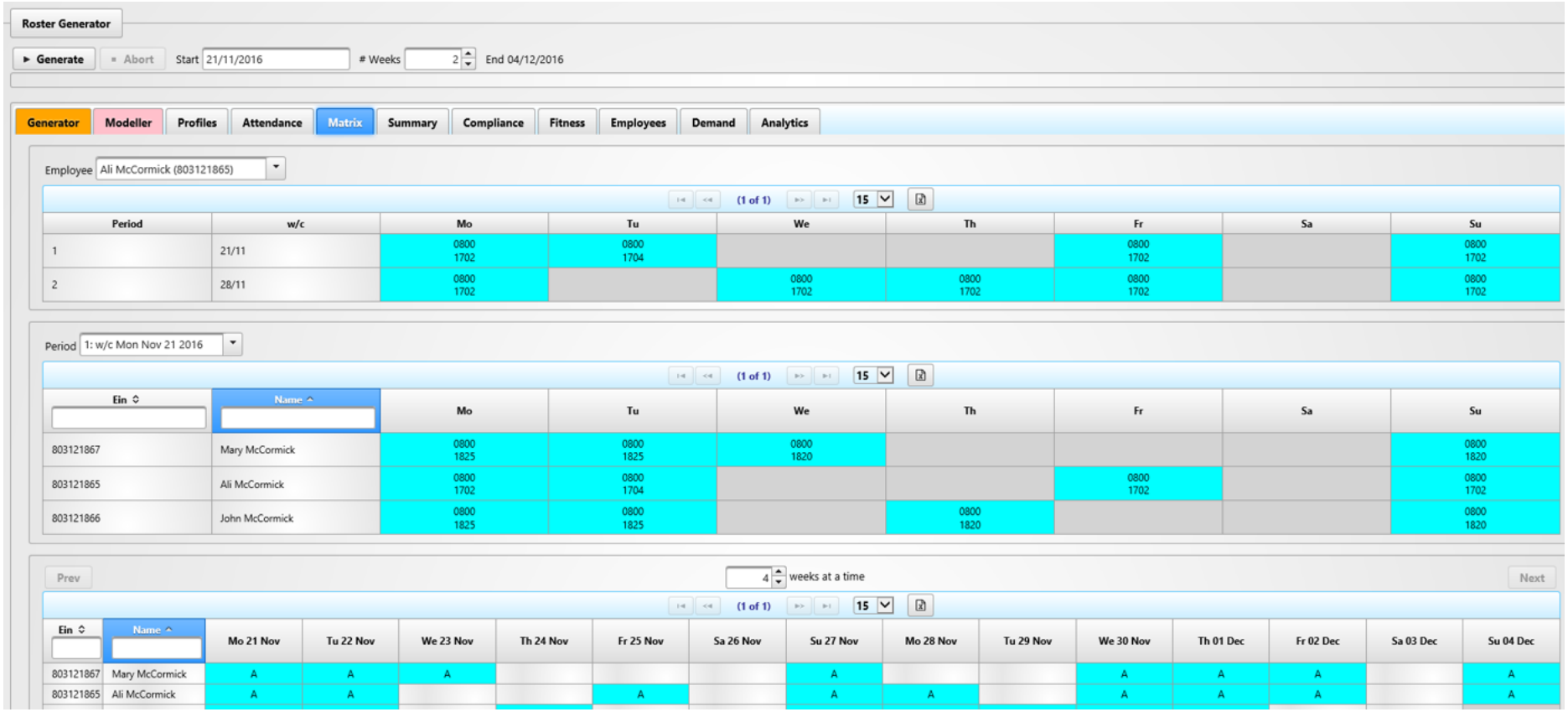


Figure 35: Chapter 8 - Employee View

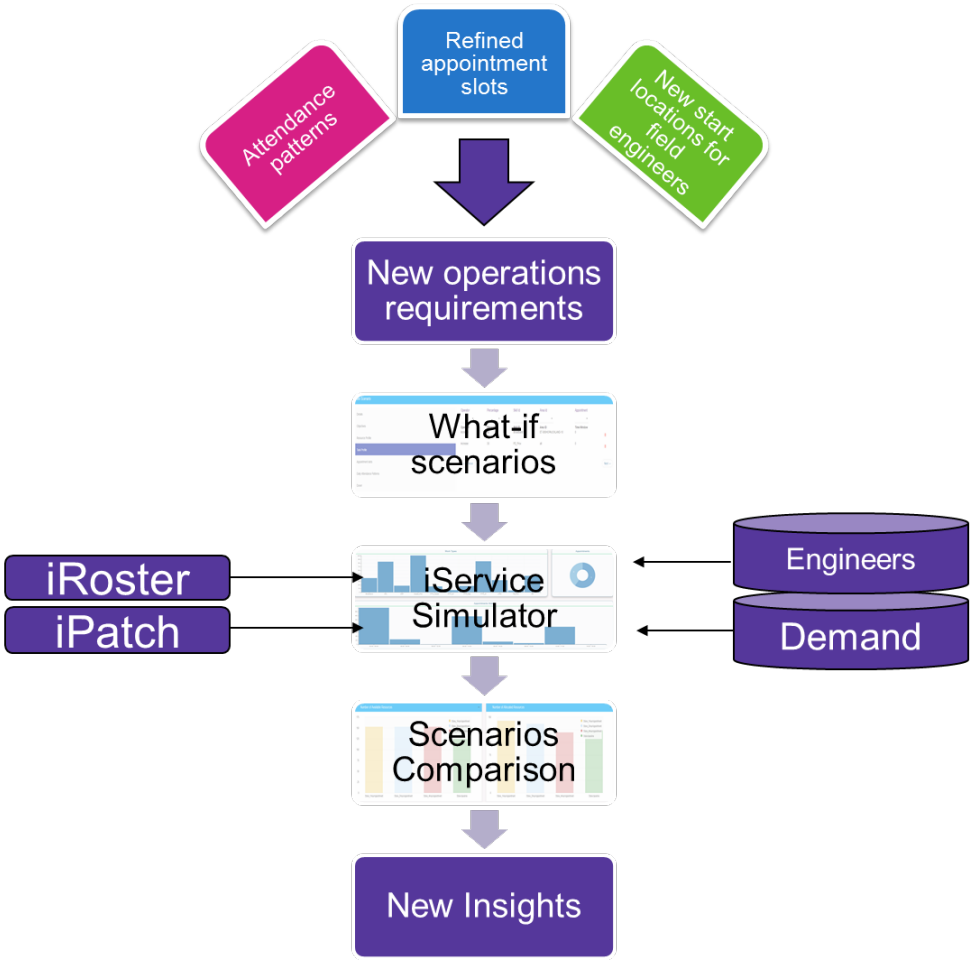


Figure 36: Chapter 8 - Cross-Project Data Flow

Figure 37: Chapter 8 - Simulation Analytics

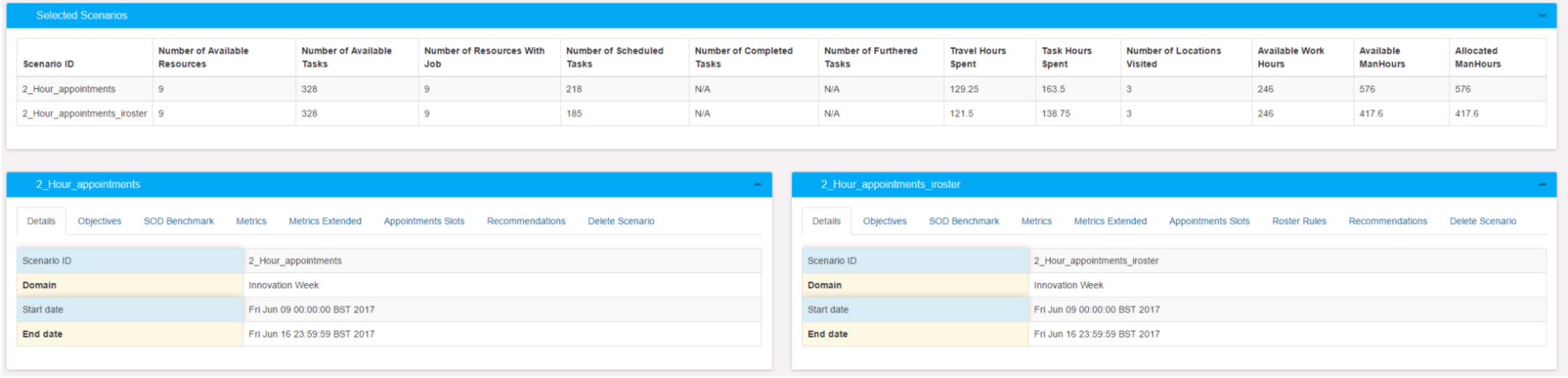


Figure 38: Chapter 8 - Additional Analytics for Simulator

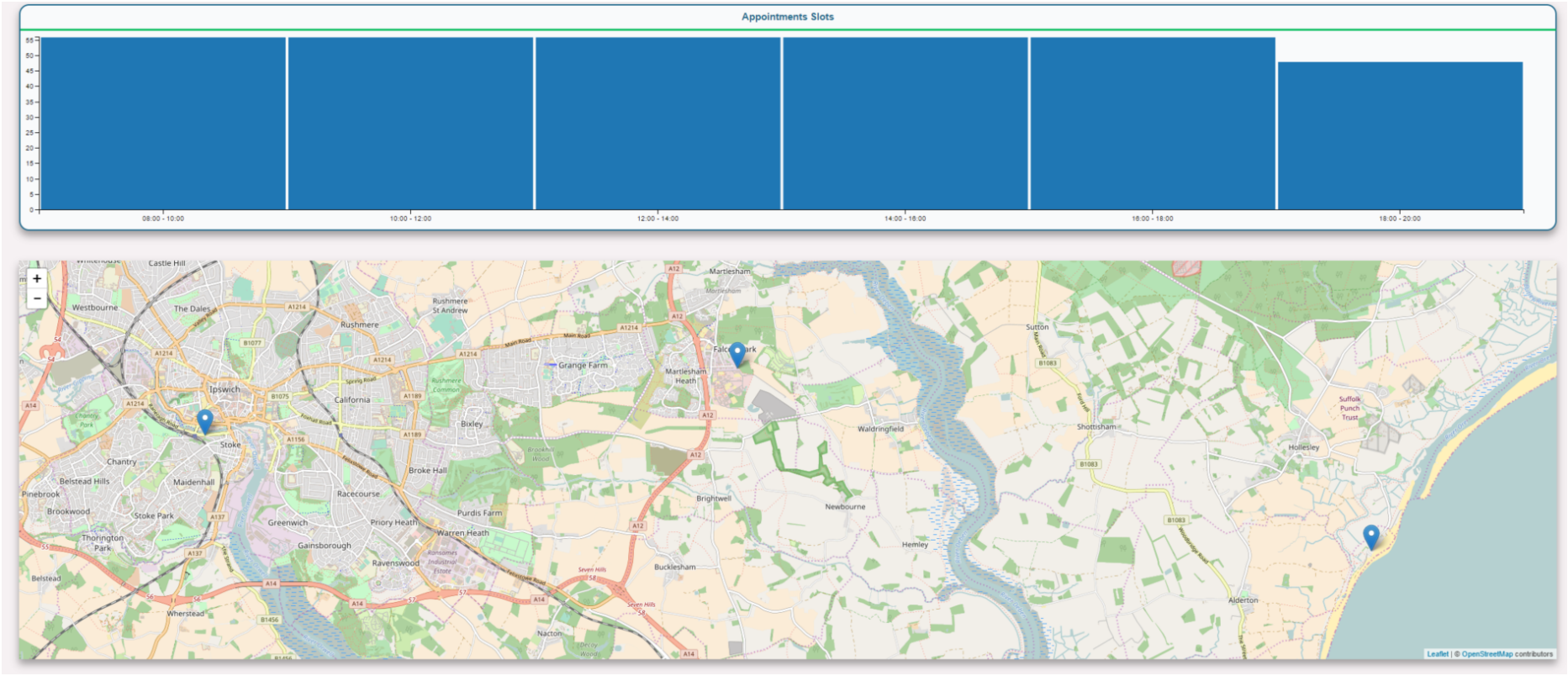


Figure 39: Chapter 8 - Geographic Visual Screenshot